Docs

Table of contents

1. Introduction

1.1 What is Pantavisor?

Pantavisor is a framework for building embedded Linux systems with lightweight LXC containers to create software-defined products for IoT.

1.1.1 Introducing Pantavisor Linux and Pantacor Hub

Pantavisor Linux is a framework for building containerized embedded systems. It implements a version of lightweight Linux Containers (LXC), and is the easiest way to build a software-defined Linux Embedded product. With your Linux distribution or custom-made firmware userland in containers, your system gets the benefits of portable container lifecycle management without having to replace your distribution or worse, the board.

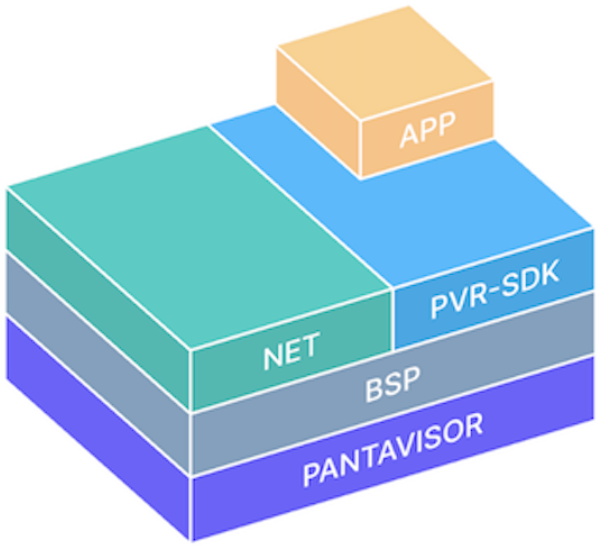
Why Pantavisor Linux?

Pantavisor Linux creates the building blocks for a universal Linux non-OS that frees up your team so they can focus on features and services. It provides a simple way to deploy and manage your containerized embedded OS components across millions of devices in a portable and reproducible manner.

Pantavisor Linux can also convert Docker containers into LXC format to run on all types of devices and boards, including those with the smallest resource footprint. Turn firmware, the OS, and even the BSP into portable, cloud native, containerized building blocks that can be shared and deployed transactionally over the air.

Manage your system with containerized building blocks

Pantavisor Linux containerizes your firmware, the OS, the networking and even the Board Support Package (BSP) making them modular and portable building blocks that can be shared and managed atomically over the air. With everything containerized on the device, it's simple to mix and match these components to build new distros and maintain any customizations you may have that are specific to your use case without having to replace your distro or worse, the entire board.



These are the base reusable building blocks that Pantavisor Linux modularizes and turns into containers:

- **Board Support Packages (BSPs)**: kernel, modules, and firmware.
- **System Middleware Containers**: you can choose to package your monolithic distro middleware in one container or build your middleware into more fine-grained units.
- **Apps**: Run them as LXC or Linux containers. Can also convert and run Docker containers.
- **Configuration**: system level configuration

1.1.2 What is Pantacor Hub?

Pantacor Hub is the open source device state management system. You can think of it as a cross between an image sharing repository for apps and devices as well as a device system revision repository. The hub allows you to share images and device data between team members or other users. It also manages the atomic revisions of the device state for over the air updates across device fleets and you can configure device and application meta-data.

1.2 Specs

Pantavisor is written in C, with an emphasis on the embedded Linux ecosystem. It is meant to be a single-binary init system that boots directly from the kernel and becomes the first process to run, which then brings up the rest of the system as a set of well-defined micro-containers.

We aim to bring a full set of containers functionality into the single binary, while keeping its size as small as possible to make sure it can cover the low-spec end of the market. Depending on the functions that are built, the size can vary, but the average size for a fully functional system puts the Pantavisor binary at around 350 Kilobytes (as compressed initial ramdisk).

Pantavisor uses pure Linux container technology, for which we use parts of the LXC suite as a library to wrap around the basic building blocks of containers. Due to the fact that LXC is also a pure C project, we are able to keep the overall footprint of Pantavisor quite small.

1.3 Licensing

Pantavisor is MIT licensed, which means tons of flexibility for commercial projects, without the headaches of a pure GPL codebase. The container runtime plugin system makes a clear separation of the provider library and Pantavisor itself.

2. Get Started

2.1 Where to Start?

This page will help you on deciding how to navigate this documentation.

2.1.1 Getting Started

If you are about to embark on a new project or you just want to give a taste to Pantavisor, you will first have to decide which device or devices are you going to use, and which path are you going to choose for managing them.

- How to Prepare your Device
- How to Manage your Device

2.1.2 Building from Source Code

If you are an application developer, you have already setup your devices, and you can manage them, then you are ready to move on to the next stage and focus on what is important to you: creating the application level.

If you want to make infrastructure changes in the BSP, including Pantavisor, then you can follow by building its source code.

- How to Build Apps
- How to Build Pantavisor

2.1.3 Understanding Pantavisor

For a deeper understanding on Pantavisor, you can first check the technical overview, which can be read from top to bottom to understand all its features.

There are also quick reference pages, meant to be easy to navigate and informative.

- Technical Overview
- References

2.2 Sign up to Pantacor Hub

> **Note**
>
> This is only necessary if you want to remotely control your devices from Pantacor Hub. If you prefer to follow the local experience path, then just skip this page.

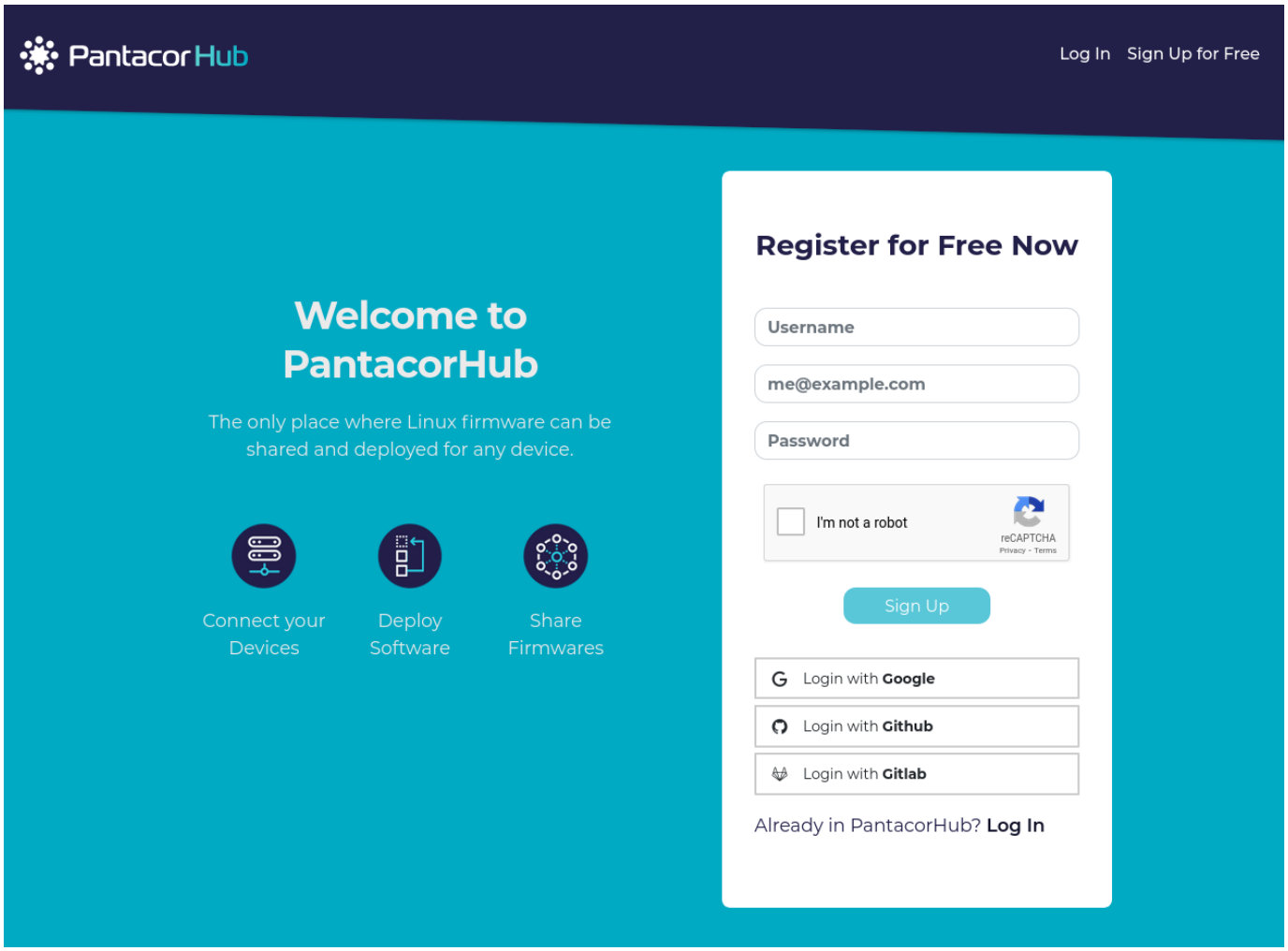
The first thing you need to do to interact with https://hub.pantacor.com is to register a user account. A user account gives you access to the full API, including the object store, and also grants you access to its dashboard.

You can sign up in the Pantacor Hub web interface, with the pvr tool from the host or with the pantabox tool from the device.

> **Note**
>
> After registering your account, make sure to follow the instruction in the verification email. Your account is not ready for use until it is verified.

2.2.1 Registering on the web

Visit the Pantacor Hub starting page at https://hub.pantacor.com and follow the sign-up process. You have the option of signing up with your email address or with your Google, GitHub or GitLab account.



2.2.2 Registering with pvr

You can use the following pvr command to register a user:

```
pvr register -u youruser -p yourpassword -e your@email.tld
```

This will generate a json response with the server-generated part of the credentials:

```
2017/06/19 11:08:43 Registration Response: {
  "id": "5947949b85188a000c143c2e",
  "type": "USER",
  "email": "your@email.tld",
  "nick": "youruser",
  "prn": "prn:::accounts:/5947949b85188a000c143c2e",
  "password": "yourpassword",
  "time-created": "2017-06-19T09:08:43.767224118Z",
  "time-modified": "2017-06-19T09:08:43.767224118Z"
}
```

2.2.3 Registering with pantabox

To register through pantabox, you must have already downloaded an initial precompiled image and flashed your device. So, firstly, you will have to make it ready.

After you have logged in your device, register and claim the device using the pantabox command.

```
pantabox-claim
```

3. How to Install Pantavisor

3.1 Choose a Device

If this is your first time and you want to try Pantavisor, we recommend you to start with a `Raspberry Pi 3 b+` or `Raspberry Pi 4`. This is the simplest and quickest way to get yourself started. Maybe you are thinking more about smaller devices with lower specs, and we support those too, but we chose Raspberry Pi as the prototyping board for our newcomers. If you prefer, you can move on to other boards though, including emulated ones.

> **Note**
>
> By default, Pantavisor is meant to work on top of the Linux Kernel as the init system, and for this you will have to flash the board storage medium, which leads us to the non embedded option.

The non embedded option is to run Pantavisor as a daemon in your device. This is ideal if you want to run it in your already setup device, using any Linux distro of your choice, which could prove useful for prototyping without having to flash the board storage.

3.2 Raspberry Pi

3.2.1 Get Started

In this guide, we will guide you through the process of flashing your `Raspberry Pi` device with a ready-to-use pre-compiled image.

> **Note**
>
> Before starting this guide, we recommend to create a Pantacor Hub account for new users. However, there are different ways to manage your Pantavisor device, and a local experience that does not require registration is available.

Requirements

To follow this guide, you will need the following:

- `Raspberry Pi 3 B+` or `Raspberry Pi 4`
- A micro SD Card
- Micro SD Card Writer

> **Note**
>
> If you are missing any of these requirements, you can have a look at how to install other images, incluiding emulated ones, or at how to run Pantavisor as a deamon. Finally, as a last resource, you can also build your own images.

Tutorial Overview

1. Image setup: Download and flash our pre-compiled Pantavisor image on a micro SD card.
2. First boot: Insert the micro SD card in the board slot and boot it up.

After this, you will be able to manage your device both locally and remotely.

3.2.2 Image Setup

First step is to download and flash our pre-compiled image.

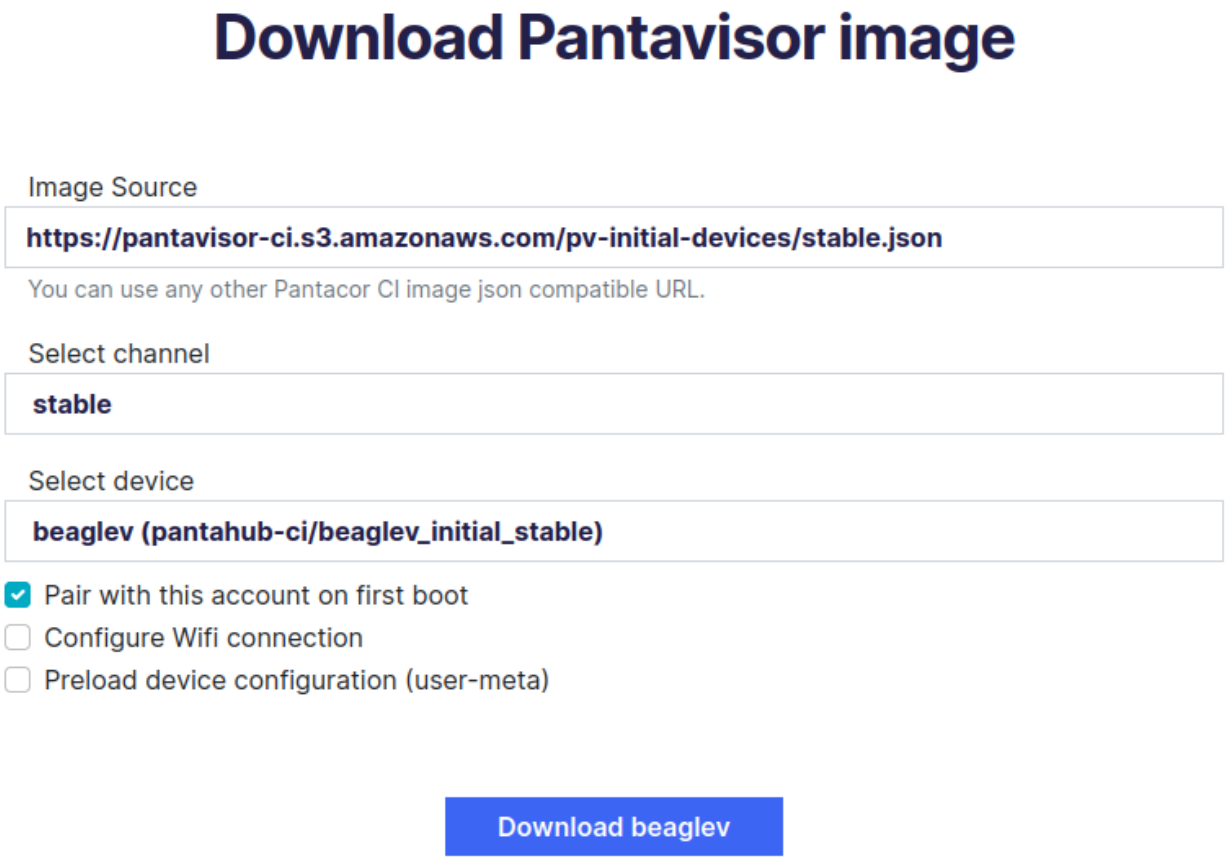
Download Initial Image

Initial images come with the Pantavisor BSP plus a set of Linux-based containers that provide basic network connectivity, discovery services and development tools. As with any other of our initial images, there are two options for downloading them:

- From Pantacor Hub
- Non-Registering Option

FROM PANTACOR HUB

This option requires registration in Pantacor Hub. After that, visit the Pantavisor image download page.



Select your `Raspberry Pi` device, the `stable` channel and `Pair with this account on first boot`. Optionally, select `Configure Wifi connection` to make further configuration steps easier.

NON-REGISTERING OPTION

If you prefer not to register, you can directly use a generic image from our download page.

Now find and download your `Raspberry Pi` version in the stable initial images list.

Flash initial image

For most users, the Raspberry Pi Imager, which is available for Linux, Mac OS and Windows, can flash our Pantavisor Image on to a micro SD card. Alternatively, check out the command line instructions for Linux and Mac OS to flash your device without additional software requirements.

RASPBERRY PI IMAGER

1. Download the Raspberry Pi Imager.
2. Select the initial image downloaded previously.
3. Select your micro SD card and click write.

LINUX COMMANDS

Firstly, find the SD card device name using `df`:

```
df -h
```

Then, you can use the `dd` tool following these steps (remember to substitute `/dev/sdX` for the device node corresponding to your SD card, or else you will overwrite the wrong device!):

```
umount /dev/sdX*
gunzip -c  arm-rpi3.img.gz | sudo dd of=/dev/sdX bs=32M
sync
```

MAC OS COMMANDS

For the Mac OS, the first step is to manually extract the `rpi3_initial_stable.img.xz` file. Then, find the SD card device name by opening a terminal and running the following command:

```
df -h
```

```
Downloads — -bash — 107×24
Sirins-MacBook-Pro-2:Downloads sirink$ df -h
Filesystem      Size   Used  Avail Capacity iused               ifree %iused  Mounted on
/dev/disk2s1   931Gi  114Gi  814Gi    13% 3015373 9223372036851760434    0%   /
devfs          201Ki  201Ki    0Bi   100%     695                   0  100%   /dev
/dev/disk2s4   931Gi  3.0Gi  814Gi     1%       3 9223372036854775804    0%   /private/var/vm
/dev/disk1s2   111Gi  212Mi  111Gi     1%      84          4294967195    0%   /Volumes/A
/dev/disk1s3   111Gi   64Gi   47Gi    58%  661070          4294306209    0%   /Volumes/B
map -hosts       0Bi    0Bi    0Bi   100%       0                   0  100%   /net
map auto_home    0Bi    0Bi    0Bi   100%       0                   0  100%   /home
/dev/disk3s1    15Gi  2.4Mi   15Gi     1%       0                   0  100%   /Volumes/BOOT
Sirins-MacBook-Pro-2:Downloads sirink$
```

You can see on the bottom a 15 GB disk `/dev/disk3` is mounted on `/Volimes/BOOT`.

Unmount it and flash it with the `dd` tool (remember to substitute `/Volumes/BOOT` and `/dev/disk3` for the device node corresponding to your SD card):

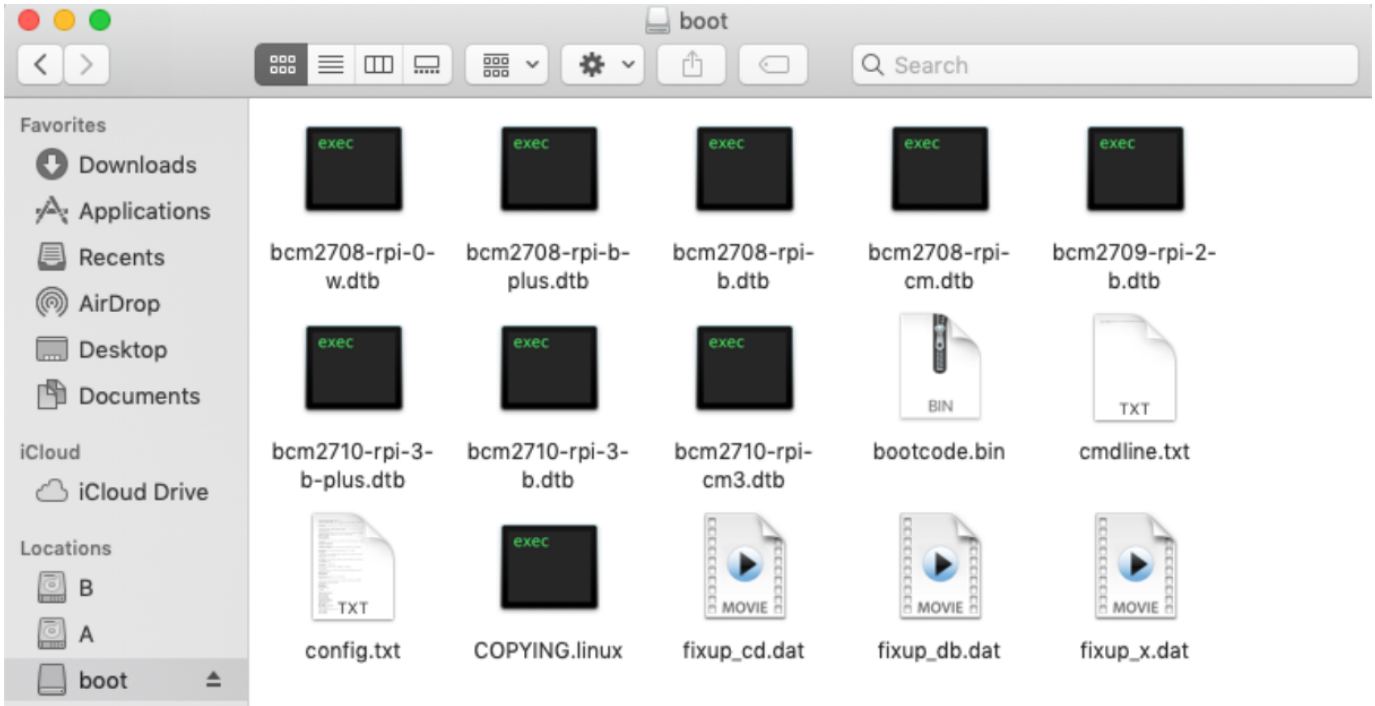
```
diskutil umount /Volumes/BOOT
```

```
Sirins-MacBook-Pro-2:Downloads sirink$ diskutil umount /Volumes/BOOT
Volume BOOT on disk3s1 unmounted
Sirins-MacBook-Pro-2:Downloads sirink$
```

```
sudo dd if=arm-rpi3.img of=/dev/disk3 bs=32m
sync
```

```
Downloads — -bash — 107×24
Sirins-MacBook-Pro-2:Downloads sirink$ sudo dd if=rpi3-pv-1024MiB.img of=/dev/disk3 bs=32m
32+0 records in
32+0 records out
1073741824 bytes transferred in 130.873372 secs (8204433 bytes/sec)
Sirins-MacBook-Pro-2:Downloads sirink$
```

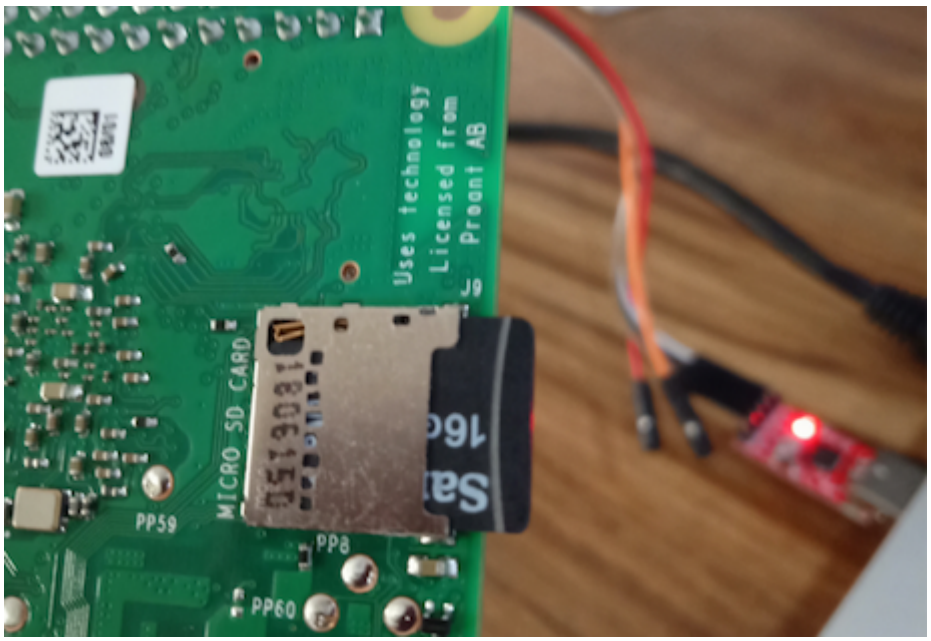
Verify the image contents by opening the SD card partition named `boot`:



3.2.3 First Boot

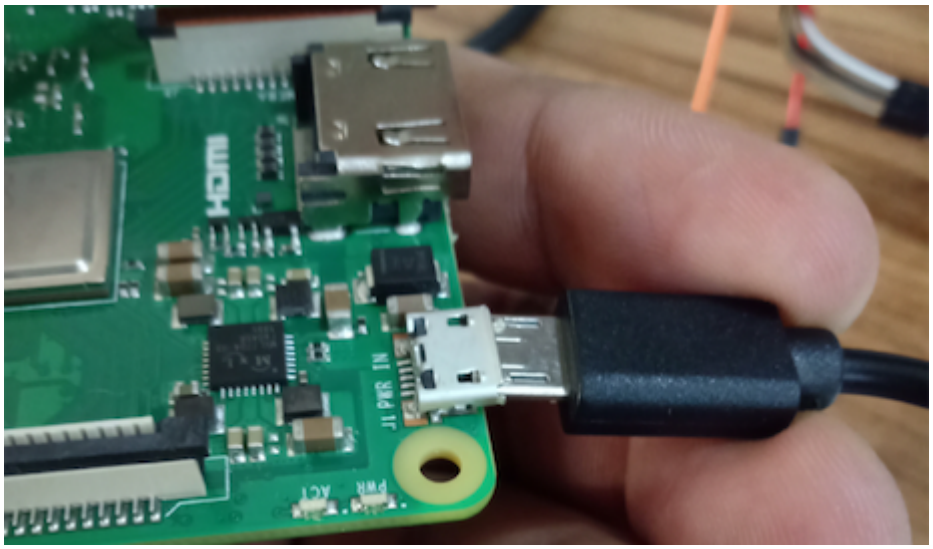
Now that we have set up the SD card, we are going to boot up the board and connect it to your local network.

Insert SD Card into the Board



Supply Power to your Board

Do that by connecting a USB micro cable from your computer to your board:



If everything went well, the PWR led (red) should now turn on, while the ACT led (green) will continue blinking. That means the board is up and running.

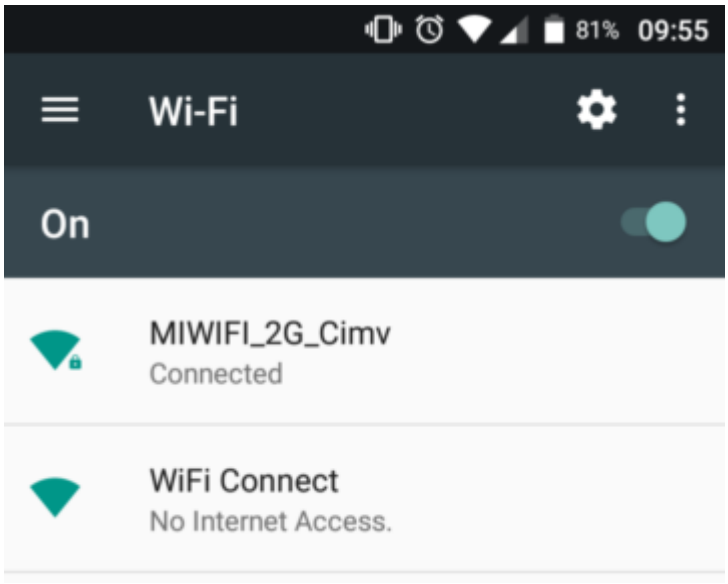
Setting up the Connection of your Board

This is only required if you have not already set up your Wifi credentials, and will be necessary so you can manage your device from your host computer. If you want to do it through Pantacor Hub, you will need to connect it to an Internet-facing network. In any case, you will have two options on your `Raspberry Pi`: WiFi or Ethernet.

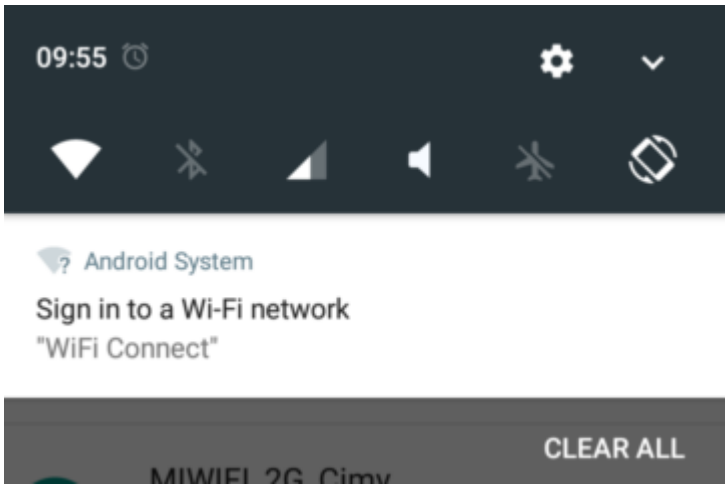
WIFI

If a network cable is not an option, you can use WiFi. To connect your Raspberry Pi to the WiFi network, provide the SSID and password to it. This can be accomplished with a smartphone or any other device with wireless capabilities.

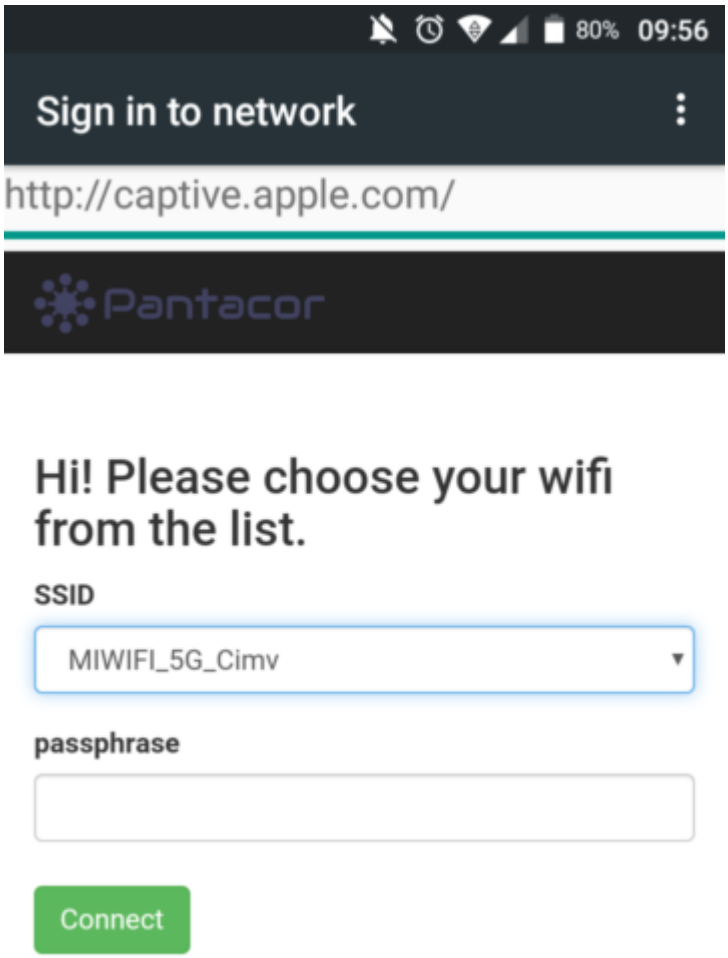
The first step is to wait a couple of minutes after boot up until you find a connection with SSID "Wifi Connect" and no password in your computer or smartphone:



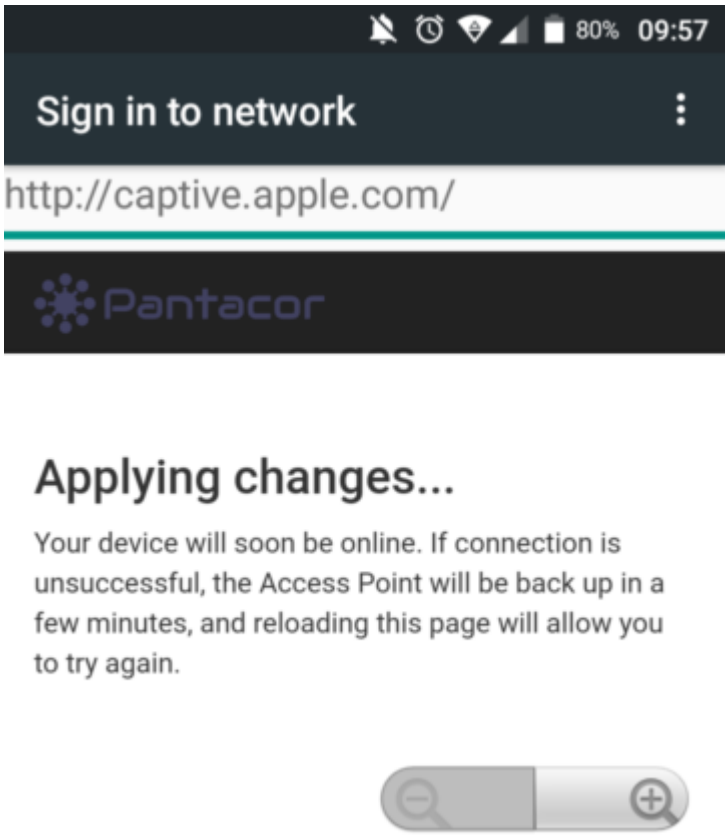
After you connect, a captive portal pops up on your phone:



Enter the portal, select the WiFi network and enter the password.



The Raspberry Pi now has your network credentials saved locally and will automatically connect to that SSID after every reset.



ETHERNET

The fastest way to connect your device to your network is with a cable. This can be done directly to your LAN or to your host computer.

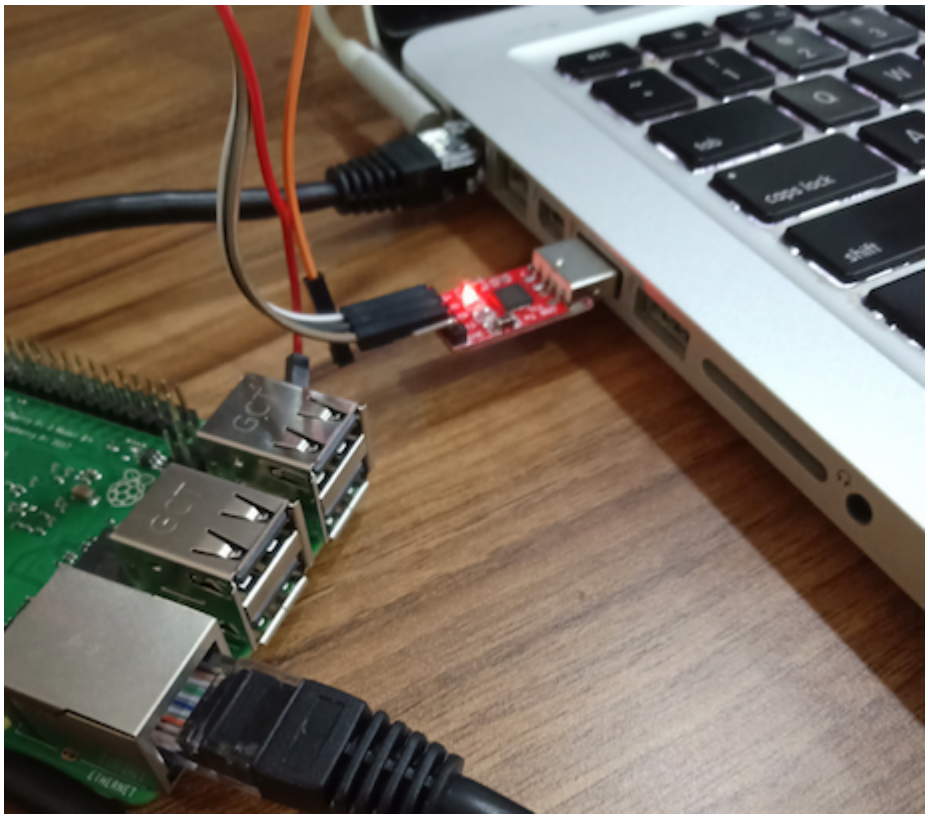
Local Network

Plug one end of the Ethernet cable to your router or switch and the other into your Raspberry Pi.

Host Computer

Alternatively you can use your PC to set up an Ethernet connection for your Raspberry Pi, by connecting an Ethernet cable from your board to your computer.

After the board is up and LAN connection is on, the orange LED on the Ethernet cable should start blinking.



If you want to share your computer's Internet connection to the board, follow your OS specific instructions:

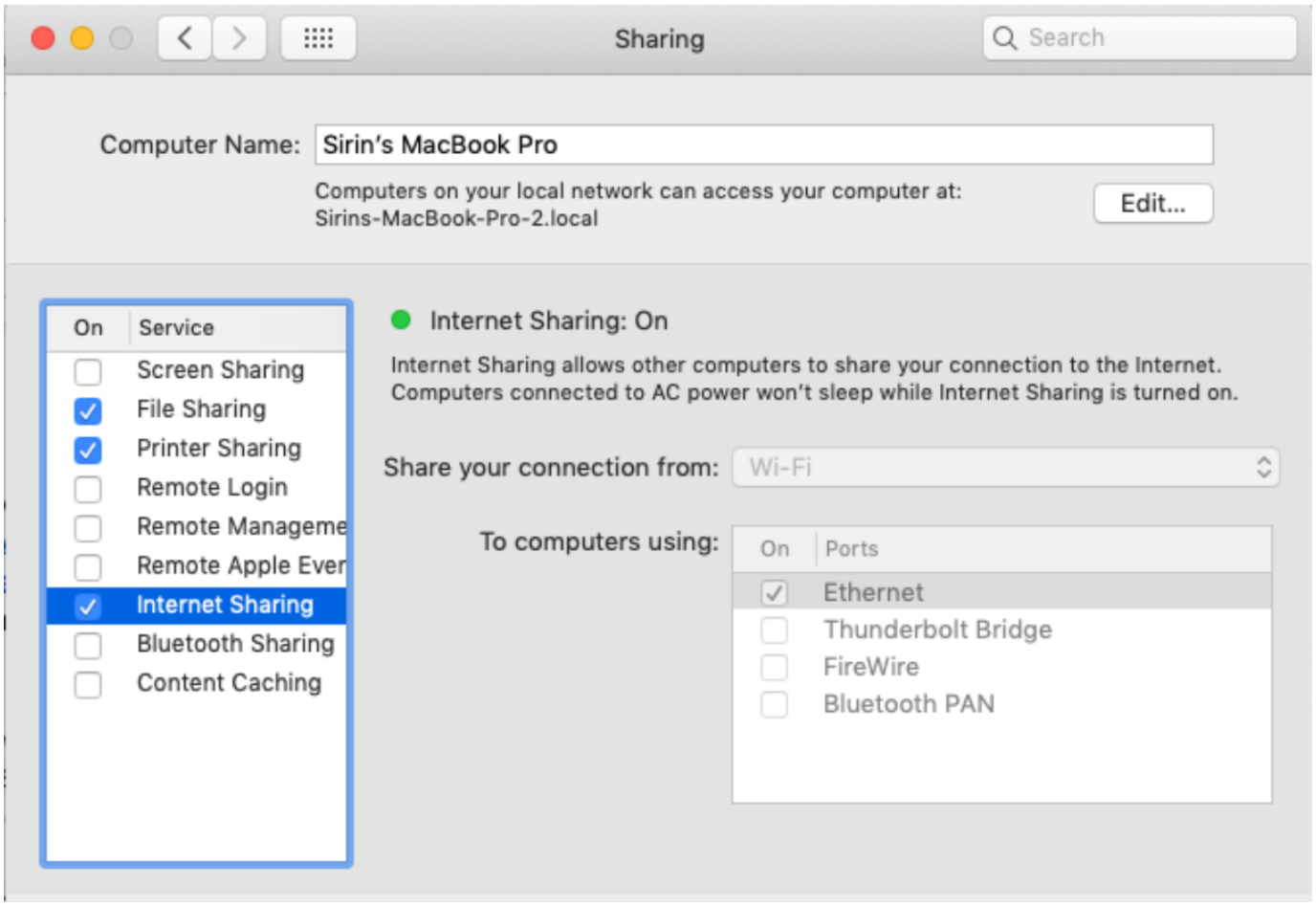
Linux

> **Note**
>
> These instructions are for Ubuntu only.

1. Click on the network icon and go to `Edit Connections...`
2. Double click on your wired connection:
3. Switch to the `IPv4 Settings` tab
4. Select `Shared to other computers` method
5. Click on `Save...`

Mac OS

1. Open System Preferences
2. Select Sharing Preference Panel
3. Enable Internet Sharing with the following settings:
4. Share your connection from: Wi-Fi
5. To computers using: Ethernet



Windows

1. Go to Network Connections
2. Open your WiFi connection's properties:
3. Switch to the `Sharing` tab
4. Enable it
5. Select Ethernet as the `Home networking connection`
6. Click on `Apply`

What is next?

Our initial images come with a set of containers that will help you with the development process. If you want to install new containers and other types of device managemtn, you can proceed to this page.

3.2.4 Troubleshooting

Board does not Boot

If the RPi3 ACT led (green) is not blinking, you may be facing a boot problem.

To check if the device flashed properly. Turn off your RPi3, remove the SD card and insert it in your computer. The SD card should have two partitions, `pvboot` and `pvroot`. If it doesn't, then go back to the image setup instructions and try again.

To eliminate any hardware issues, try flashing your device with one of the Raspberri Pi initial images like this one: NOOBS.

Board Boots Up but does not Connect to LAN

If the RPi3 ACT led (green) is blinking, but the ethernet led (orange) is not blinking, you can see more logs during the boot process via TTY. TTY debugging needs a Pantavisor image built using the debug build option. You can use HDMI (just to check the logs) or TTY to check the logs and display a debug console.

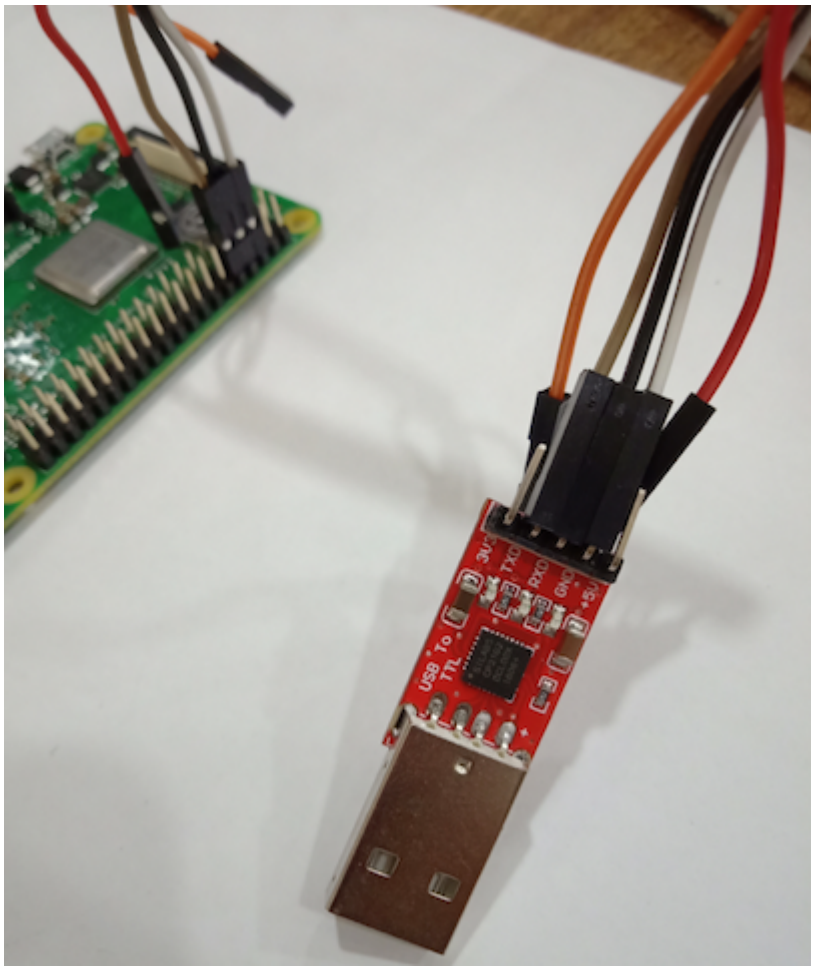
HDMI CABLE

Connect an HDMI cable to the Raspberry Pi before booting will display the kernel log when the board starts.

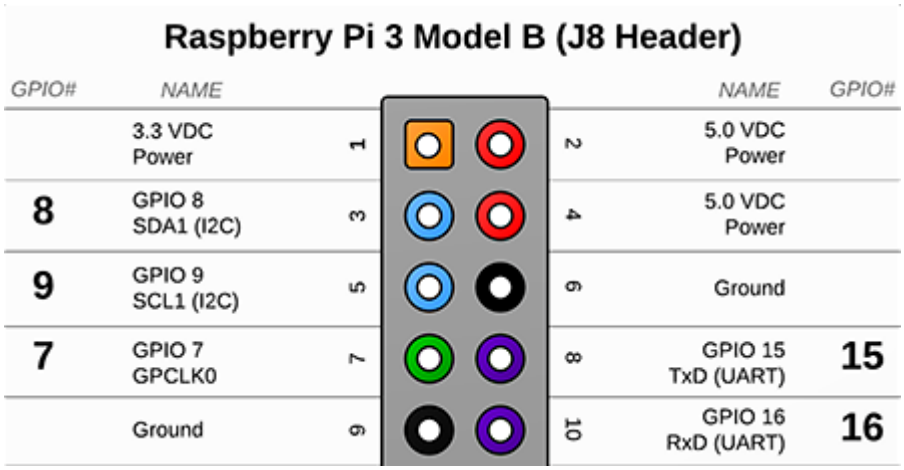
TTY

Connect the Raspberry Pi board to a USB-to-TLL board (or similar) with the following configuration:

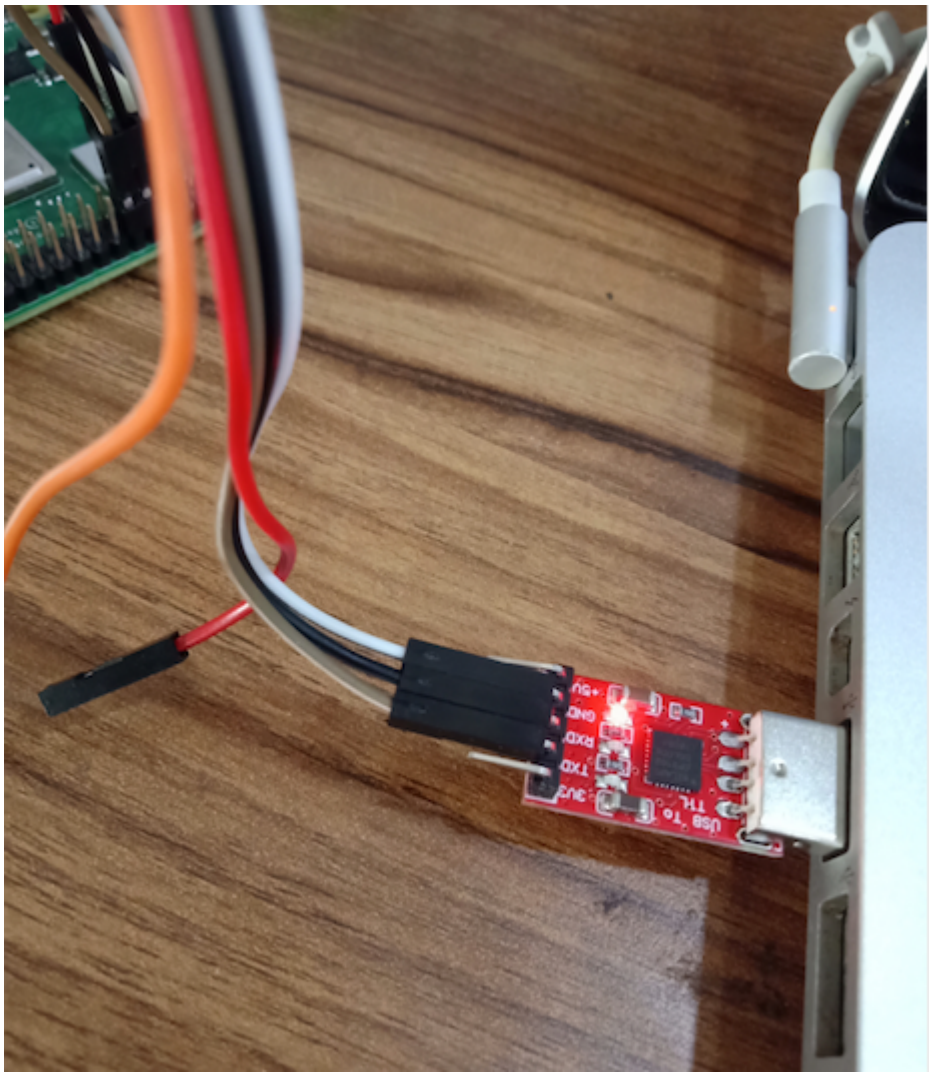
Raspberry Pi Board	USB to TTL
Pin 6	GND
Pin 8	RXD
Pin 10	TXD



To Identify the Pins(6,8 & 10) on the board, please check below:



Then connect the USB-to-TTL board to your computer:



After this, these are the preparations you have to made in each OS before booting up the device.

Linux

List serial lines with the `dmesg` command to identify which `/dev/ttyX` device corresponds with the connected USB-to-TLL board. After identifying the device, then start minicom with said device as a parameter:

```
dmesg | grep tty
sudo minicom /dev/ttyX
```

Mac OS

First install the CP210x USB to UART Bridge VCP drivers:

- Download link: CP210x USB to UART Bridge VCP Driver for Mac OS

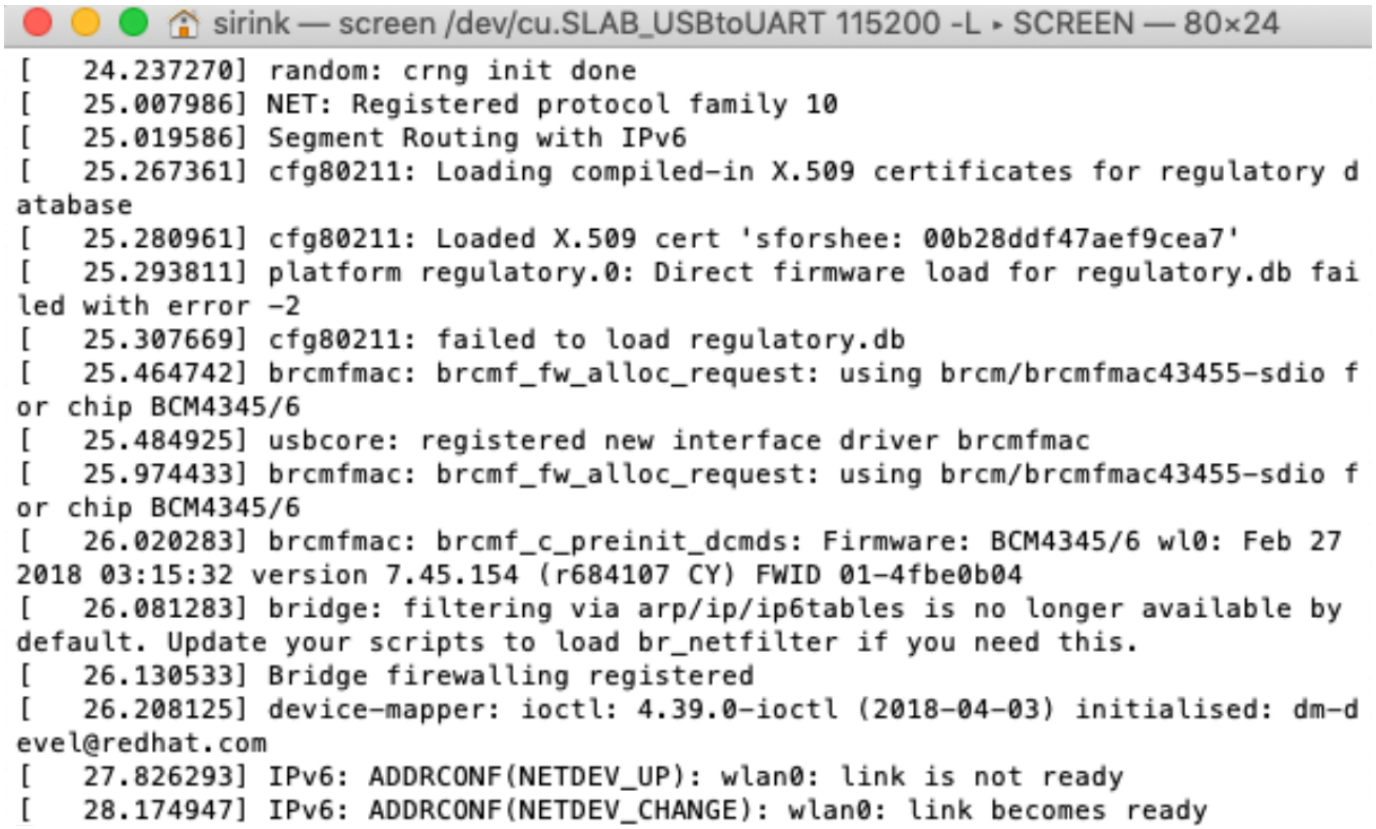
Verify the USB-to-TLL board can be detected by your computer:

```
$ ls /dev/cu*
> cu.SLAB_USBtoUART
```

Next detect data from the serial cable with: `screen`

```
screen /dev/cu.SLAB_USBtoUART 115200 -L
```

You will see a blank terminal waiting for the Raspberry Pi's input. Now it is time to turn it on.



Board Connect to LAN but pvr scan does non Detect Device

If no devices are detected, try running through this check-list:

- both your Raspberry Pi and your computer are connected to the same network
- mDNS packets are allowed in your router and port 5353 is open on your computer firewall
- access your router admin settings and check that DHCP has served an IP to the board and that you can ping it

3.3 Other Devices

3.3.1 Choose your Image

If this is your first time working with Pantavisor, we recommend you to get back to the Raspberry Pi get-started guide, which follows a more step-by-step approach for beginners. If you cannot get your hand into either a `Raspberri Pi 3 B+` or a `Raspberri Pi 4`, or if you have already completed the get-started guide and you want to move forward to other boards, this tutorial is for you.

First step is to choose and download the image that you are going to use. If you prefer to build your own images, just skip this page and move on to boot it up.

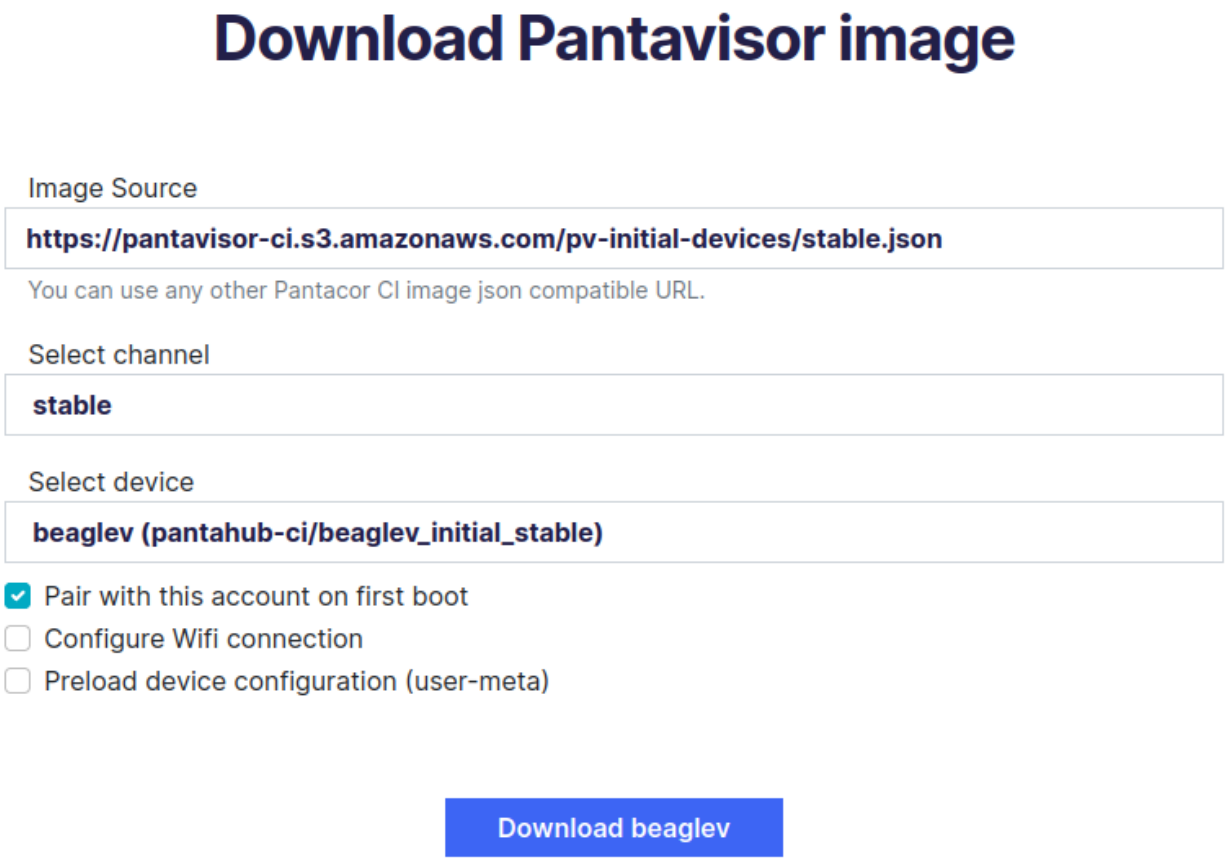
Download Initial Image

Initial images come with the Pantavisor BSP plus a set of Linux-based containers that provide basic network connectivity, discovery services and development tools. There are two options for downloading them:

- From Pantacor Hub
- Non-Registering Option

FROM PANTACOR HUB

This option requires registration in Pantacor Hub. After that, visit the Pantavisor image download page.



From there, you can select device and channel. Additionally, the download page offers options for personalizing your image (only supported on the `Raspberry Pi` targets):

- `Pair with this account on first boot` : Select this to avoid to manually claim your device
- `Configure Wifi connection` : Select this to insert WiFi credentials so the device can connect to your network
- `Preload device configuration (user.meta)` : Select this to add default user meta to your device

NON-REGISTERING OPTION

If you prefer not to register, you can directly use a generic image from our download page.

Just choose a version, a target architecture, and click on the download link.

3.3.2 First Boot

If you have already downloaded a Pantavisor image, it is time to use it for the fist time in your device. For that we are generally going to flash the storage medium of the device and set up its connectivity so you can manage it from your host computer. If you want to perform this management using Pantacor Hub, you will need to connect it to an Internet-facing network.

Target Device Architectures

> **Note**
>
> These guides do not cover all the boards we support. The more complete list of boards currently supported can be consulted here.

RPI3, RPI4, BPI-R64, RPI0W, ROCK64 OR NVIDIA TEGRA

To use these images, you need to flash your SD card and then boot the board up. You can see how to do this with more detail in the get started guide, which focuses RPi3 and RPi4, but the process will be similar for all of these devices.

To connect the board to a LAN or directly to your computer, you can follow the RPi3 and RPi4 instructions again. Notice in the case of Rock64, only Ethernet will be a possibility.

We recommend downloading this image through the wizard on https://hub.pantacor.com/download-image which allows you to markt he image to auto join your account and even configure wifi credentials for systems that have wifi built-in or in a supported dongle.

COLIBRI-IMX6DL OR COLIBRI-IMX6ULL (TORADEX)

To install our Toradex build, we offer a tarball that you can unpack on an SD Card that you then plug into your Toradex board before booting the Tezi Easy Installer. It will automatically install the image on an SD Card (tested with Tezi Easy Installer 5.7.1 for iMX6 and iMX6ULL).

- Colibri-iMX6dl
- Colibri-iMX6ull

Our builds support both gadget networking as well as ethernet if your carrier board has them wired.

Before you install you can also overload the "configargs" variable in uEnv.colibri-imx6dl.txt and uEnv.colibri-imx6ull.txt to inject pvnet_ arguments to configure the default network, for example:

```
$ cat uEnv.colibri-imx6ull.txt
localargs=pvnet_type=gadget pvnet_address=dhcp
```

Some examples are included in the default uEnv files:

```
#localargs=pvnet_type=ethernet pvrsdk_httpd_listen=0.0.0.0
#localargs=pvnet_type=gadget pvrsdk_httpd_listen=0.0.0.0
#localargs=pvnet_type=wifi pvnet_ssid=YOURSSID pvnet_pass=YOURPSK
#localargs=pvrsdk_httpd_listen=0.0.0.0
localargs=pvnet_type=gadget pvrsdk_httpd_listen=0.0.0.0
```

> **Note**
>
> The defaults will open the pvtx API, which will enable both the pvtx web UI and local pvr. Please, remember to remove the pvrsdk_httpd_listen flag for anything running in the wild.

X64-UEFI OR X64-UBUNTU

These images boot from an external storage device (SD card, USB key..) from BIOS/UEFI. The default image is meant to be directly flashed onto the final storage device where the system will run, while the installer is flashed into a temporary storage device. To see how to flash your storage device, take a look at the get started guide. Even though this guide is for RPi3 and RPi4, the flashing process will be similar.

Once this is done, insert the storage device into your x64 device and power it up. You might also need to access the BIOS/UEFI menu and boot the target device from the storage device, if that is not the default boot option. In that case, you will need to do this with a keyboard and screen connected to your device.

To connect the board to a LAN or directly to your computer, you can follow the RPi3 and RPi4 instructions again.

Default

If you are using the default image, Pantavisor boots from the storage device. Just make sure your BIOS/UEFI is set to run that device by default and you should be good to go.

Installer

After booting up, the installer prompts with a menu and automatically flashes the first available permanent storage device found. A reboot is required after this finishes.

For troubleshooting or to simply choose the permanent storage device from a list of flashable devices, select the manual installation option in the menu before it triggers the automatic installation. Instructions will follow on screen that describe how to do this from the console.

QEMU

Despite Pantavisor is made to be run on embedded devices, you can also try it out by emulating the device with QEMU. Another option that we cover in a different guide is to run Pantavisor as a deamon in any Linux system.

x64

Before you can use the image, change its format to QCOW2 using the `qemu-img` tool. Remember to use the default (non-installer) x64 image:

```
qemu-img convert -O qcow2 x64_initial_stable.img x64-uefi-pv-4096MiB.qcow2
```

Besides that, it is necessary to run an OVMF. Once done, execute this command:

```
qemu-system-x86_64 -enable-kvm \
    -bios /usr/share/qemu/OVMF.fd \
    -drive file=x64-uefi-pv-4096MiB.qcow2,format=qcow2 \
    -nographic \
    -m 512m \
    -net nic \
    -net user
```

This should run the image and show a menu to finally execute Pantavisor.

Malta

We prepared a Docker container that allows you to run the malta-qemu image. Keep in mind though that the script that runs changes the network configuration on your host:

```
docker run --privileged --net host -v </your/host/path/to/pflash.img>:/tmp/pflash.img -it --rm pantacor/qemu-malta-16m
```

When the initialization script and boot up log ends, press ENTER to see the LEDE console.

What is next?

Our initial images come with a set of containers that will help you with the development process. If you want to install new containers and other types of device managemtn, you can proceed to this page.

Troubleshooting

Our pre-compiled images are built with the PANTAVISOR_DEBUG option. This means you will have SSH, TTY and other goodies available for debugging your devices. Most of the items explained in our getting started troubleshooting section are applicable here.

3.4 App Engine

3.4.1 Requirements

Pantavisor App Engine init mode lets you run Pantavisor in your already set up Linux distro. This is a less intrusive way to try out Pantavisor without having to flash your device storage. Pantavisor App Engine will not get you to the minimal system we strive for. However, it will be a perfect tool for quick and non-disruptive prototyping of our container engine on your devices. It can be also used for container and Pantavisor development and testing.

To start using it, you have several options:

- Downloading the self-contained x64-appengine delivery package from our CI (only for Ubuntu 22.04)
- Building the x64-appengine target from source code (only for Ubuntu 22.04)
- Installing App Engine in your device using the installer. This option will setup Pantavisor without containers, which you will be able to add after manually claiming your device

3.4.2 Installation

> **Warning**
>
> The rest of this how to guide will only apply to downloading or building the x64-appengine package in Ubuntu 22.04. Other distros might work but could require further work.

Once you have built or downloaded the x64-appengine target, you will have, at least, these files in your Ubuntu 22.04 host:

```
docker  scripts  test.docker.sh  tests
```

That is, some packaged Docker containers, testing data and some scripts to run it all.

Firstly, you will need to install App Engine and its dependencies with this command:

```
./test.docker.sh install-deps
```

3.4.3 First Run

Now that you have installed Pantavisor App Engine, let us see how to run it.

To list all available tests, you can execute:

```
./test.docker.sh ls
```

If you want to run the most basic test (pvcontrol ls and check mounts in both mgmt and non-mgmt containers):

```
./test.docker.sh run local:0
```

This will execute the test in a one-shot command. To run the test interactively:

```
./test.docker.sh run bshpvctld:0 -i
```

This will open a session inside the App Engine Docker container. Then, make sure to exit so Pantavisor can release its resources:

```
exit
```

4. How to Use Pantavisor

4.1 Choose your Way

We try to keep management of your devices open and not force a single path. With management, we mean updating the device, getting its logs, sending and receiving other useful metadata, etc. This can be done mainly in two ways:

- Remote experience: manage your devices through Pantacor Hub or any other cloud of your choice.
- Local experience: directly manage your devices from your host computer.

If this is your first time, we recommend you to go with the remote experience and Pantacor Hub, as it is the simplest and most visual way to get started. However, if you prefer the local experience, Pantabox offers both a ncurses menu UI and a CLI tool, while pvtx allows basic management from a web UI stored on the device itself.

For both remote and local experiences, we have developed pvr, a CLI tool for management of devices from the host computer.

> **Note**
>
> Remote and local modes can be switched during execution time in a Pantavisor-enabled device, so you are not burning bridges just by starting with one or the other!

Once you have a grasp on our management tools, you will be able to move on to more advanced stuff to get the most out of Pantavisor.

4.2 Basic Management

4.2.1 From Pantacor Hub UI

Claim your Device

The first step to remotely manage your device is to claim it. Claiming is the process of linking a newly flashed device and a Pantacor Hub account.

> **Note**
>
> For this how-to guide, it is assumed that you are registered in Pantacor Hub.

To claim a device, it must meet some pre-requisites:

- The board has to be connected to an internet-facing network.
- The device was not previously claimed.
- One of this must be true:
  - It must be running revision 0, which should happen in any newly flashed device
  - If running a local revision, the go remote command must be executed.

There are three ways to claim your device, depending on the capabilities of the target image you just installed: automatically, manually from the host and manually from the target.

AUTOMATICALLY

This process can be done automatically if the pairing option is set in the download Pantavisor image page.

MANUALLY FROM THE DEVICE

Manually claiming a device from the device itself will depend on where you are on the device, that is, if you have access to a pvr-sdk container running on top of your Pantavisor-enable device or if you have access to the Pantavisor file system.

Using Pantabox

This is the easiest way if you have downloaded a non-personalized image that has the pvr-sdk container.

This container has the Pantabox tool installed by default. To claim the device, just inspect your device from your host computer and execute the pantabox-claim command. The script will take care of the pairing process.

Getting Credentials from Pantavisor File System

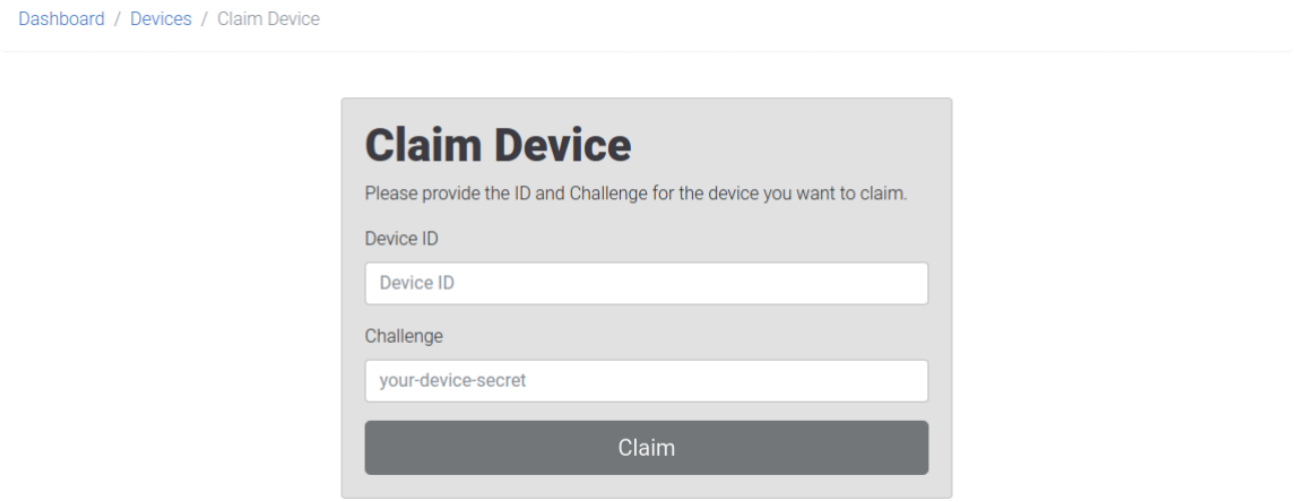
This is the way if you are running Pantavisor using QEMU or with App Engine images.

The credentials to claim your device are located in these files inside of the Pantavisor file system:

```
# cat /pv/challenge
pleasantly-finer-unicorn
# cat /pv/device-id
5b582638c67920b9de2
```

> **Note**
>
> This information will also be printed in pantavisor log.

You will need to insert those credentials in Pantacor Hub. Just click on the `Devices` tab on the left side of the UI and press the blue `Claim a Device` button that will appear on the right side.



MANUALLY FROM THE HOST

As using a CLI tool can be more convenient than doing it from the device itself, we offer the possibility to claim your device using the pvr tool, if the image that you just flashed has the pv-avahi container.

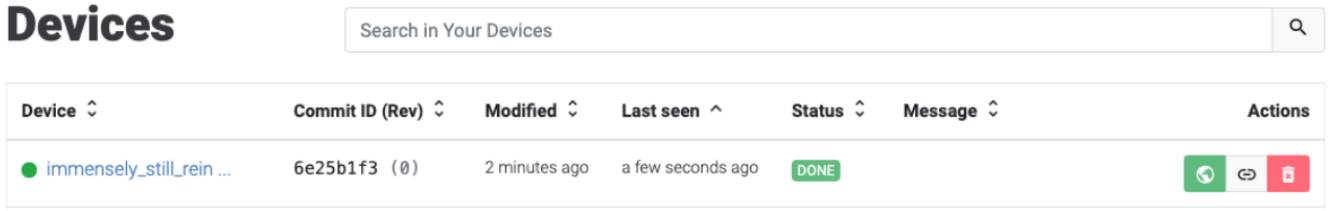
Assuming your computer is in the same local network as your board, just execute this command to discover the device:

```
pvr device scan
```

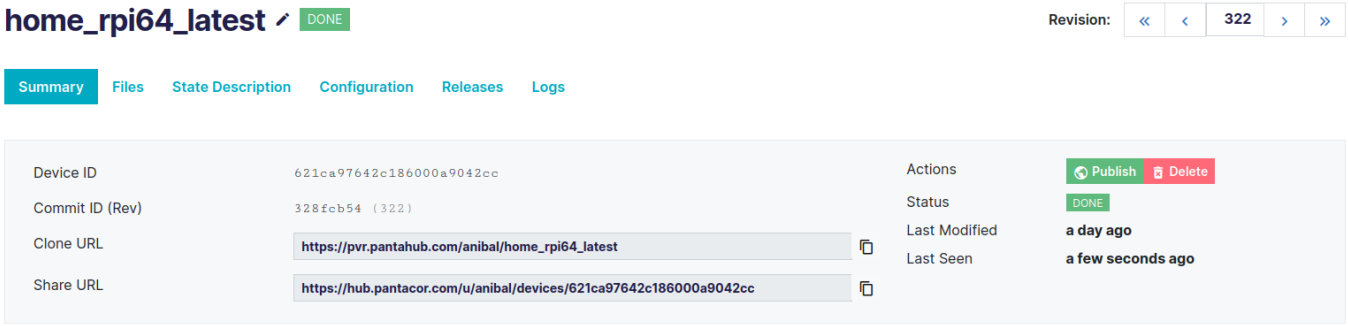
If `pvr` finds a device on the local network, it will present this metadata for you to claim it via the command quoted in the output. Just run that command as-is and your user should now own your device.

Device Dashboard

Assuming the claiming process has been successful, the new device should appear in the device list of your Pantacor Hub account:



If you click on the name of the new device, you will access to the device dashboard:



Bsp information

Target	Platform	Source	Version	PVR version
rpi64-5.4.y	rpi64-5.4.y	https://pvr.pantahub.com/pantahub-ci/rpi64_5_4_y_bsp_latest#bsp	015-rc5-73-g31c0857	034-34-g52038ee2

Components

Name	Image	Version	Run Level	pvr
awconnect	wifi-connect:arm32v5	81809a6	platform	
pv-avahi	pv-avahi:arm32v6	895b2af	platform	
pvr-sdk	pvr-sdk:arm32v6	f14625a	platform	
webserver	nginx:latest	d3d73c0	app	

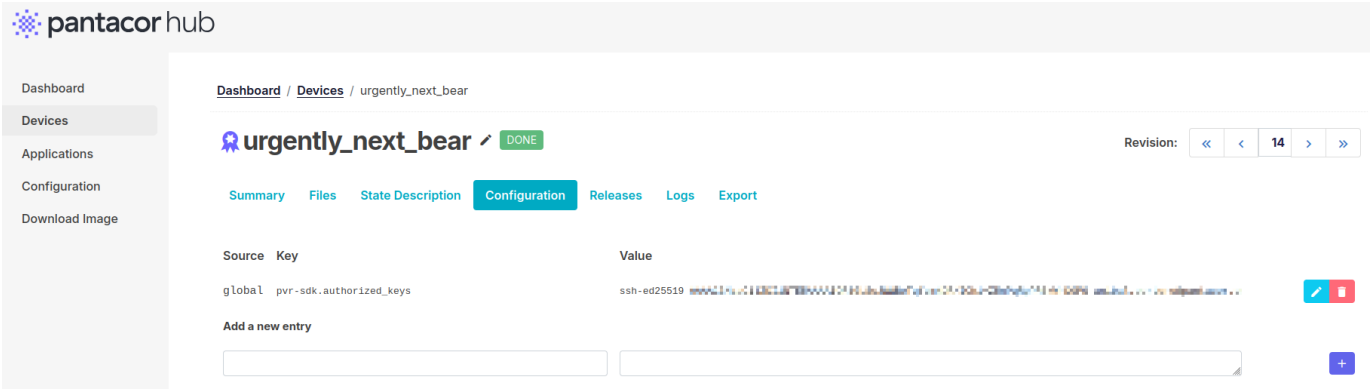
From the dashboard you can perform some simple management operations:

- Get the `Clone URL` that pvr can use to send new revisions to the device
- Get the `Share URL` so you or others can get a direct link to the device dashboard
- Check the device status and containers
- Check metadata sent by the device (storage and memory use, system information, Pantavisor state, etc.)
- Check the JSON state and revision artifacts
- Set user metadata to the device
- Redeploy old revisions
- Get the logs pushed by the device

Set User Metadata

User metadata can be used to send information from the cloud to the device. Check the metadata reference to see which values are parsed by default in Pantavisor.

To set metadata key/value pairs, just go to the tab `Configuration` in your device dashboard.

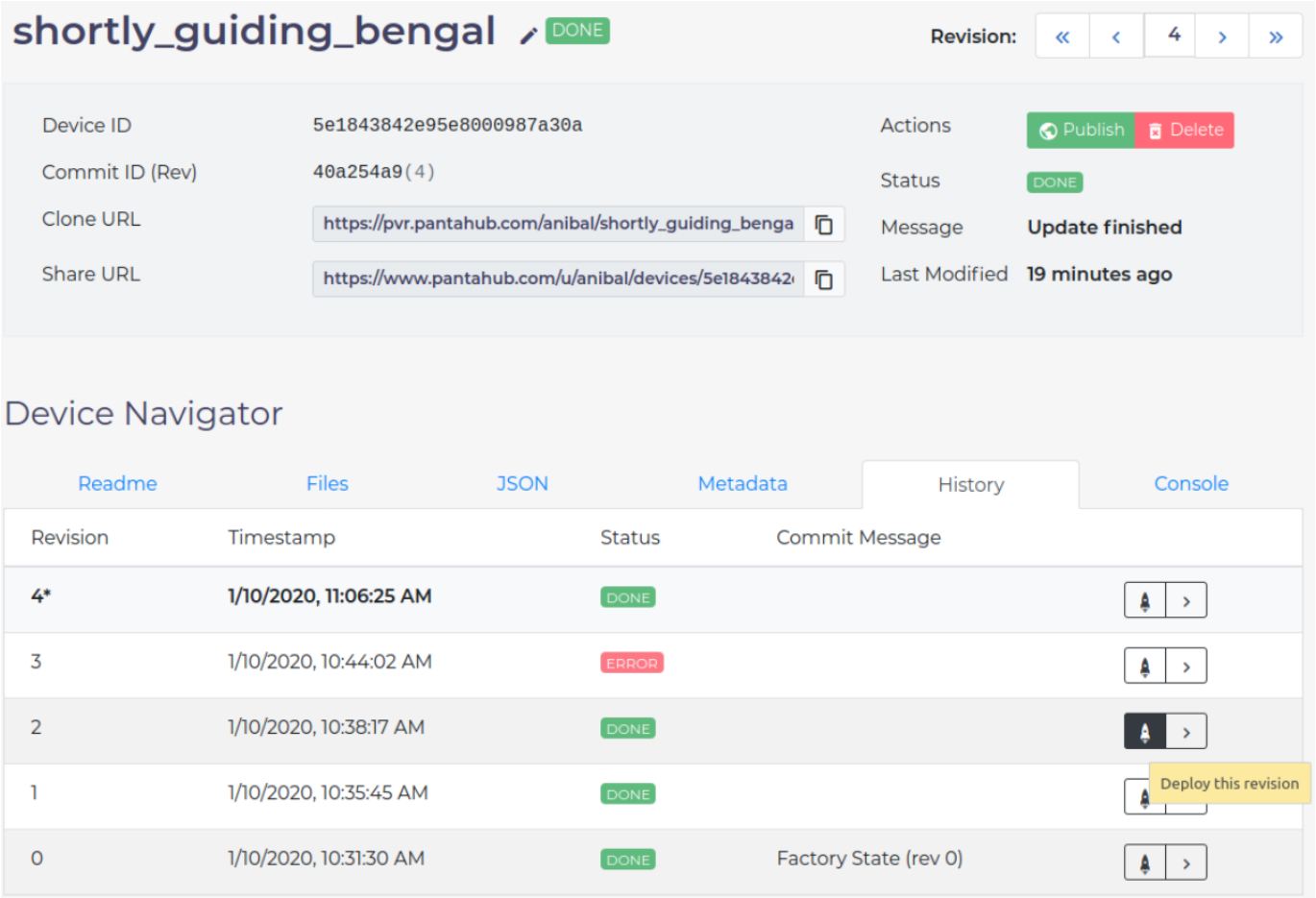


You can also set user metadata at a global level for all your present and future devices in the `Global Configuration` tab on the left side of the UI.

Redeploy Old Versions

Pantavisor will automatically rollback to a previous revision if a newly pushed version does not permit a stable connection with Pantacor Hub.

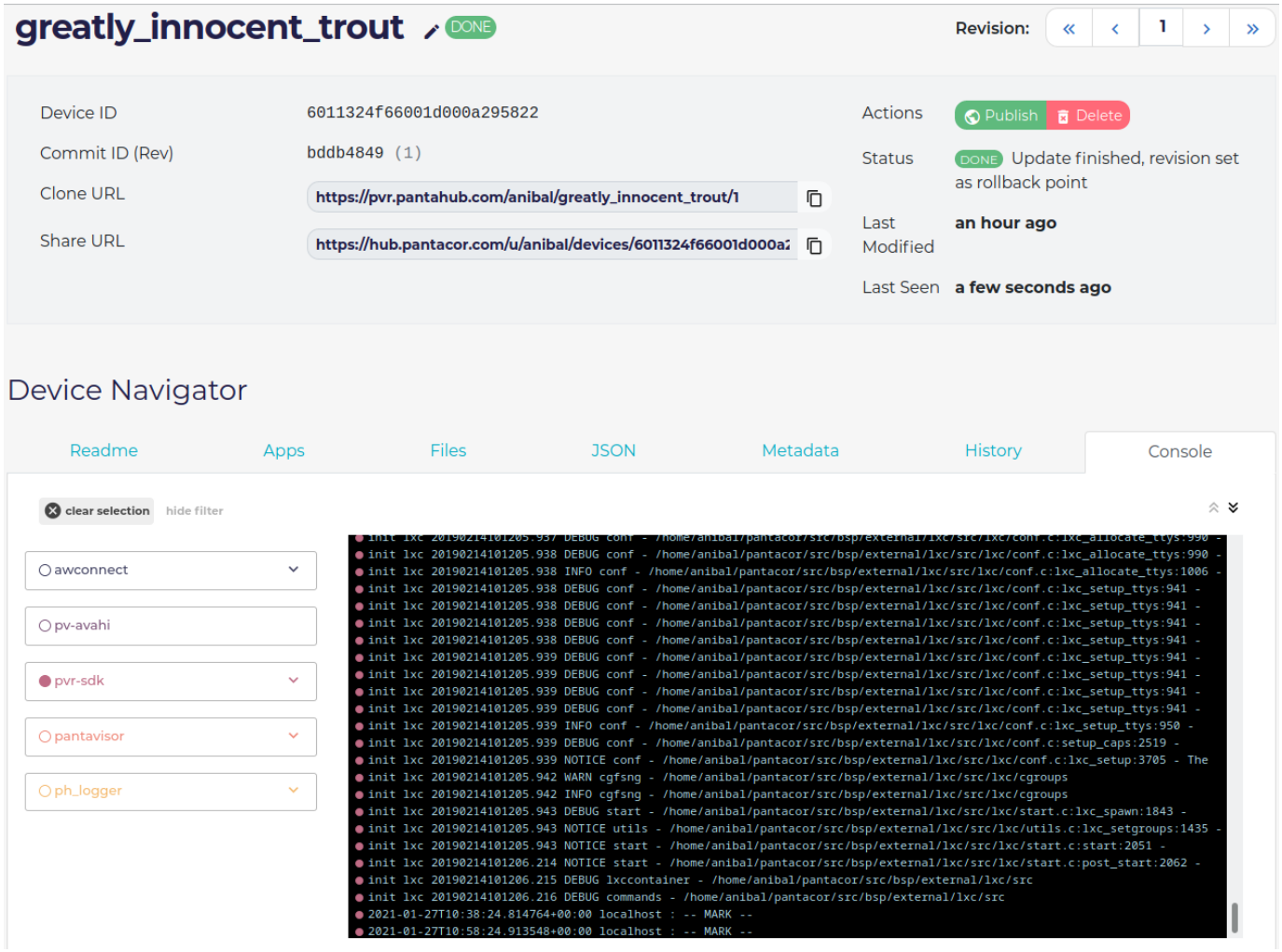
Besides that, you can redeploy an old revision using the Pantacor Hub UI. Just go to your device dashboard and click in the `Releases` tab, where you will find a list of revisions with their time-stamp, status and commit message. At the right column, you have two buttons: one for redeploying that revision, the other one for navigating to that revision in the UI.



The current revision number will be automatically updated by the device along other metadata and can be consulted from the dashboard.

Get the Logs

The device stored logs are pushed to Pantacor Hub by default and can be monitored from the device dashboard in the `Logs` tab:



4.2.2 From the Device Console

Inspect your Device

To locally handle your device, it is necessary to remotely access it via SSH, TTY or by any other means from your host computer, as Pantavisor offers management capabilities from the device itself.

SSH

> **Note**
>
> Knowing your device IP is necesary to SSH it. You can get the IP from the Pantacor Hub device dashboard, by using pvr, TTY inspection or with keyboard and monitor. Other options non related to our tooling are navigating to your router admin page or using arp-scan.

If your device has the pvr-sdk container installed and it is not yet claimed, you can SSH the device using password authentication and then set a public key for further use.

If your device is already claimed, you can directly set your SSH public key.

Password

The pvr-sdk container sets up a password-accessible SSH server by default. To log in, just type this command:

> **Warning**
>
> Make sure you use your own device IP.

```
ssh root@10.0.0.1
```

When prompted, log in with the password `pantabox`.

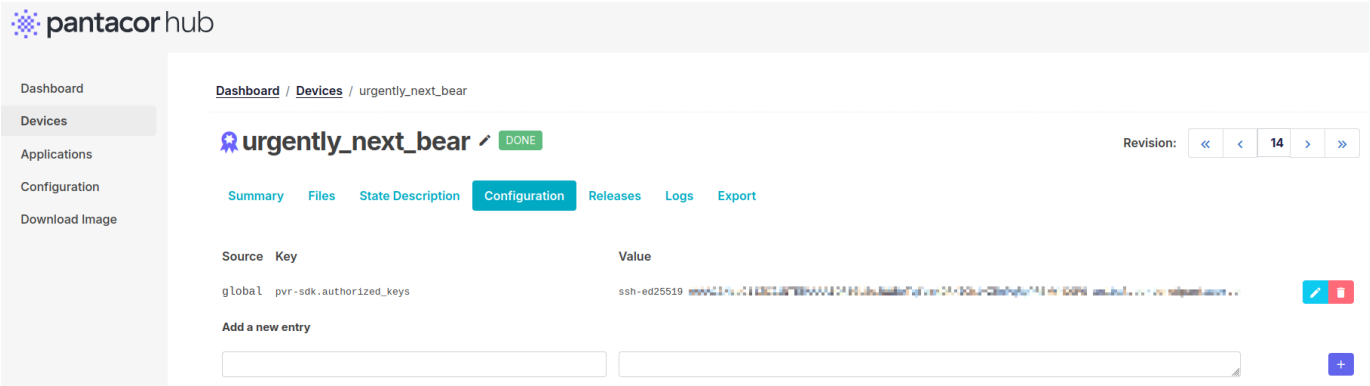
Public Key

We will have to create an SSH public key and set it up in your user metadata. From your Linux host, you can create an SSH pair and print your public key with these commands:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
cat ~/.ssh/id_ed25519.pub
```

Now, the process varies depending whether your device is claimed or not. In case it is, create a new user metadata pair with the pvr-sdk.authorized_keys key:

```
key: pvr-sdk.authorized_keys
value: <ssh pub id>
```



If your device is not claimed in Pantacor Hub, you can access your device with the password authentication method and then set the user metadata pair with the SSH public key using the Pantabox `pantabox-edit-sshkeys` command.

> **Warning**
>
> It is important to remember that the device user metadata can only be edited using `pantabox-edit-sshkeys` if the device is in not under remote control. In that case all the user metadata stored in the device will be automatically overloaded from the one that is stored in Pantacor Hub.

Once your SSH public key is stored in your device, the Dropbear SSH server should be accessible at port 8222 and with the desired container name as user:

> **Warning**
>
> Make sure you use your own device IP.

```
ssh -p 8222 alpine-hotspot@10.0.0.1
```

Furthermore, you can directly reach Pantavisor console using the "/" user:

```
ssh -p 8222 /@10.0.0.1
```

TTY

> **Note**
>
> Not all our initial devices are baked to open a TTY channel out of the box. One example when this TTY is set is the RPi4 board.

An alternative to the network accessed option is getting into Pantavisor through a TTY shell. This will grant an earlier access than SSH and will not require any further configuration on the device side besides connecting the hardware, plus giving access to the bootloader console.

To open a TTY console in your host, you can use `minicom`:

```
sudo minicom -D /dev/ttyUSB0
```

After booting up, you will have two chances to stop the boot up and get a shell that will allow you to interact with the boot up process. First one will be before the bootloader loads the kernel and Pantavisor. If you press any key during when prompted, you will get to the bootloader shell:

```
U-Boot 2016.09-g15600e2 (Jan 15 2021 - 16:18:19 +0000)

Board: MIPS Malta CoreLV
DRAM:  128 MiB
Flash: 16 MiB
In:    serial@3f8
Out:   serial@3f8
Err:   serial@3f8
Net:   pcnet#0
Warning: pcnet#0 MAC addresses don't match:
Address in SROM is         02:b8:2a:7f:48:b6
Address in environment is  52:54:00:12:34:56
, pcnet#1
Hit any key to stop autoboot:  0
malta #
```

From here, you will be able to set Linux environment variables that will let you configure some Pantavisor parameters.

After kernel and Pantavisor is loaded, you can also access Pantavisor user space from an ash shell. For that, just press `d` when prompted:

```
[    1.685794] Run /init as init process
Press [d] for debug ash shell... 5 d

#
```

Pantavisor will prevent rebooting or powering off the device while the debug shell is open. This will allow to debug faulty revisions that might need a rollback. To enable reboot and power off again, use `exit` command or press CTRL-d to close the shell.

> **Note**
>
> The debug shell countdown can be disabled so it is always entered without waiting for user confirmation. See the `PV_DEBUG_SHELL_AUTOLOGIN` key at Pantavisor configuration reference. Just remember to exit the console if you are waiting for a board reboot. It can also be disabled completely to speed up the bootup process. To do so, use the `debug.shell` key.

KEYBOARD AND MONITOR

> **Note**
>
> Not all our initial devices have HDMI and keyboard control enabled. One example when this is set is the RPi4 board.

If you have a keyboard and a monitor at hand, you will get a similar workflow as with the TTY, with access both to the bootloader console and the Pantavisor debug shell.

Navigating the Console

This page shows the most important commands to start navigating through each of the Pantavisor consoles.

BOOTLOADER CONSOLE

The bootloader console can be accessed by interrupting the boot up from either TTY or using keyboard and monitor.

U-Boot

Among other things, from the U-Boot console, you can configure some Pantavisor parameters with Linux environment variables.

To do so, you just have to assign a string with one to many *key=value* pairs, separated by space to the `configargs` variable. In this example, we are setting the log level to WARN and disabling the push of logs to Pantacor Hub:

```
setenv configargs $configargs PV_LOG_LEVEL=2 PV_LOG_PUSH=0
saveenv
reset
```

You can check these and more configurable keys in our reference.

Grub

Setting Linux environment variables is not supported on the Grub platforms. Contact us if you need support for your board!

PANTAVISOR CONSOLE

The Pantavisor console can be accessed using the '/' root container via SSH. Also, getting into the debug shell from either TTY or keyboard and monitor.

Pantavisor Root

To list all running container names:

```
lxc-ls
```

Here, we have access to the different containers using the `pventer` command. To open a session into the `alpine-nginx` container:

```
pventer -c alpine-nginx
```

pvr-sdk

From the pvr-sdk container console, you have at your disposal some handy development utilities such as Pantabox, pvtx and pvcontrol.

The rest of this how-to guide focus on the higher level Pantabox tool.

Using Pantabox

Pantabox allows to interact with Pantavisor from inside of the device.

Pantabox is included by default in the pvr-sdk container, so to begin using it, you just have to access the running instance of pvr-sdk in your device. For example, to do it via SSH:

```
ssh -p 8222 pvr-sdk@10.0.0.1
```

Once you are in, you will find instructions on how to use Pantabox:

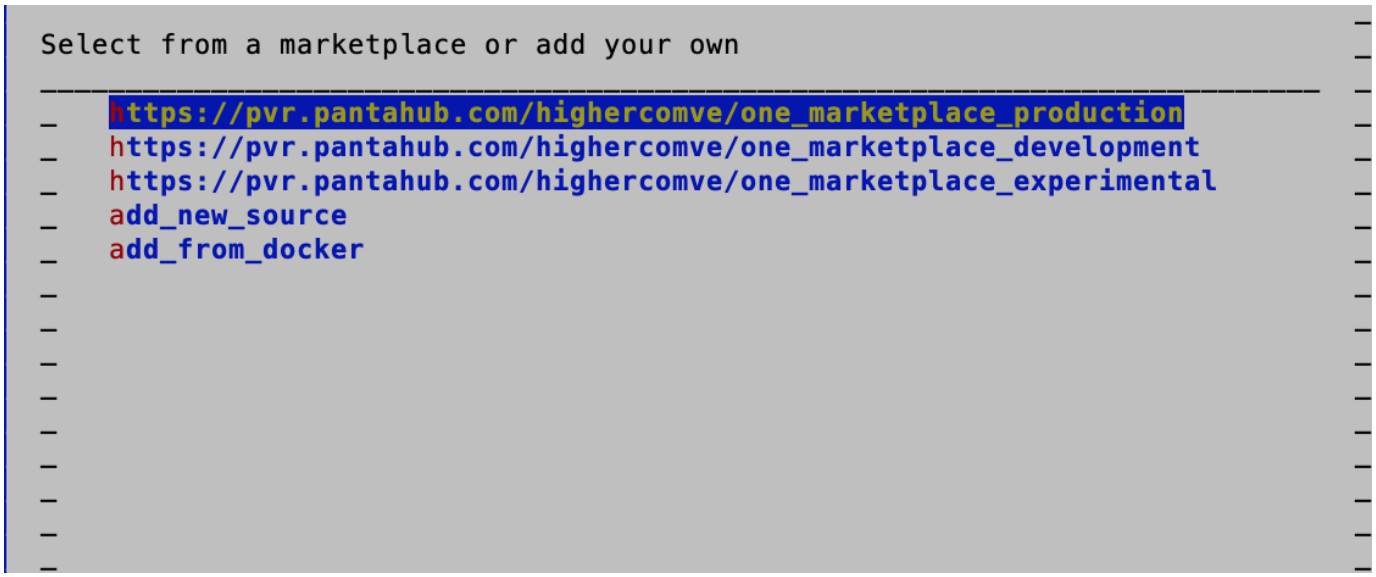
```
Welcome to Pantabox and Pantavisor Linux!

Pantabox is a self-contained front-end to Pantavisor Linux.  With Pantabox you have the
ability to edit, configure, update and manage revisions on your Pantavisor Linux enabled
devices both locally and remotely.

If this is your first login, please remember to change your user password by using
pantabox-chpasswd.

Get started with this command

  pantabox

If you want to learn more about pantabox and the available commands run:

  man pantabox

Try our getting started guides: https://www.pantavisor.io/guides/getting_started/.

Join our Slack community at https://pantavisor.slack.com/.

pvr-sdk:/root#
```

Add Apps from Marketplace

> **Note**
>
> An app is a container that can be run in a Pantavisor device that takes part on the application level functionality.

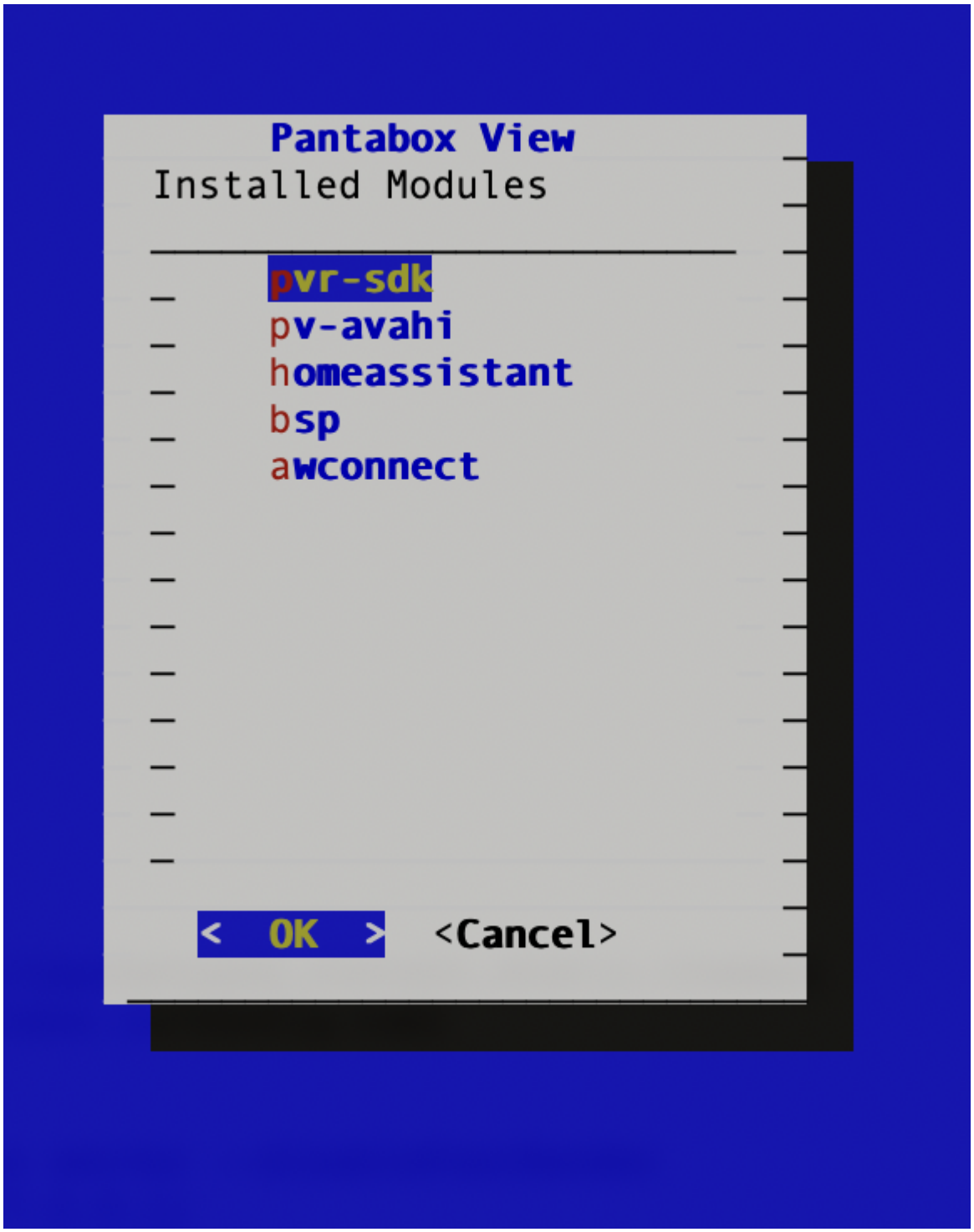
Pantabox offers the possibility to add apps from a Marketplace without going through the cloud. First, execute the `pantabox` command to get its menu:

Select the `install` option on the menu and you will get this:



This lets you install containers from the Pantacor One Marketplace or from one of your own. Choose the Pantacor One Marketplace. A list of selectable apps will appear where you can make your changes and press `OK`.

If you now run `pantabox` and select `view`, you will see see the new revision components:



To add, commit and apply your changes, execute these commands:

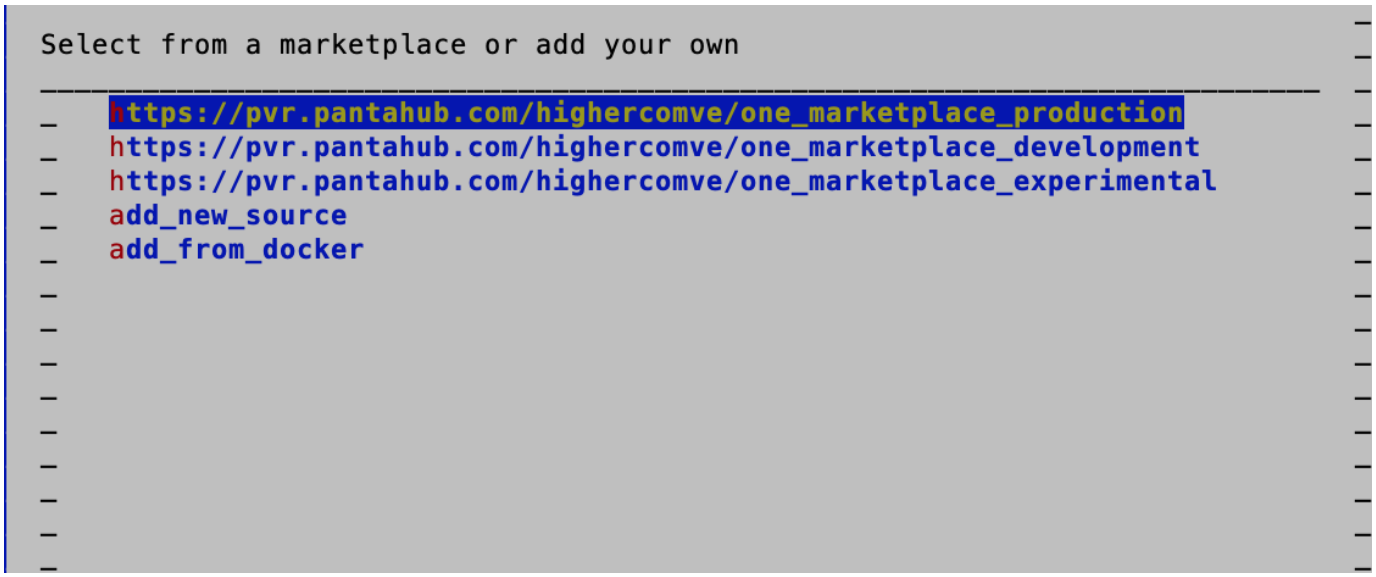
```
pvr add .
pvr commit
exit 0
```

Add Apps from Docker

> **Note**
>
> An app is a container that can be run in a Pantavisor device that takes part on the application level functionality.

Pantabox offers the possibility to add apps generated from an image from Docker Hub. First, execute the `pantabox` command to get its menu:

Select the `install` option of the menu and you will get this:

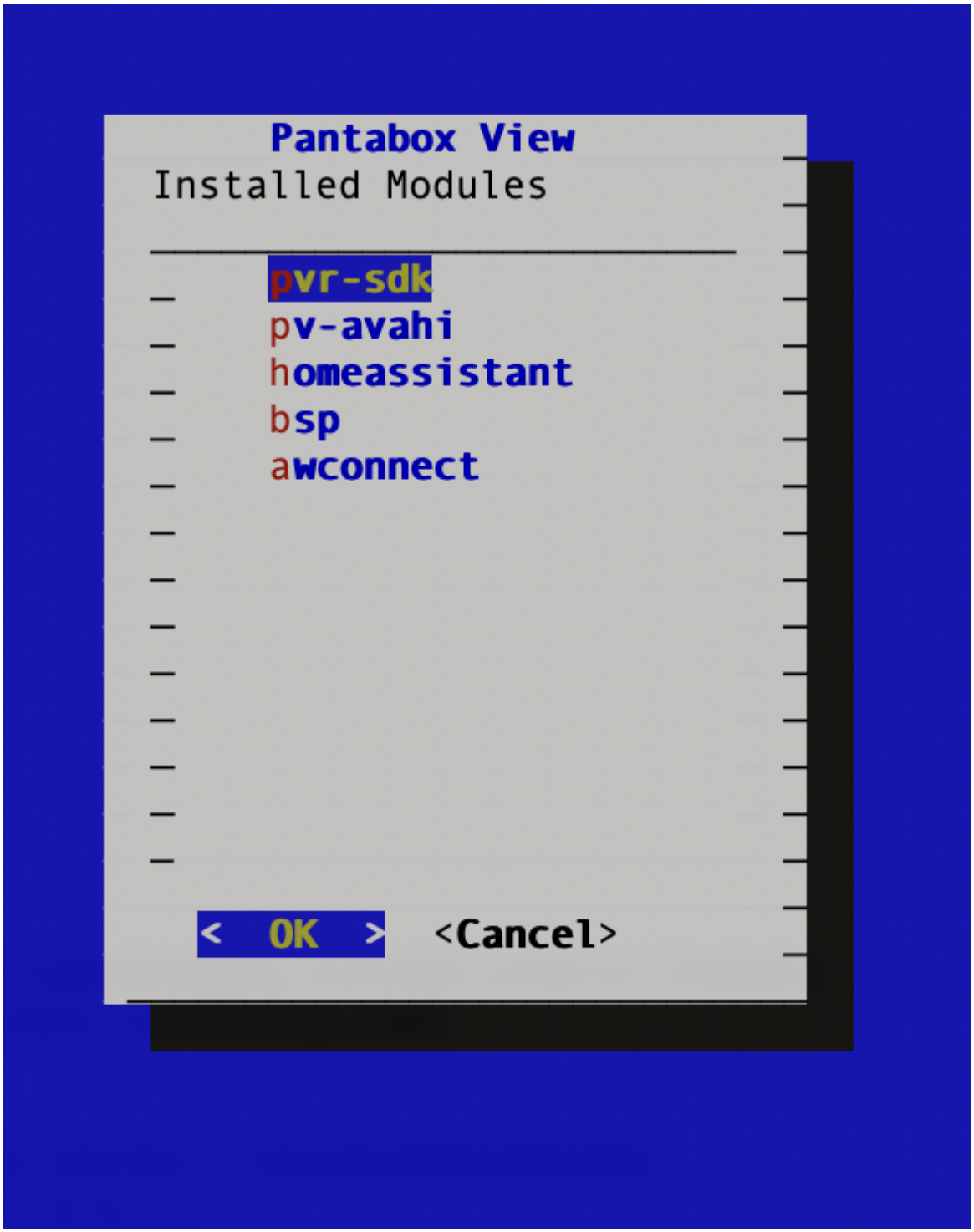


On the install menu, choose `add from docker` and enter the docker name and tag. For example to install the Nginx image from Docker Hub:

```
nginx:latest
```

The `pantabox` and select `view` option should show the new Nginx app.

If you now run `pantabox` and select `view`, you will see see the new revision components:



To add, commit and apply your changes, execute these commands:

```
pvr add .
pvr commit
exit 0
```

Add Additional Files to Apps

Pantabox offers the user the possibility to make changes in your revisions without going through the cloud. Display the menu with the `pantabox` command, select `edit-config` and then choose a container.

The configuration for that container can be added in this path:

```
_config/<container>/
```

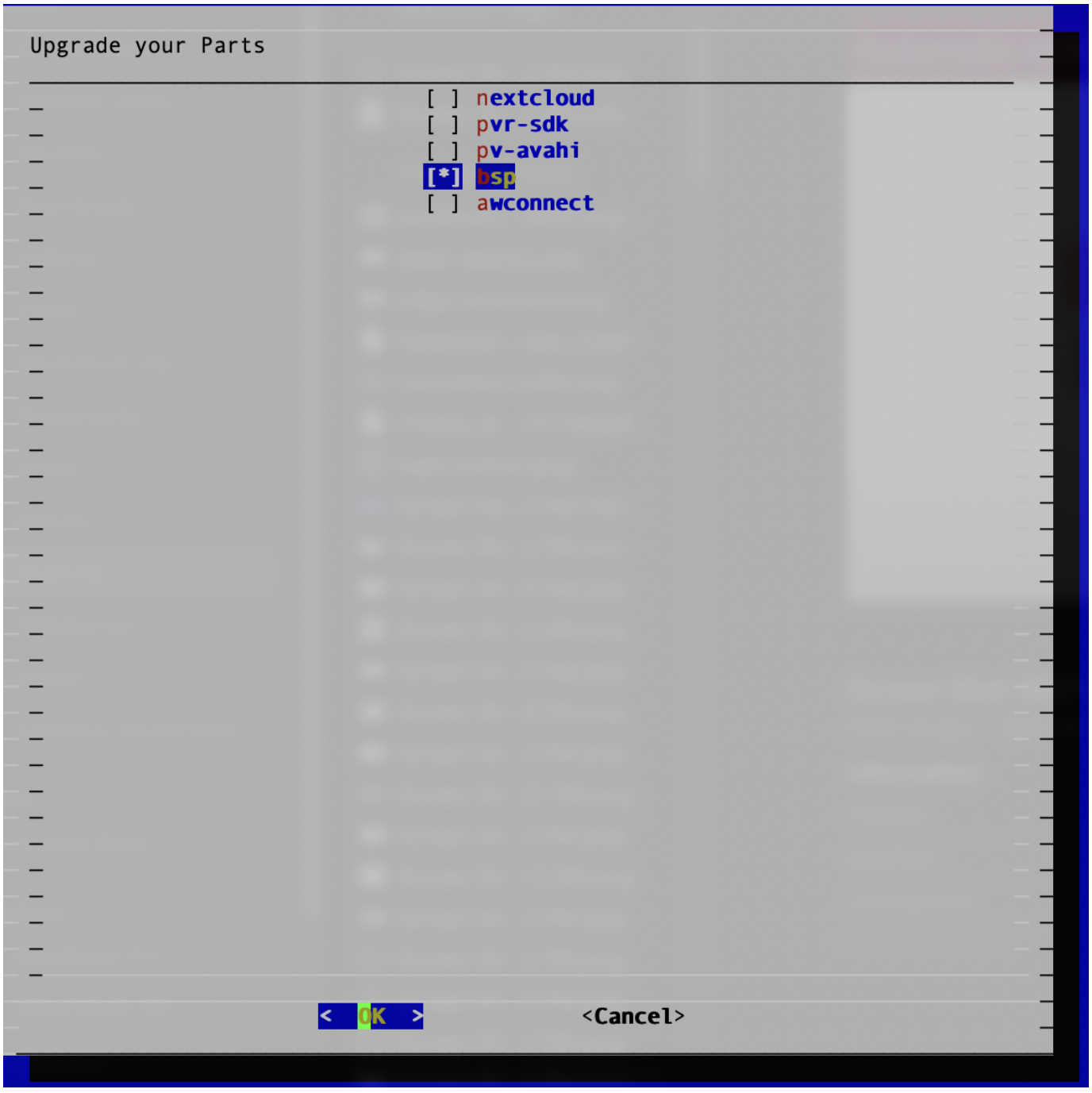
To add, commit and apply your changes, execute these commands:

```
pvr add .
pvr commit
exit 0
```

Update your BSP

You can also update your BSP or any installed container with Pantabox.

Run the `pantabox` command and select `update`. Then, select the `BSP`:



Pantabox will update the selected elements and run the new revision. Pantavisor will test and rollback to a previous revision if needed.

4.2.3 From the Device Web UI

Open the pvtx API

> **Note**
>
> It is important to notice that this operation will open a port for development of a device in your LAN, and thus is insecure and not suited for production devices.

This is necessary if you need to use either pvtx UI or locally manage your device using pvr. It is only going to be possible if the device has the pvr-sdk container running, which is available in our initial images. The `pvtx` API will serve as an endpoint for communication with the both the `pvtx` UI and `pvr` and then will direct the orders to Pantavisor via pvcontrol.

> **Note**
>
> Bear in mind that opening your device to local `pvr` will not close the door to remotely manage it, the same way a remote device can be locally controlled. After a local revision is installed, the device will stop communicating altogether with Pantacor Hub, but this can be reverted by going back to a remotelly installed revision or revision 0.

What we are going to do here is to add a configuration overlay for pvr-sdk. More specifically, we are going to change a configuration file to open the endpoint for hosts in your local network. This can be achieved locally, by using pantabox or remotely, using pvr itself. In any of the two cases, we are going to create this configuration file in the new revision:

```
_config/pvr-sdk/etc/pvr-sdk/config.json
```

With this content:

```
{
  "httpd": {
    "listen": "0.0.0.0",
    "port": "12368"
  }
}
```

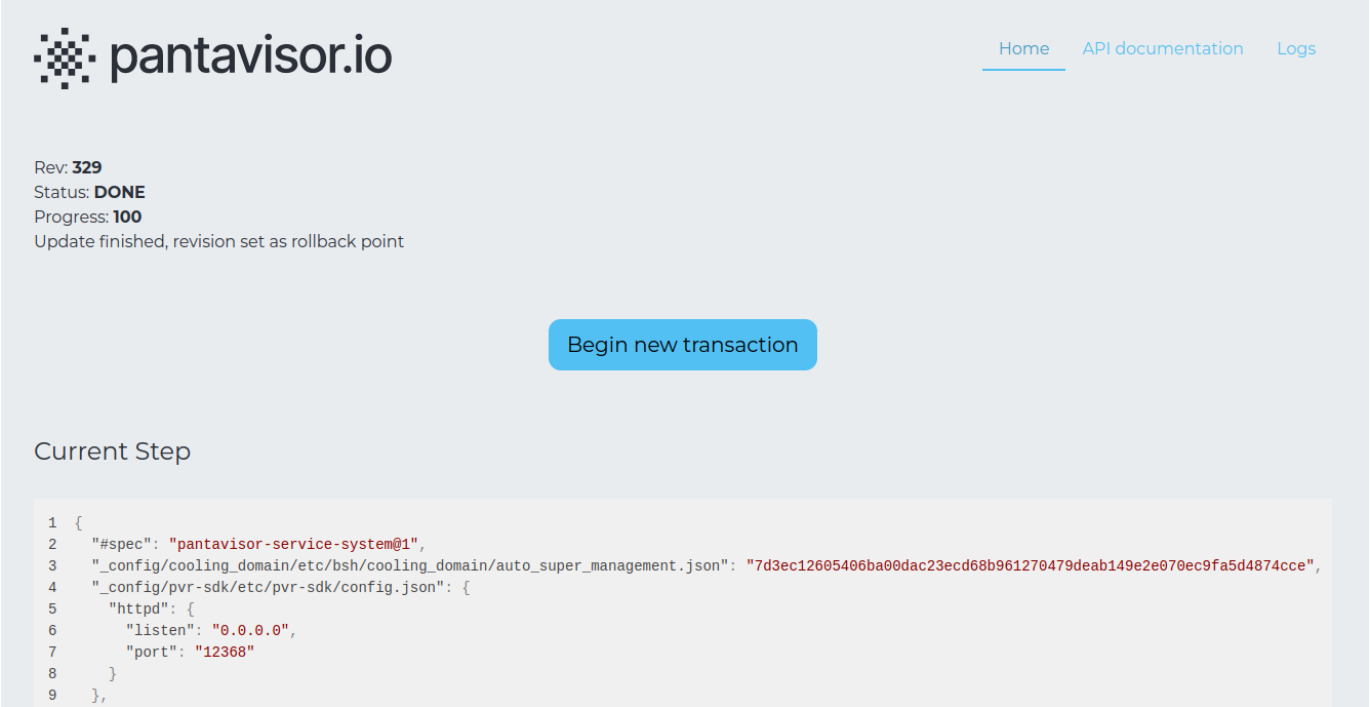
This will allow you to perform cloning and posting revisions to your device IP in your local network using the `12368` port. Also, it will open the `pvtx` UI in that port.

Using pvtx

To access the `pvtx` UI, just make sure your host computer is in the same network and access this URL from your web browser. Make sure to use your device IP and configured port:

```
http://192.168.2.55:12368/app/
```

This will get you access to the `pvtx` UI, where you can see some basic device info, make changes on the running revision and inspect the logs.



4.2.4 From the Host Console

Install pvr

The `pvr` tool enables you to interact with your Pantavisor-enabled devices both remotely through Pantacor Hub and locally. With this CLI tool, you can `clone` your device and `post` different revisions to your device for testing and seamless onboarding, among other operations.

GET PVR STABLE

You can download it from the following locations:

- Linux/AMD64
- Linux/ARM32V6
- Linux/ARM64V8
- Darwin/AMD64
- Darwin/ARM64
- Windows/x32
- Windows/x64

INSTALL THE PVR BINARY

Linux

To install the Linux version you need to extract and place the binary in your `$PATH`:

```
tar xvzf pvr-033.linux.amd64.tar.gz
mkdir -p ~/bin
cp pvr ~/bin/pvr
chmod +x ~/bin/pvr
export PATH=$PATH:~/bin
```

Windows

To install the Windows version all you need is unzip the binary and place it in a directory to which your user has access and can run executables from. `C:\Users\YOURUSER` is usually a good location.

TEST PVR

Once you have it installed just calling the `pvr` command from your shell should show you the help menu, where you can get familiarized with the different features that it provides.

GET LAST PVR VERSION

> **Note**
>
> If you follow this instructions, you will get a non-stable release of pvr, which could mean that you may encounter some bugs.

If you prefer to use the last `pvr` version instead of using the stable one, you can do the following:

```
pvr global-config DistributionTag=develop
pvr self-upgrade
```

The stored binary will automatically upgrade to the last version.

> **Note**
>
> If you find any bug or stability issue while using the develop tag release, you can give us feedback using our community forum.

Discover your Device

You can use `pvr` to discover your running devices in your local network.

> **Note**
>
> This assumes the device is running the pv-avahi container, as is the case with our initial images, and that your computer is connected to the same local network.

To list the devices in your local network, just execute:

```
pvr scan
```

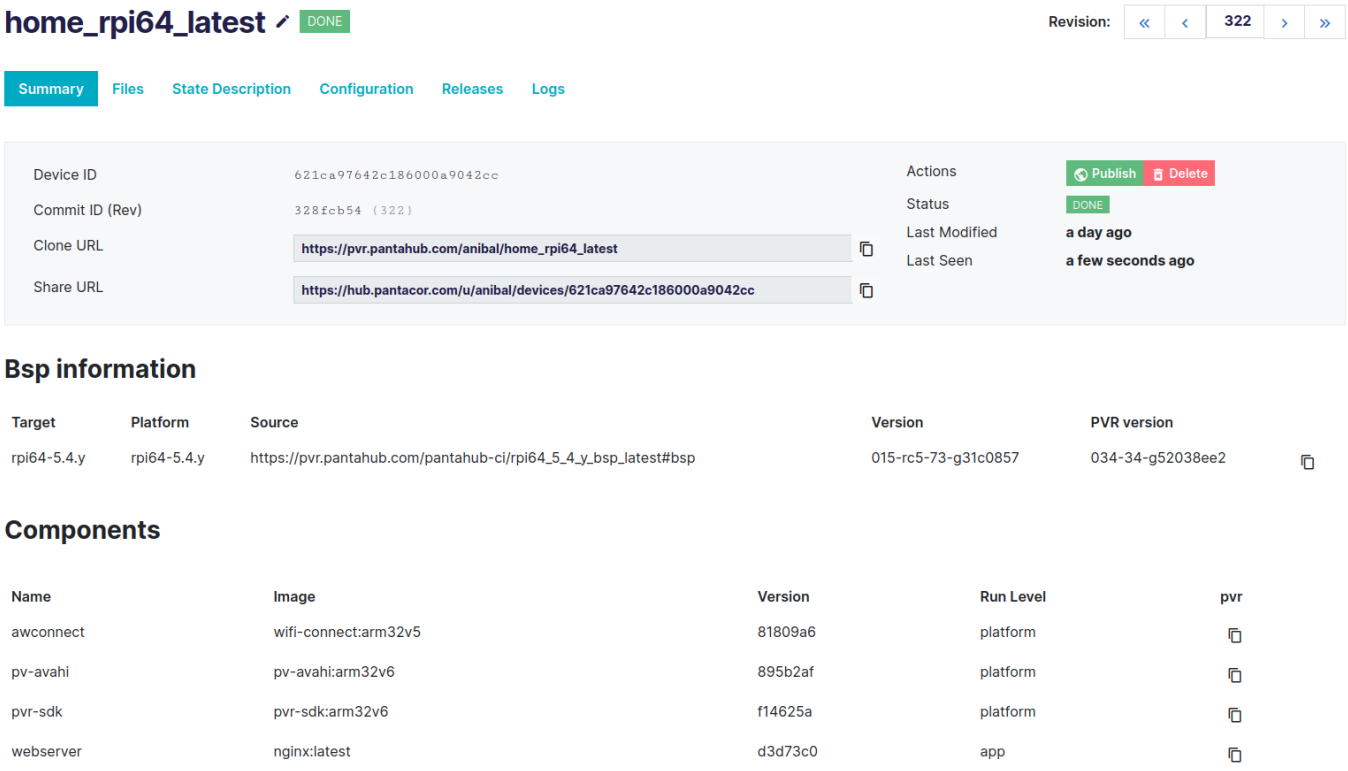
As you can see, claiming information is provided for non-claimed devices for remote managemt. This can be ignored though, if you prefer to locally manage it.

Clone Your Systems

The first step to start making modifications to your new device using `pvr` is always to `clone` any of the revisions of it or of any other device. This is done one way or another depending on whether you aim to manage your device locally or remotely. Once the revision is cloned, modification and deployment to the device will be done the same way for both paths from the `pvr` user point of view.

FROM PANTACOR HUB CLOUD

Pantavisor-enabled systems automatically sync their revisions on first contact with their Pantacor Hub instance immediately after being claimed.



To clone a device, copy the `pvr Clone URL` that appears in the Pantacor Hub device dashboard:

```
pvr clone https://pvr.pantahub.com/user1/device1 my-checkout
```

After executing this, `pvr` asks you to log in, then it downloads the objects and unpacks them locally on disk.

FROM THE DEVICE ITSELF

You can also clone the current running revision of a device that is connected to your local network if the pvtx API is opened.

First, you will need to know your device IP, which is possible with pvr.

After that, just clone the revision into your computer with this command:

```
pvr clone 192.168.1.122 my-checkout
```

Make a New System Revision

Once you have the device cloned on disk, you can make changes to the runtime state on your host computer and once happy, use `pvr commit` to stage the current state as a checkpoint. This new revision can be then sent as a new update to be consumed by your device.

CHECK THE CHANGES

To see current, non-staged file changes, use the `pvr status` command:

```
pvr status .
```

The output of this command will list after-clone changes following this code:

- A - file is not index, but have been marked as added
- C - file is in index, but disk version has changed
- ? - file is not in index and has not yet been added. Commit will ignore if not added
- D - file is in index, but disk version does not have this file. Commit will remove file from index

To see some JSON diff output of the JSON state you can also use `pvr diff`:

```
pvr diff
```

ADD NEW FILES

To add new files so they can be commited:

```
pvr add .
```

COMMIT THE CHANGES

To index the current changes use `pvr commit`:

```
pvr commit
```

Deploy a new System Revision

Once you have done some changes, you can deploy the currently staged state to any device for which you have owner permissions.

For that, use `pvr post` command. This command has an optional argument where you can specify a `pvr clone URL` for the device you want the currently staged system state to be posted to. When not explicitly specified, `post` will send the new revision to the device whose state was originally cloned:

```
pvr post
```

Once post has been submitted to a device, the device will eventually wake up and try to consume the new state.

Add Apps from Docker

> **Note**
>
> An app is a container that can be run in a Pantavisor device that takes part on the application level functionality.

To add new apps, you would use `pvr app add` in a cloned device revision. To add the `nginx` container with the tag `latest` from Docker Hub:

```
pvr app add --from nginx:latest webserver
```

If you wanted to get the docker image from your host computer, you would use the `--source` option:

```
pvr app add --from custom-nginx --source local webserver
```

In either case, it will generate a `webserver/` folder with a matching `src.json`. You can inspect the produced output and use `pvr` to commit and post the new revision to your device:

```
# check status of files on disk
$ pvr status
? webserver/lxc.container.conf
? webserver/root.squashfs
? webserver/root.squashfs.docker-digest
? webserver/run.json
? webserver/src.json

# add new files to pvr control
$ pvr add .

# commit new files
$ pvr commit
Adding webserver/lxc.container.conf
Adding webserver/root.squashfs
Adding webserver/root.squashfs.docker-digest
Adding webserver/run.json
Adding webserver/src.json

# post changes to your device
$ pvr post
```

To ensure that revisions are 100% reproducible, we include the docker digest that `pvr` consumed during during the installation in the src.json.

Add Additional Files to Apps

We have seen how apps can be added and updated. Now, we are going to add or modify files inside of the app file system without having to change modify the app artifacts. To do so, you can take advantage of our additional files feature.

Let us see how to do it for the container from our previous example. We are going to add a new _config directory to the revision with the name of the app and the path inside of its file system:

```
mkdir -p _config/webserver/etc
echo "hello world!" > _config/webserver/etc/test
```

To commit and push the update:

```
pvr add .
pvr commit
pvr push
```

Now, if you inspect the `webserver` app, you will find the new file:

```
# cat /etc/test
hello world!
```

Update your Apps

In order to get the latest bits by docker name (for instance, if the branch tag has been changed or moved forward), you can use `pvr app update`. To update the `webserver` from our previous example:

```
# update to latest "nginx:latest"
$ pvr app update webserver
Application updated

# confirm that there are were changes
$ pvr status
C webserver/src.json

# commit and post
$ pvr commit
Committing /tmp/ppp/api2-canary-01/.pvr/objectswebserver/src.json

$ pvr post
```

This will update your app based on the data extracted from src.json, which can be modified by the user. If you want to only change the configuration and avoid updating, you can also use the `pvr app install` command, which is the equivalent to `pvr app update` for configuration changes in src.json.

Bring Components from Other Devices

Both Apps and the the BSP can be brought from other devices using the `pvr merge` command. For example, to merge the full `x64_initial_latest` device into your checkout:

```
pvr merge https://pvr.pantahub.com/pantahub-ci/x64_initial_latest
```

`pvr` will notify you to use `pvr checkout` to make the changes affective in your workspace. After that, you can either keep making changes or `post` to your device:

```
pvr checkout
pvr post
```

You can also filter the component that is going to be merged by using the `#` prefix. To get only the `pvr-sdk` contianer elements from the device:

```
pvr merge https://pvr.pantahub.com/pantahub-ci/x64_initial_latest#pvr-sdk
pvr checkout
pvr post
```

Bear in mind that merge will keep the elements that were present in your checkout while overwriting the new ones. If you want to completely ignore the old ones and just overwrite everything, use `pvr get`, with the same syntax as `merge`:

```
pvr get https://pvr.pantahub.com/pantahub-ci/x64_initial_latest#pvr-sdk
pvr checkout
pvr post
```

Furthermore, tarballs can be created from a device to enable a local `merge`, `get` or `import`:

```
pvr export /tmp/pvr-export.tgz
```

These can then be imported, merged or get the same way as we did with the `Clone URL`:

```
pvr import /tmp/pvr-export.tgz
pvr checkout
pvr post
```

Finally, you can also `merge` or `get` the `BSP` produced by a Pantavisor build.

```
pvr get out/rpi64-5.10.y/rpi64-5.10.y.pvr.tgz#bsp/,_sigs/bsp.json,device.json
pvr checkout
pvr post
```

Get the Logs

Logs can be downloaded from Pantacor Hub using `pvr` too.

For example, if you want to get the Pantavisor logs from device `5df394ec94a09300095bfab7` and revision `291`:

```
pvr device logs --rev 291 --source=/pantavisor.log 5df394ec94a09300095bfab7
```

4.3 Advanced Features

4.3.1 Secure your Revision

This page shows how to make Pantavisor check the integrity of your revision using the secureboot feature.

> **Note**
>
> Bear in mind that secureboot is set to *lenient* by default in the configuration. This means that only the signed artifacts will be verified by Pantavisor but if we wanted it to check that all artifacts in the revision were signed, we would need to set it to *strict*.

Sign with Pantavisor Keys

First let's start by using Pantavisor Keys. This is convinient for testing purposes only, as private keys are publicly available and thus everyone can sign the artifacts.

By default, `pvr` with work with Pantavisor Keys. To inspect the signature state of a checkout:

```
cd my-checkout
pvr sig ls
```

This will return something similar than to this:

```
{
    "protected": [
        "#spec",
        "_hostconfig/pvr/docker.json",
        "awconnect/lxc.container.conf",
        "awconnect/root.squashfs",
        "awconnect/root.squashfs.docker-digest",
        "awconnect/run.json",
        "bsp/drivers.json",
        "bsp/firmware.squashfs",
        "bsp/kernel.img",
        "bsp/modules.squashfs",
        "bsp/pantavisor",
        "bsp/run.json",
        "bsp/trail.config",
        "device.json"
    ],
    "excluded": [
        "_sigs/awconnect.json",
        "_sigs/bsp.json",
        "awconnect/src.json",
        "bsp/build.json",
        "bsp/src.json"
    ],
    "notseen": [
        "pv-avahi/lxc.container.conf",
        "pv-avahi/root.squashfs",
        "pv-avahi/root.squashfs.docker-digest",
        "pv-avahi/run.json",
        "pv-avahi/src.json"
    ]
}
```

Under the `protected` lists, you will see the checkout elements that are signed, in `excluded` those that are acknowledged but not signed and, in `notseen` all that are not acknowledged. It is important to notice that the `notseen` list must be empty if we want to post updates when the device is in `strict` secureboot mode.

To sign the pv-avahi container:

```
pvr sig add --raw pv-avahi --include 'pv-avahi/**' --exclude _sigs/pv-avahi.json --exclude pv-avahi/src.json
pvr add .
pvr commit
pvr post
```

This will completely sign the rest of the checkout. You can also inspect each signature to see how it is formed:

```
cat _sigs/pv-avahi.json  | jq -r .protected | base64 -d | jq
```

In the case of the newly signed signature:


```
{
  "alg": "RS256",
  "jwk": {
    "kty": "RSA",
    "n": "v4IkL530XFY2OUxhoo9eQifuNlQFOf8lGnZ0RUFJoZNDQ5iGEiqyxM0Ki9XyQJUWX6DzWcyRUVORIIKoaE-rfPwaHuBgNT4WcdsF3z54lXd9MClug_z9ylH-
TSKPaJjoNFP2MWC_Jyuu1lnnqnb5GsRT5XXuFP813pT9g_n_FaVl3nzwLRt9fhG8k7cS92RrGf4GZjBv3PKNFrV8poNfuh29Xrmq0Vsy-TjO5JaOd6o5EqxFepJNs-Kmf1GLf9_a-k7B7aD2AgJc-
aJOFgeDV3mPPDIq_nkOxomap4kveGDcnLLp86OUdPBZ-ecdyxfhQOzern8sj8845nhfZZOFCw",
    "e": "AQAB"
  },
  "pvs": {
    "include": [
      "pv-avahi/**",
      "_config/pv-avahi/**"
    ],
    "exclude": [
      "pv-avahi/src.json",
      "_sigs/pv-avahi.json"
    ]
  },
  "typ": "PVS",
  "x5c": [
    "MIIDajCCAlKgAwIBAgIRAKBshe/
AwP9db6uOSILSu8IwDQYJKoZIhvcNAQELBQAwHDEaMBgGA1UEAwwRUGFudGF2aXNvciBEZXYgQ0EwHhcNMjIwNjEyMTAwMDE4WhcNMjQwOTE0MTAwMDE4WjAaMRgwFgYDVQQDDA9wdi1kZXZlbG9wZXItMDEwggE
iMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC/giQvnfRcVjY5TGGij15CJ+42VAU5/yUadnRFQUmhk0NDmIYSKrLEzQqL1fJAlRZfoPNZzJFRU5EggqhoT6t8/Boe4GA1PhZx2wXfPniVd30wKW6D/
P3KUf5NIo9omOg0U/YxYL8nK67WWeeqdvkaxFPlde4U/zXelP2D+f8VpWXefPAtG31+EbyTtxL3ZGsZ/gZmMG/c8o0WtXymg1+6Hb1euarRWzL5OM7klo53qjkSrEV6kk2z4qZ/UYt/
39r6TsHtoPYCAlz5ok4WB4NXeY88Mir+eQ7GiZqniS94YNycsunzo5R08Fn55x3LF+FA7N6ufyyPzzjmeF9lk4ULAgMBAAGjgagwgaUwCQYDVR0TBAIwADAdBgNVHQ4EFgQU1e3xZhHRBJz2JJJyR16f2LG22GMw
VwYDVR0jBFAwToAUWYuQ+pKC4IbpmCmvE/
j4IeqRgT2hIKQeMBwxGjAYBgNVBAMMEVBhbnRhdmlzb3IgRGV2IENBghRjcQZ0eiqqtBk80eBjIASTScWJyjATBgNVHSUEDDAKBggrBgEFBQcDAzALBgNVHQ8EBAMCB4AwDQYJKoZIhvcNAQELBQADggEBABW0UC
zH5m7kq4Q4b/jzQK3NPy9PF8p9lh8Ks5dvHIAQbLmMKyFbXQirnXqeeeHX9+MTE6L7mJUZ6EZobe1jiJ6TG3prq83WrQHEhBPWY95aldPAnXBxHOKz9o/
qa3+eUYXuhMZ+PPbYdcmZoe82A77JVsG1JFceiDlXVzb8/+jumgx0U3sXW7aAr1ZWaM4VjZLtlfwBOJ/QCM8ltRaR2Oz7mTjfU53ZX3Q8sVATGeUEhGlSzHHTB4/oMgFvwahOifSy2nRME/Ay/CjL3/H21E0GDp/
iBoAgPo7/AhMdE9iF2WgtcAwEUEpvjvWI4Yk7Ybm33PItThl7CCNco4uNO9o="
  ]
}
```


When consuming the new update, Pantavisor will validate that the `_sigs/pv-avahi.json` file. If that fails in *lenient* or *strict* mode, the new revision be aborted before being isntalled.

Sign with your Own Keys

To get your keys into a Pantavisor device, you will have to build your own Pantavisor BSP. You can check how keys are included in the project, taking our Pantavisor Keys repo as an example. This repo will be download to your host machine when syncing the BSP source code so you can fork that repo and continue from there.

Once that is done, you will have to tell `pvr` how to get the private keys for signing your revision. To do that, you can use these environment variables:

```
PVR_SIG_KEY=path/key.default.pem
PVR_X5C_PATH=path/x5c.default.pem
PVR_SIG_CACERTS=path/cacerts.default.pem

export PVR_SIG_KEY PVR_X5C_PATH PVR_SIG_CACERTS

chmod 600 $PVR_SIG_KEY
chmod 644 $PVR_X5C_PATH
chmod 644 $PVR_SIG_CACERTS
```

Now, you can follow the previous section how-to and `pvr sig` will use your own keys.

4.3.2 Load Managed Drivers

Pantavisor offers a declarative framework that allows a Pantavisor BSP to declare which abstract drivers they offer and how those are mapped to loadable modules and Pantavisor containers to specify which drivers they can or need to use to be functional.

Pantavisor runtime manages the lifecycle of drivers used through this framework, e.g. it loads and unloads them at the appropriate time (usually before the container that needs it starts) or through manual operations on REST API method that it offers to containers with privileges.

The syntax for defining drivers available in a BSP, you can visit the define managed drivers for BSPs section.

The syntax how a container defines the drivers it wants to use can be found here.

For the rest of this guide we will look at an example on how to declare an abstract `wifi` driver in a BSP and how to map that to an underlying kernel module. Further we will then show how containers can add their driver requrements to automatically or manually load drivers through Pantavisor.

Driver Definition in a BSP

Firstly, we are going to define the list of managed drivers at the BSP level in the pvr checkout.

For that, we are going to create a new bsp/drivers.json:

```
{
    "#spec": "driver-aliases@1",
    "all": {
        "wifi": [
            "iwlwifi ${user-meta:drivers.iwlwifi.opts}"
        ]
    }
}
```

We have set a new driver named *wifi*, that is composed solely by one Linux module, *iwlwifi*. Additionally, we have added the ability to set arguments for `modprobe` using the user metadata key *drivers.iwlwifi.opts*.

Then, we need to tell Pantavisor that the *wifi* driver will be available from the containers. We do so by adding this information to container/run.json. In this case, we are setting the drivers on *manual* mode, as we want to load it at any point from the container, but we could set *required* or *optional* of we wanted it to be loaded as soon as the container started:

```
{
    ...
    "drivers": {
        "manual": [ "wifi" ],
    }
}
```

You can avoid to manually edit container/run.json, as pvr offers templating support to generate the right run.json.

For that, you have the template "args" keys PV_DRIVERS_REQUIRED, PV_DRIVERS_OPTIONAL and PV_DRIVERS_MANUAL available in src.json.

See the example below:

```
{
    ...
    "args": {
        "PV_DRIVERS_MANUAL": ["wifi"]
    }
}
```

Don't forget to re-run `pvr app install APPNAME` after changing src.json to apply the variables to the runtime configs.

modprobe Arguments

Each abstract driver name is mapped to one-or-many module entries in the json array. Here each line can be the full argument set of modprobe, so you can define static arguments in a very simple manner like:

```
{
    "#spec": "driver-aliases@1",
    "all": {
        "wifi": [
            "iwlwifi 11n_disable=1"
        ]
    }
}
```

The above will pass 11n_disable=1 to the modprobe.

Further each line has a bit of a template syntax that allows you to add values from user-meta or device-meta entries.

First an example on how to use user-meta:

```
{
    "#spec": "driver-aliases@1",
    "all": {
        "wifi": [
            "iwlwifi power_save=${user-meta:drivers.iwlwifi.powersave} 11n_disable=${user-meta:drivers.iwlwifi.11n_disable}"
        ]
    }
}
```

Here a made up example of using device metadata:

```
{
    "#spec": "driver-aliases@1",
    "all": {
        "wifi": [
            "iwlwifi somethingelse=${device-meta:drivers.iwlwifi.somethingelse}"
        ]
    }
}
```

More info about user metadata can be found in the state JSON. User metadata can be set using the control socket in local mode. In remote mode, you would do that through Pantacor Hub.

From the container and in local mode, you can set user metadata with the following curl command.

```
$ curl -X PUT --unix-socket /pantavisor/pv-ctrl \
    --data="power_save=true 11n_disable=1"  \
    http://localhost/user-meta/drivers.iwlwifi.opts
```

If you prefer, you can also do this with pvcontrol.

This will practically result in a module being loaded using the following modprobe command next time the "wifi" driver is loaded:

```
$ modprobe iwlwifi power_save=true 11n_disable=1
```

Load Driver

Drivers are automatically loaded by Pantavisor if they are either defined as "required:" or "optional:".

For "required:" drivers Pantavisor will fail to start the container and rollback if a required driver is not offered by BSP.

Drivers can also be of type "manual" in which case the container gets the ability to manage the lifecycle manually through Pantavisor ctrl socket REST API::

```
$ curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/load
```

If this fails, the HTTP call will return a code 500. Containers that cannot operate without the driver loaded should then do the neede to mark themselve as UNHEALTHY.

4.3.3 Push App Logs to Pantahub

By default, some logs are directed to the log server. These logs are stored on-disk and pushed to Pantacor Hub. This behavior can be partially or totally disabled using pantavisor configuration, but can also be expanded at revision level in the state JSON.

In this tutorial, we are going to see how to add more log files to the list of logs that Pantavisor is already storing on the device and sending to the cloud.

Logs are classified in two types:

- LXC (App startup) logs and console logs.
- Logs from within a container. For example: syslogd or messages.

An example on how to push additional log files to Pantahub

For this example, we have an app named alpine-nginx comprised of an Alpine rootfs with a nginx server installed in it. We want to enable syslog and ngingx log. We will also override the configuration for lxc log to increase its stored size to 4 MiB.

As we were saying in the introduction, it is going to be necessary to modify the state JSON. Let us first check how alpine-nginx/run.json looks by default in the checkout of our cloned device:

```
{
    "#spec": "service-manifest-run@1",
    "config": "lxc.container.conf",
    "name":"alpine-hotspot",
    "storage":{
        "lxc-overlay" : {
            "persistence": "boot"
        }
    },
    "type":"lxc",
    "root-volume": "root.squashfs",
    "volumes":[]
}
```

To make the changes described above, we would have to add this `logs` object:

```
{
    "#spec": "service-manifest-run@1",
    "config": "lxc.container.conf",
    "name":"alpine-hotspot",
    "storage":{
        "lxc-overlay" : {
            "persistence": "boot"
        }
    },
    "type":"lxc",
    "root-volume": "root.squashfs",
    "volumes":[],
    "logs": [
        {"file":"/var/log/syslog","maxsize":102485760,"truncate":true,"name":"alpine-logger"},
        {"file":"/www/log/wwwlog","maxsize":102485760,"truncate":true,"name":"nginx-logger"},
        {"lxc":"enable","maxsize":409943040,"truncate":true,"name":"alpine-lxc"}
    ]
}
```

5. How to Build

5.1 Apps

5.1.1 Create your Apps

Pantavisor natively runs its containers. If you take a closer look to one of our initial devices, you will see that each platform contains these elements:

```
$ ls rpi3_initial_stable/awconnect/
lxc.container.conf  root.squashfs  root.squashfs.docker-digest  run.json  src.json
```

As you can see, it's just a bunch of configuration files and the compressed file system. Therefore, to develop your own app, you will only have to basically prepare this file system for your container.

There is a number of ways to achieve this, but we use Docker. Remember that the containers themselves are not going to be run with docker. We use it just to get the file system in an easy way from a Dockerfile. We have a number of projects that you can use as examples on we we do this.

After that, we are going to add the resulting image to our project using pvr app add. If you have already gone through the `pvr app add` how-to page, you already know that you can add a Docker image to your device. In the case of our examples, we are just customizing that image.

5.1.2 Communicate with Pantavisor

Your container can communicate with Pantavisor to perform several actions such as setting, removing and listing metadata, reboot, poweroff or even installing your own revisions without going through the cloud.

This communication is done through our pv-ctrl socket using HTTP protocol. Our pvr-sdk container, which is included in our default initial images, contains the pvcontrol tool. This pvcontrol script can either be directly used if you want to include it in your container (it requires sh and cURL), or you can use it as an example to implement your own client with the HTTP library of your choice.

Take a look at the different supported queries in the commands reference.

5.2 Pantavisor BSP

5.2.1 Environment Setup

The following dependencies are required for building Pantavisor and installing containers. These are for any apt-based system such as Ubuntu, but you can also adapt them for your favorite distro.

Installing Build Dependencies

Every package (with the exception of `repo` in old distros) is available from apt and can be installed with:

```
sudo apt update
sudo apt install curl squashfs-tools git repo docker.io clang-format-11
```

To install Docker, follow the instructions for your particular Linux Distribution here: https://docs.docker.com/engine/install/

We use the repo tool developed by Android project to maintain the Pantavisor source distribution.

Setting Up Build Dependencies

Some dependencies require further setup.

GIT SETUP

Before you can use git, remember to configure your name and email globally:

```
git config --global user.name "Your Name"
git config --global user.email "your@email.tld"
```

DOCKER SETUP

After installing Docker, you'll need to grant rights to your non-root user, so that you can run all of docker's commands without `sudo`. To do so, you can execute the following commands. Make sure to log out and log back in to activate the new group rules.

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

Now check that you can use Docker echo container as a normal user:

```
docker run hello-world
```

Expected output of that command should be similar to:

```
docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:41a65640635299bab090f783209c1e3a3f11934cf7756b09cb2f1e02147c6ed8
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/
```

```
For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

REPO SETUP

It is possible that repo cannot find python3. In that case:

```
sudo apt install python-is-python3
```

Installing Test Dependencies

These dependencies have to be installed if you want to run Pantavisor Tests in a x64 host:

```
sudo apt install binfmt-support docker.io git jq
```

Then, to run ARM binaries on your x64 host that enables wireless network simulated capabilities, it is necessary to install netlink generic compatible qemu binaries from our CI:

```
sudo apt remove qemu-user-static
mkdir ~/bin
wget https://pantavisor-ci.s3.amazonaws.com/qemu/1303841432/qemu-arm -O ~/bin/qemu-arm
wget https://pantavisor-ci.s3.amazonaws.com/qemu/1303841432/qemu-aarch64 -O ~/bin/qemu-aarch64
chmod +x ~/bin/qemu-arm
chmod +x ~/bin/qemu-aarch64
sudo update-binfmts --install qemu-arm ~/bin/qemu-arm --offset 0 --magic "\x7f\x45\x4c\x46\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x28\x00" --
mask "\xff\xff\xff\xff\xff\xff\xff\x00\xff\xff\xff\xff\xff\xff\xff\xff\xfe\xff\xff\xff" --fix-binary yes
sudo update-binfmts --install qemu-aarch64 ~/bin/qemu-aarch64 --offset 0 --magic
"\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\xb7\x00" --mask
"\xff\xff\xff\xff\xff\xff\xff\x00\xff\xff\xff\xff\xff\xff\xff\xfe\xff\xff\xff" --fix-binary yes
```

5.2.2 Choose your Target

> **Note**
>
> This guide focuses on building Pantavisor from source code. If you do not need to compile Pantavisor, pre-compiled Pantavisor images are available for all supported targets in our download page.

Currently supported targets are:

- aarch64-appengine
- aarch64-generic
- arm-appengine
- arm-bpi-r2
- arm-bpi-r64
- arm-generic
- arm-odroid-c2
- arm-toradex-colibri-imx6_7
- beaglev
- nv-tegra-4_9
- malta-qemu
- mips-appengine
- mips-generic
- rock64
- rpi0w-5.10.y
- rpi64-5.10.y
- x64-appengine
- x64-generic
- x64-ubuntu
- x64-uefi

Note down the `Target` value of your initial device. For our example, that would be:

```
rpi64-5.10.y
```

This will be enough if you only want to build the BSP, which you can then use to update a Pantavisor-enabled device. In that case, you can continue to the next page of this guide.

Choose your Reference Device

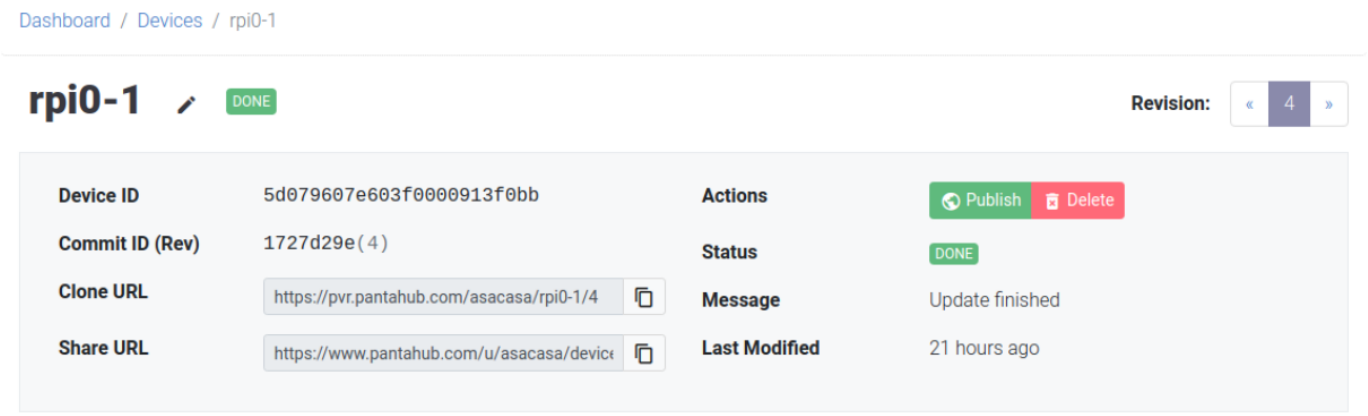
If what to want is to build a flashable image, you will also need a Pantavisor reference device. The final image will contain the BSP generated from source code plus the containers imported from that device. This Pantavisor reference device can be either stored in Pantacor Hub or locally in your machine. By default, if no reference device is specified, the build system will generate an image with no containers (therefore, no network management and no functionality).

For instance, if you want to produce an image that has the same app stack than one of your online devices, you would use the pvr `clone URL` of that device as a parameter to build pantavisor. If you do not have your own custom device developed yet, you can use any of our initial devices as your starting point:

```
https://pvr.pantahub.com/pantahub-ci/rpi64_5_10_y_initial_latest
```

You can get this and other reference devices in our initial devices page, in the `Pantacor Hub devices` column of the table.

If you rather want to use one of the devices that you have developed and burn an image from it, you would go to your device details page on https://www.pantahub.com and get the pvr `Clone URL` from there:



As previously stated, you can also use a device that is already cloned in your machine. In this case, you would use the path to the cloned device checkout plus the `.pvr` suffix. For example:

```
/home/anibal/pantacor/src/devices/rpi64_5_10_y_initial_latest/.pvr
```

5.2.3 Get the Source Code

This build system is based on the Alchemy tool, plus modifications specific to the requirements of the Pantavisor build. First, make a new workspace directory:

```
mkdir pantavisor
cd pantavisor
```

To initialize it, use `repo`. You can download the necessary code using the -g option. Just substitute `rpi64-5.10.y` with the target you want to build.

```
repo init -u https://gitlab.com/pantacor/pv-manifest -m release.xml -g runtime,rpi64-5.10.y
```

You may prefer to download the source code for all targets. In that case:

```
repo init -u https://gitlab.com/pantacor/pv-manifest -m release.xml
```

Now, it is time to download the code with the sync command:

```
repo sync -j10 -c --no-clone-bundle
```

Afterwards, you should have contents similar to below in your Pantavisor directory:

```
alchemy  bootloader  build.docker.sh  build.sh  ca  config
external  firmware  internal  kernel  prebuilt  run.docker.sh
scripts  test.docker.sh  tests  vendor
```

5.2.4 Build Pantavisor

For building, the flow is very similar to the Android build system, except that targets are not setup in advance of operations but rather these are conditional on the target and the flow is managed by scripts/build.sh.

Build your chosen reference device

Once you have chosen a target, a reference device and downloaded the Pantavisor source code, you can now start the building process.

First of all, check you still are in your Pantavisor workspace path. Output of `ls` should look similar to:

```
alchemy  bootloader  build.docker.sh  build.sh  ca  config
external  firmware  internal  kernel  prebuilt  run.docker.sh
scripts  test.docker.sh  tests  vendor
```

Now, you can execute the `build.docker.sh` script, optionally using the PVR_MERGE_SRC build option if what you want is a fully flashable image:

```
PVR_MERGE_SRC=https://pvr.pantahub.com/pantahub-ci/rpi64_5_10_y_initial_latest ./build.docker.sh rpi64-5.10.y
```

You can directly clone a device and use the local checkout path adding the `.pvr` suffix:

```
pvr clone https://pvr.pantahub.com/pantahub-ci/rpi64_5_10_y_initial_latest my-checkout
PVR_MERGE_SRC=my-checkout/.pvr ./build.docker.sh rpi64-5.10.y
```

> **Note**
>
> For other advaced building options and configuration, check our build options, build components or Pantavisor configuration references.

This will compile all dependencies of the `rpi64-5.10.y` target build tree and produce:

- A flashable image according to the values defined in `config/rpi64-5.10.y/image.config`. You can find the generated image in `out/rpi64-5.10.y/rpi64-5.10.y-pv-1024MiB.img` that you can use to flash your device.
- A BSP tarball in `out/rpi64-5.10.y/rpi64-5.10.y.pvr.tgz` to update your running devices.

5.2.5 Customize your Build

Once you have synced the manifest, you have at your disposal all the source code that you need to build Pantavisor:

```
alchemy  bootloader  build.docker.sh  build.sh  ca  config
external  firmware  internal  kernel  prebuilt  run.docker.sh
scripts  test.docker.sh  tests  vendor
```

There are some recurrent locations you might need to play with in order to customize Pantavisor to fit your needs. If you want to create your own customized build, forking these repositories will be something you might need at some point:

- ca
- config
- vendor

ca/

In this directory, we store the secureboot certificates and keys. Some of files are:

- `ca/pvs/pvs.defaultkeys.tar.gz` : contains the default root certificate.
- `ca/pvs/pvs.oemkeys.tar.gz` : contains the OEM root certificate.

You can check our `ca/` repository for our public devices.

config/

This directory is meant for various platform specific configuration files.

- `config/<platform>/pantavisor.config` : config file for the `PV_` keys.
- `config/<platform>/pantahub.config` : config file for the `PH_` keys.

You can check our `config/` repository for our public devices.

vendor/

This directory is meant for various platform specific vendor files.

- `vendor/<platform>/skel/` : this directory contains files that will be added to the Pantavisor rootfs.
- `vendor/<platform>/stepskel/` : this directory contains files that will be added to revision 0.

You can check our `vendor/` repository for our public devices.

5.2.6 Test Pantavisor

To run our tests locally, you have to first build Pantavisor for the x64-appengine target:

```
PANTAVISOR_DEBUG=yes ./build.docker.sh x64-appengine
```

Then, to run all tests:

```
./test.docker.sh run
```

For more information, take a look at the Pantavisor Test Framework reference.

5.2.7 Debug Pantavisor

Sometimes things don't go as planned and you will need some insight on what is going on. For those occasions, we offer some tools to debug Pantavisor.

Debug coredump using gdb

Pantavisor will generate a Core Dump when a crash occurs. To open this Core Dump, you just have to retrieve it from storage. To do so, you can just SSH cat the file from a device with an already set public key to your host computer:

```
ssh -p8222 10.42.0.158 -l/ cat /storage/corepv > /tmp/corepv
```

To open it in your machine, we will need to tell gdb where both the init binary with debugging simbols and the sysroot path resulted from the building process are located:

```
$ gdb-multiarch /home/anibal/pantacor/src/pantavisor/out/x64-uefi/staging/init /tmp/corepv
(gdb) set sysroot /home/anibal/pantacor/src/pantavisor/out/x64-uefi/staging/
(gdb) bt
#0  0x00007f262bcc01b0 in write () from /lib/x86_64-linux-gnu/libpthread.so.0
#1  0x0000564ec360b04f in mbedtls_net_send ()
#2  0x0000564ec360d0d3 in mbedtls_ssl_flush_output ()
#3  0x0000564ec360d4e2 in mbedtls_ssl_write_record ()
#4  0x0000564ec361073d in ssl_write_real ()
#5  0x0000564ec36107bc in mbedtls_ssl_write ()
...
```

> **Note**
>
> Use your own paths.

Remotely debug using gdbserver and gdb-multiarch

You can also remotely debug Pantavisor while running on your target device from your host computer.

INSTALL GDBSERVER IN YOUR DEVICE

For this guide, we are going to use a Raspberry Pi 3 as the target device. First, we need to make sure gbdsever is installed in your machine. To do so, we need to check the bsp addons in your device checkout look like this:

```
$ cd <device-checkout>
$ cat bsp/run.json | python -m json.tool
{
    "addons": [
        "addon-gdbserver.cpio.xz4"
    ],
    "firmware": "firmware.squashfs",
    "initrd": "pantavisor",
    "linux": "kernel.img",
    "modules": "modules.squashfs"
}
$ ls bsp/
firmware.squashfs  addon-gdbserver.cpio.xz4  kernel.img  modules.squashfs  pantavisor  run.json  src.json
```

If the addon is set in bsp/run.json and the cpio.xz4 file is in the bsp path, you can continue to the next section.

If not installed, you need to get a statically compiled gdbserver with cpio.xz4 compression. We are building this for several targets here. In your device checkout:

```
cd bsp
docker run --rm registry.gitlab.com/pantacor/pantavisor-addons/gdbserver:arm32v6-musl | tar x
```

Then, reference the file in bsp/run.json with a Pantavisor addon so it looks like the example.

Don't forget to follow this how-to guide to add, commit and post these changes to your device.

ATTACH GDBSERVER TO THE INIT PROCESS IN YOUR DEVICE

Once gdbserver is installed in your system, get access to your device via ssh and attach gdbserver to the Pantavisor process:

```
$ ssh -p 8222 /@192.168.1.134
$ ps | grep init
    1 0         1064 S    /init noswap fastboot
   78 0         3208 S    {pantavisor} /init noswap fastboot
...
$ gdbserver --attach localhost:2000 78
Attached; pid = 78
Listening on port 2000
Remote debugging from host ::ffff:192.168.1.132, port 46982
```

The election of Pantavisor's PID is important. In this case, we want to debug something in pantahub.c. This is running in a fork created by init (PID 1) in the early stages of Pantavisor, so we chose the second PID with the *init* keyword, which in this case is PID 78.

REMOTELY DEBUG FROM YOUR HOST COMPUTER WITH GDB-MULTIARCH

Open a new terminal in your host computer. Make sure you have the gdb-multiarch tool installed in your system. In a Debian-based distro:

```
apt-get update
apt-get install gdb-multiarch
```

We need to tell gdb-multiarch where both the init binary with debugging simbols and the sysroot path resulted from the building process are located:

```
$ gdb-multiarch /home/anibal/pantacor/src/pantavisor/out/rpi3/staging/init
(gdb) set sysroot /home/anibal/pantacor/src/pantavisor/out/rpi3/staging/
```

Now, we just have to connect to device IP and port:

```
(gdb) target remote 192.168.1.134:2000
Remote debugging using 192.168.1.134:2000
Reading symbols from /home/anibal/pantacor/src/pantavisor/out/rpi3/staging/lib/ld-musl-armhf.so.1...done.
Reading symbols from /home/anibal/pantacor/src/pantavisor/out/rpi3/staging/lib/pv_lxc.so...done.
Reading symbols from /home/anibal/pantacor/src/pantavisor/out/rpi3/staging/lib/libgcc_s.so.1...done.
0x00055074 in mbedtls_base64_decode ()
(gdb) b internal/init/pantahub.c:290
Breakpoint 1 at 0x17fa6: file internal/init/pantahub.c, line 290.
(gdb) c
Continuing.

Breakpoint 1, pv_ph_is_available (pv=pv@entry=0x76f18e30) at internal/init/pantahub.c:290
290         pv_log(DEBUG, "PH available at '%s:%d'",
(gdb) bt
#0  pv_ph_is_available (pv=pv@entry=0x76f18e30) at internal/init/pantahub.c:290
#1  0x00014b2e in _pv_wait (pv=0x76f18e30) at internal/init/controller.c:446
#2  0x000156ce in _pv_run_state (pv=0x76f18e30, state=STATE_WAIT) at internal/init/controller.c:748
#3  pv_controller_start (pv=pv@entry=0x76f18e30) at internal/init/controller.c:760
#4  0x000147f2 in _pv_init () at internal/init/pantavisor.c:546
#5  pantavisor_init (do_fork=do_fork@entry=true) at internal/init/pantavisor.c:564
#6  0x0001264c in main (argc=<optimized out>, argv=<optimized out>) at internal/init/init.c:276
```

Note

Use your own paths and IP.

5.2.8 Pantavisor Coding Style

Pantavisor coding style is based on the Linux Kernel one with minor modifications. You can check our .clang-format file here.

Check Your Code Style

Before submitting changes to Pantavisor, it is important to check if your code style honors the one defined in the previously mentioned .clang-format. To do so, you can use our building tools:

```
./build.docker.sh rpi64-5.10.y init-codecheck
```

If there is some code mismatch, a detailed error message will appear, like:

```
init: Checking 'c' files...
--- /dev/fd/63  2022-07-19 10:16:04.152675578 +0000
+++ /dev/fd/62  2022-07-19 10:16:04.152675578 +0000
@@ -1453,8 +1453,8 @@
                &config_list, "secureboot.mode", SB_LENIENT);
        config->secureboot.certdir = config_get_value_string(
                &config_list, "secureboot.certdir", "/certs");
-       config->secureboot.checksum =
-               config_get_value_bool(&config_list, "secureboot.checksum", true);
+       config->secureboot.checksum = config_get_value_bool(
+               &config_list, "secureboot.checksum", true);

        config_clear_items(&config_list);

@@ -18196,8 +18196,7 @@
        struct pv_object *o;
        struct pv_json *j;

-       if (getenv("pv_quickboot") ||
-               !pv_config_get_secureboot_checksum()) {
+       if (getenv("pv_quickboot") || !pv_config_get_secureboot_checksum()) {
                pv_log(DEBUG, "state objects and JSONs checksum disabled");
                return true;
        }
make: *** [/home/anibal/pantacor/src/bsp/alchemy/classes/codecheck-rules.mk:60: init-codecheck-c] Error 1

MAKE ERROR DETECTED
```

It is important to fix these errors before pushing them upstream, as the CI will run this same code check command.

Fix Your Code Style

To automatically format your files, you can just use clang-format-13:

```
clang-format-13 -i ctrl.c
```

After this, run the code check command again:

```
./build.docker.sh rpi64-5.10.y init-codecheck
```

This will happily result in success:

```
init: Checking 'c' files...
+ restore_kernel_logo
+ '[' -f /home/anibal/pantacor/src/bsp/kernel/linux-stable/drivers/video/logo/logo_linux_clut224.ppm.backup ']'
+ mv /home/anibal/pantacor/src/bsp/kernel/linux-stable/drivers/video/logo/logo_linux_clut224.ppm.backup /home/anibal/pantacor/src/bsp/kernel/linux-stable/
drivers/video/logo/logo_linux_clut224.ppm -f
```

5.2.9 Pantavisor Contribution Guidelines

In order to keep an homogeneous and encourage thoughtful commit messages, we follow the Conventional Commits Specification for our commit messages:

```
<type>: <description>

[optional body]

[optional footer(s)]
```

In order to land changes, you will need to create a Pull Request first. To do so, you need to create a branch.

For branches we use the type tag from Conventional Commits plus a meaningful name, using the underscore character `_` to separate words:

```
<type>/<branch_name>
```

In the future, we will enforce this through our CI.

5.2.10 Pantavisor SBOM

Pantavisor build system can generate a basic SBOM in SPDX JSON format for a specific target:

```
./build.docker.sh <TARGET> sbom
```

Using this command, the SBOM for the target will be generated be in `out/TARGET/sbom.json`. If you need to create an SBOM including packages from your containers you can use PV_MERGE_SBOM to specify a path where Pantavisor look for SBOM documents, extract its packages and adds them to the Pantavisor's SBOM

6. Technical Overview

6.1 Pantavisor Architecture

At a high level, Pantavisor is just in charge of two things: container orchestration and communication with the outside world.

6.1.1 Container Orchestration

The software that is running in a Pantavisor-enabled device at a certain moment is called a revision. A revision is composed by a BSP and a number of containers. Pantavisor is able to go from the current running revision to a new revision in a transactional manner.

Revisions, as well as other relevant data, are stored on-disk in the device to make them persistent.

6.1.2 Communication with the Outside World

Pantavisor-enabled devices need to communicate with the outside world to consume new updates, exchange metadata or send logs. Users can achieve this remotely as well as locally.

6.1.3 Customisation

Pantavisor can be set up in different ways, offering multi-level configuration as well as several operational init modes.

6.1.4 Service Mesh

Containers running in the same revision can communicate with each other through `pv-xconnect`, Pantavisor's built-in service mesh. It handles service discovery and resource injection (Unix sockets, D-Bus, DRM, Wayland) between containers in a secure and declarative way. See xconnect.

6.1.5 State Machine

To get a very simplified view on how Pantavisor works, you can take a look at its state machine:

Each state can be summarized as:

- **INIT**:
  - Prepare the base rootfs
  - Prepare persistent storage
  - Parse configuration
  - Choose init mode
  - Initialize interaction with bootloader
  - Initialize log system
  - Initialize control socket
  - Initialize watchdog
- **RUN**:
  - Initialize the running revision
  - Check object checksum
  - Check revision signatures
- **WAIT**:
  - Start containers and check its statuses
  - Manage any ongoing update, incluiding installation, verification, transition, etc.
  - Run Pantacor Hub client full state machine
  - Process control socket requests
  - Manage metadata
  - Check and run garbage collector

6.2 Revisions

A revision is composed by a BSP (Pantavisor binary, Linux kernel, modules and firmware) plus a number of containers.

In order to make revisions reproducible, they can be defined in a state JSON. This JSON is a flat representation of a set of either binary objects or other inline JSON documents. Here you can take a look at a simple example revision formed by just a BSP and a container named *awconnect*:


```
{
  "#spec": "pantavisor-service-system@1",
  "_hostconfig/pvr/docker.json": {
    "platforms": [
      "linux/arm64",
      "linux/arm"
    ]
  },
  "awconnect/lxc.container.conf": "153d58588b0327f73c8424c214c039fcdd975814bc075bc5c72f82fd3cdfd7b6",
  "awconnect/root.squashfs": "e1ddabe573021b48dd5d66d59593d94fbc57b7a2f85dac59628959ae6955d2e2",
  "awconnect/root.squashfs.docker-digest": "828054813b64d71d26756903010a52828941f6bb0859e878cb70f6f1e0ec7d2d",
  "awconnect/run.json": {
    "#spec": "service-manifest-run@1",
    "config": "lxc.container.conf",
    "name": "awconnect",
    "root-volume": "root.squashfs",
    "storage": {
      "docker--etc-NetworkManager-system-connections": {
        "persistence": "permanent"
      },
      "lxc-overlay": {
        "persistence": "boot"
      }
    },
    "type": "lxc",
    "volumes": []
  },
  "awconnect/src.json": {
    "#spec": "service-manifest-src@1",
    "docker_config": {
      "AttachStderr": false,
      "AttachStdin": false,
      "AttachStdout": false,
      "Cmd": [
        "/lib/systemd/systemd"
      ],
      "Domainname": "",
      "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
      ],
      "Hostname": "",
      "Image": "sha256:a8c4da0f0bde245a971a4a63a205cf56e071611f78b3d650715f309b7cefc57b",
      "OpenStdin": false,
      "StdinOnce": false,
      "Tty": false,
      "User": "",
      "Volumes": {
        "/etc/NetworkManager/system-connections/": {}
      },
      "WorkingDir": "/opt/wifi-connect/"
    },
    "docker_digest": "registry.gitlab.com/pantacor/pv-platforms/wifi-connect@sha256:b2ad073c0a41d186b6338fb8b81714eb1b8da9421383bbf8914fb86a01bbcafb",
    "docker_name": "registry.gitlab.com/pantacor/pv-platforms/wifi-connect",
    "docker_source": "remote,local",
    "docker_tag": "arm32v5",
    "persistence": {},
    "template": "builtin-lxc-docker"
  },
  "bsp/addon-plymouth.cpio.xz4": "beae6a7bb235916cac52bcfece64c30615cded8c4c640e6941e7ecabe53b4920",
  "bsp/build.json": {
    "altrepogroups": "",
    "branch": "master",
    "commit": "e2a4911eb35de2032e85f74c8f239de81c6f622b",
    "gitdescribe": "014-rc14-18-ge2a4911",
    "pipeline": "436189414",
    "platform": "rpi64",
    "project": "pantacor/pv-manifest",
    "pvrversion": "pvr version 026-52-gbf3bd5d6",
    "target": "arm-rpi64",
    "time": "2021-12-24 01:25:27 +0000"
  },
  "bsp/firmware.squashfs": "f37e9699ea8add7042e2843d095e68a316e6344d832b74d41244cb0bca29464e",
  "bsp/kernel.img": "990f8b0fcab8b99f631497753cc55b70f6f522a1d91cd4ae0777a7747b98509e",
  "bsp/modules.squashfs": "0e202a7ee3a575bc502ec3869251a3587a3110079f221fc15c63da1e8d8a08ae",
  "bsp/pantavisor": "1e6561f75cba98500f023e09aae430557fe0d1b02aeb1fa9adb3c2d3b6d250c6",
  "bsp/run.json": {
    "addons": [
      "addon-plymouth.cpio.xz4"
    ],
```

```
    "firmware": "firmware.squashfs",
    "initrd": "pantavisor",
    "initrd_config": "",
    "linux": "kernel.img",
    "modules": "modules.squashfs"
  },
  "bsp/src.json": {
    "#spec": "bsp-manifest-src@1",
    "pvr": "https://pvr.pantahub.com/pantahub-ci/arm_rpi64_bsp_latest#bsp"
  }
}
```


To know more about this, you can take a look at our state JSON reference.

6.3 BSP

In addition to containers, Pantavisor is in charge of the life-cycle of the Linux kernel, modules and firmware. To allow upgrading all this plus Pantavisor itself, all these binaries have been included under the BSP denomination in the state JSON.

6.3.1 Pantavisor

The Pantavisor binary and dependency tree are part of the revision BSP in the state JSON.

Addons

Additional files can be added to Pantavisor initrd rootfs without having to do it during compile time.

It is important to remark that the binary file has to be under cpio.xz4 compression. That is, cpio.xz with 4 Byte alignment. You can take a look at how we do the cpio.xz4 compression at this example. For the rest of gdbserver installation and use, go here.

6.3.2 Linux Kernel

The Linux Kernel, modules and firmware are also part of the revision BSP of the state JSON.

Managed Drivers

Pantavisor offers a declarative way to define a list of drivers at BSP level, each driver being just a set of Kernel modules. Parameters for Kernel loading are supported too by mapping drivers to device or user metadata.

The modules that are part of a driver will only be loaded if referenced from a container. Therefore, the loading order of the drivers will depend on their start up order.

6.3.3 Bootloader

To natively run Pantavisor on a device, it is necessary to have some on-disk artifacts that fall out of the umbrella of Pantavisor revisions. This is the case of the bootloader, which will load the Linux kernel and directly execute the initrd (Pantavisor) after that. Both the bootloader and Pantavisor will interact with Pantavisor storage to communicate with each other so the bootloader kwnows where to find the artifacts to boot.

We support a number of boards, using both U-Boot and GRUB bootloaders. These include all the bring up mechanisms that is necessary to run Pantavisor in its minimal form. You can take a look at the supported boards here. Contact us if you need support for your board!

In regards of how the interaction with the bootloader is done, Pantavisor can be set up to different modes using the `PV_BOOTLOADER_TYPE` config key:

- uboot
- uboot-ab
- rpiab
- grub

uboot

In this mode, both Pantavisor and U-Boot will write and read to and from a file in storage that contains the revision information that U-Boot needs so it can boot up at any time. With that information, Kernel and initrd will be loaded from that same storage by U-Boot.

uboot-ab

This mode is similar to uboot but it will use fit images stored in two mtd partitions so the Kernel can be loaded in a faster manner by the bootloader. The selection of the partition to be booted is done by Pantavisor using U-Boot env variables, as well as the writing of the Kernel images in each partition.

This mode can be tuned up with the following config keys:

- `PV_BOOTLOADER_UBOOTAB_A_NAME`
- `PV_BOOTLOADER_UBOOTAB_B_NAME`
- `PV_BOOTLOADER_UBOOTAB_ENV_NAME`
- `PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME`
- `PV_BOOTLOADER_UBOOTAB_ENV_OFFSET`
- `PV_BOOTLOADER_UBOOTAB_ENV_SIZE`

rpiab

This mode sets Pantavisor up to work with tryboot_a-b for Raspberry Pi.

grub

In this mode, both Pantavisor and GRUB will write and read to and from a file in storage that contains the revision information that GRUB needs to boot up at any moment. With that information, Kernel and initrd will be loaded from that same storage by GRUB.

6.4 Containers

Pantavisor implements a lightweight container run-time with the help of Linux Containers (LXC). Each container, in its minimal form, is then comprised of a rootfs that will be run isolated in its own name-space and an LXC configuration file. All these, as well as more advanced configuration, are included in the state JSON.

6.4.1 Storage

Rootfs

The most basic storage unit of any container is always the rootfs.

Volumes

In addition to the rootfs, a container may define more auxiliary storage volumes. Pantavisor gives flexibility to configure the persistence of changes and encryption.

There a three types of persistence options:

- permanent: changes are stored in a single writable location.
- revision: changes are stored in a writable location pegged to the revision.
- boot: a pure tmpfs storage that will throw away changes after a reset.

Storage can be linked to a storage disk.

Remount Policies

After mounting the rootfs and additional volumes, Pantavisor can perform one to many remount operations. These operations can be defined either at a global level or per container.

Several policies can be defined, but only one will be run, depending on the `PV_REMOUNT_POLICY` key set in configuration.

Each policy contains a list of remount directives that follows the mount command format for path (can use wildcards) and mount options.

Configuration Overlay

On top of that, additional files can be attached to a revision to create a new overlay that will overwrite whatever is in that location in the rootfs of the container, creating the directories or files if necessary.

Thanks to this, configuration files or scripts can be added or modified without having to do it in the rootfs itself when preparing a new revision. It is advisable to limit this fature to small text based files, as bigger files might make boot up slower.

Underneath the cover, these config overlay files will be attached using the multiple-lower-dir feature of Linux 'overlayfs'. The mount command that sets up this multi lowerdir overlay mount for the pantavisor rootfs is akin to:

```
mount -t overlay overlay -olowerdir=/configs/container:/volumes/container/root.squashfs,upperdir=/volumes/container/lxc-overlay /path/to/rootfs/mountpoint
```

Exports

Container rootfs directories can be mounted back into the host using the `exports` feature from a container. Those will be mounted to `/exports/<container>` at a relative path equal to its absolute path in the container.

This opens the possibility to share directories between containers and Pantavisor itself.

6.4.2 Groups

Containers can be grouped.

Groups main function is to define the order in which containers are started. Groups are ordered and will not begin the mount and/or start up of their containers until all status goals from all the containers belonging to the previous group are achieved. The status goal of each container can be configured at group level as well as overloaded for each container. If not configured at container level, group also determines the restart policy in a similar way as the status goal.

If groups are not explicitly configured, Pantavisor will create the default ones:

name	default status goal	default restart policy	description
data	MOUNTED	system	containers which volumes we want to mount but not to be started
root	STARTED	system	container or containers that are in charge of setting network connectivity up for the board
platform	STARTED	system	middleware and utility containers
app	STARTED	container	application level containers

When using the default groups and if a container is not linked to a group, it will be automatically set to *platform*, except if it is the first container in alphabetical order and no other container has been set to *root*, in which case it will be set to *root*. If not using the default groups and if a container is not linked to a group, the revision will fail.

6.4.3 Roles

Roles can be set to a given container. Roles will determine the elements that Pantavisor will make available in the container rootfs. There is just two role supported for now: `mgmt` and `nobody`.

If no role or the role `nobody` is defined, Pantavisor just mounts these elements into the container under the /pantavisor path:

- pv-ctrl socket with no privileges (only allows to report signals to alter the status goal).
- pv-ctrl-log socket to send logs at Log Server.
- pv-fd-log socket to subscribe file descriptor at Log Server.
- The stored logs for that container and revision.
- The stored user metadata and device metadata for that container.

In addition to this, `mgmt` containers get these elements in /pantavisor:

- pv-ctrl socket with privileges (full request support) for local control.
- pv-ctrl-log socket to send logs at Log Server.
- pv-fd-log socket to subscribe file descriptor at Log Server.
- Full stored logs for all containers and revisions.
- The stored user metadata and device metadata for all containers.
- Challenge and device-id information for Pantacor Hub.

6.4.4 Restart Policy

Restart policy defines how Pantavisor is going to transition into a new revision. There are two types of policies:

- system: any update that modifies any object or JSON belonging to at least one of the containers with *system* restart policy will result in a reboot transition.
- container: any update that only modifies objects or JSONs belonging to containers with the *container* restart policy will result in a non-reboot transition

If the restart policy is not explicitly configured in a container, it will be set according to its group default one.

6.4.5 Status

After a new revision is updated, or after the board is booted up, the containers will try to start if they were not previously started (this could happen in case of a non-reboot update).

These are the different status containers can be at:

- INSTALLED: the container is installed and ready to go.
- MOUNTED: the container volumes are mounted, but not yet started.
- BLOCKED: any of the status goals from a container belonging to the previous group are not yet achieved.
- STARTING: container is starting.
- STARTED: container PID is running.
- READY: Pantavisor has received a readiness signal from the container.
- RECOVERING: container crashed and auto-recovery is waiting to restart it.
- STOPPING: container is stopping because of a update transition.
- STOPPED: container has stopped.

This status is also stored at the group level. The status of a group is always READY, except if any of the containers that form the group has not yet achieved their status goal. In that case, the status of a group is the same as the container with the lower status, not counting the containers that have reached its status goal. This group status can be consulted from our local control interface and is also registered at the Pantavisor logs.

Same way as with the group status, a revision global status is also stored. The way to calculate this status is the same as with the group one, but taking all containers from the revision into account. The revision status is stored in device metadata, can be consulted from our local control interface and changes are registered at the Pantavisor logs.

Status Goal

Status goal defines the status that Pantavisor is going to aim for a container and, ultimately, this is going to affect how groups are activated.

These are the status goals currently supported:

- MOUNTED: for containers whose volumes we want to be mounted but not started.
- STARTED: rest of containers that we want mounted and started, but we only check if its PID is running.
- READY: same as STARTED, but a readiness signal coming from the container namespace is required.

If the status goal is not explicitely configured in a container, it will be set according to its group default one.

A timeout can be configured so an update will fail if the status goal is not achieved withing the defined time value. If the timeout occurs during a regular bootup, the status goal checking will be omited and the following group will be unlocked.

Signals

Signals can be sent from the container namespace to Pantavisor using the local control interface in order to affect the container status.

For now, we only support the `ready` signal, which can be used to get to the READY status goal from a container.

6.4.6 Auto-Recovery

Containers can be configured to automatically restart after a crash using the `auto_recovery` object in run.json or inherited from the container's group.

Recovery Policies

Policy	Behavior
`no`	Never restart (default).
`on-failure`	Restart on exit. (Note: the current implementation does not distinguish exit codes — it behaves the same as `always`. A future revision will leave the container stopped if it exits with status 0.)
`always`	Restart on any exit.
`unless-stopped`	Restart on any exit unless the container was explicitly stopped via API.

Exponential Backoff

When a container crashes, Pantavisor waits `retry_delay` seconds before restarting. On each subsequent crash, the delay is multiplied by `backoff_factor` (e.g., 5s, 10s, 20s, 40s with factor 2.0). The `reset_window` resets the retry counter if the container has been running longer than the configured seconds since its last start.

Stability Tracking

The `stable_timeout` field defines how many seconds a container must survive after reaching its status goal before being considered stable. This does **not** block group startup chaining — groups still gate on `status_goal` only. However, during TESTING, the commit is held until all containers with a `stable_timeout` have proven stable. If a container crashes within its stability window, the timer resets on the next successful start.

Backoff Policy

The `backoff_policy` field controls what happens after `max_retries` is exhausted:

Value	Behavior
`reboot`	Reboot the system (default).
`never`	Leave the container stopped; system continues running.
Duration (e.g., `10min`)	Wait the specified duration, reset the retry counter, and restart the full recovery cycle. Supported units: `s` (seconds), `min` (minutes), `h` (hours).

During TESTING, `max_retries` exhaustion always triggers a rollback regardless of the backoff policy.

Group-Level Defaults

Groups in device.json can define a default `auto_recovery` object. Containers inherit this configuration **all-or-nothing**: if a container has its own `auto_recovery` in `run.json`, it is used entirely; otherwise the group's default applies. No field-level merging is performed. The default `app` group ships with an on-failure recovery policy.

6.4.7 Lifecycle Control

Containers with restart_policy: "container" can be stopped, started, and restarted at runtime via the control socket. Containers with restart_policy: "system" cannot be controlled this way — they require a system-level transition (reboot or update).

Stop

Stopping a container via the API is fundamentally different from a crash. It sets the `user_stopped` flag on the container's auto-recovery state, which tells the recovery engine to leave the container stopped. No retry counters are consumed and no backoff policy is triggered — even for containers with `max_retries: 0` that would normally go to backoff on first crash.

When a container transitions to STOPPED, its volumes are unmounted. This ensures clean state for a subsequent start.

Start

Starting a previously stopped container clears the `user_stopped` flag and transitions the container to INSTALLED, which triggers the normal engine lifecycle: volumes are mounted, drivers are loaded, and the container process is started. Auto-recovery is fully restored with its original configuration.

Restart

Restart force-stops the container and resets the auto-recovery retry counter to zero. For containers with auto-recovery configured, the recovery engine picks up the stopped container and restarts it through the standard recovery path (respecting retry delays and backoff). For containers without auto-recovery, the restart transitions the container directly to INSTALLED for an immediate start.

6.4.8 Drivers

Containers can reference the BSP managed drivers as required, optional or manual.

- required: these drivers will be loaded as soon as container is STARTED. The revision will fail if the drivers are not enabled through BSP as managed drivers.
- optional: these drivers will be loaded as soon as container is STARTED too. In this case the revision will not fail if the drivers are not defined in the BSP.
- manual: drivers can be loaded from within containers trough local control. The success or failure of loading drivers using the REST API will not determine whether a revision fails or not, but the calls will return an error response if necessary.

6.4.9 Loggers

Containers, by default, will automatically direct these logs from the container to the Log Server:

- syslog
- messages
- lxc log
- lxc console

This list can be expanded to other files using the state JSON.

6.4.10 Service Mesh

Containers can expose services to each other and consume services from other containers through xconnect, Pantavisor's built-in service mesh. Providers declare services in a `services.json` manifest; consumers declare requirements in their `run.json`. The `pv-xconnect` daemon handles mediation and resource injection at runtime, supporting Unix sockets, REST, D-Bus, DRM, and Wayland.

6.5 Updates

Getting back to revisions, these can be modified and upgraded, thus creating a new revision of the device.

6.5.1 Flow

Updates will be processed differently depending on the source being remote or local.

Remote

The diagram shows a high level concept of how remote updates work. In this case, we have a periodic routine that checks for pending updates upstream and executes the installation and transition to them one by one:

Local

In this diagram, you can see how local updates work. In the case of this kind of update, we have a one-shot process triggered by a run revision command:

6.5.2 Progress

Pantavisor keeps the update progress information for each revision in storage.

This info will also be reported to the cloud in case of the device being remote and communication available at that moment.

Pantavisor will only progress to the new revision in case of success. Otherwise, no action or rollback will ensue, depending on the point where the error is detected. This is the list of possible update states:

- NEW
- SYNCING
- QUEUED
- DOWNLOADING
- INPROGRESS
- TESTING
- UPDATED
- DONE
- WONTGO
- ERROR
- CANCELED

NEW

Initial state for remote updates. Set by the cloud side.

SYNCING

Device is syncing its first revision with the cloud. Set by the cloud side.

QUEUED

Only valid for remote updates.

Pantavisor has got the state JSON of the new revision, but is performing other operations and has put it to the queue to be processed later.

Messages
Retried X of Y

DOWNLOADING

Only valid for remote updates.

Downloading the artifacts for the new revision.

Messages
Retry X of Y

INPROGRESS

Installing or progressing to this revision. Transitions to new revisions can either require a reboot or not.

To finish this state, it is necessary that all status goals existing in the new revision have been achieved. Also, in the case of a remote update, Pantavisor needs to have performed communication with Pantacor Hub. If these two conditions are not met within a configurable time, Pantavisor will rollback the revision.

Messages
Update objects downloaded
Update applied
Update installed
Starting updated version
Transitioning to new revision without rebooting
Rebooting

REBOOT TRANSITION

Reboot transitions are performed based on the location of the changes belonging to the new revision update:

- In the root of the status JSON
- In the BSP
- In any of the containers with a *system* restart policy
- In any additional file that belongs to a container with *system* restart policy
- In any additional file that does not belong to any container

In this case, Pantavisor will stop all the containers and reboot the board.

NON-REBOOT TRANSITION

Non-reboot transitions are performed after an update that does not contain any changes in any of the components described for the reboot updates.

In this case, Pantavisor will only stop the containers that were affected by the update and restart them with the recently installed new revision artifacts.

TESTING

Waiting to see if the revision is stable. During this stage, Pantavisor checks if all containers are running and will rollback if any of them exits. Besides that, in the case of a remote update, it will also rollback in case Pantacor Hub communication is lost.

If any container has a stable_timeout, the commit is held even after the commit delay timer expires, until all containers have survived their stability window. If a container with auto-recovery exhausts its `max_retries` during TESTING, a rollback is triggered immediately regardless of the configured `backoff_policy`.

Messages
Awaiting to set rollback point if update is stable
Awaiting to see if update is stable
Waiting for all containers to become stable

UPDATED

The revision is stable, but the update did not need a board reboot, so the rollback point is not set until you force a reboot.

Messages
Update finished, revision not set as rollback point

DONE

The revision has been fully booted and is stable. The rollback point is set.

Messages
Update finished, revision set as rollback point
Factory revision

WONTGO

The new revision cannot be installed because of a bad state JSON, so it is aborted before getting into INPROGRESS or TESTING.

Messages	Possible causes
Update aborted	Local update cancelled by a command
Max download retries reached	The maximum processing or download retry number was reached for a remote update
Space required X B, available Y B	Not enough space in disk for remote update
Internal error	Memory allocation error or code bug
State not fully covered by signatures	The revision has some element that is not signed
Signature validation failed	Any of the revision signatures is not up to date with its covered content
Unknown error	Signature failed without a known cause
State JSON has bad format	State JSON could not be parsed

ERROR

The new revision failed during INPROGRESS or TESTING stages. Pantavisor will try to rollback to the latest DONE revision.

Messages	Possible causes
Internal error	Memory allocation error or code bug
State not fully covered by signatures	The revision has some element that is not signed
Signature validation failed	Any of the revision signatures is not up to date with its covered content
Unknown error	Signature failed without a known cause
Object validation went wrong	Stored object used by the revision changed or went corrupt
Hub not reachable	Connection with hub could not be achieved after a remote update
Hub communication not stable	Connection with hub was established but then failed during TESTING
Stale revision	An revision was found in hub with an index that is less than the current running revision one
Status goal not reached	Status goal of a container could not be reached before the end of TESTING
A container could not be started	A container failed during LXC start up
Unexpected rollback	Crash or power cycle before having the chance to report any meaningful status

CANCELLED

Only applicable on remote updates. The revision has been marked as cancelled by the cloud side.

6.6 Storage

Pantavisor needs some persistent artifacts on disk to perform several operations such as storing revisions, updates, disks, logs and other RW operations.

These artifacts are stored in the `PVROOT` partition, which is mounted to `/storage` by Pantavisor during initialization.

Its structure is as follows:

```
.
    boot                        # bootloader files
        boot.scr
        uboot.txt
    cache                       # metadata
        devmeta
        dropbear
        usrmeta
    config                      # Pantacor Hub RW configuration
        pantahub.config
    disks                       # persistent container and Pantavisor storage
    logs                        # container and Pantavisor logs are centralized here
    objects                     # general revision artifacts
        1dc4107c036a14aac38b5218fe50a8c5fa9e73d92a1d24337a57cd833c0a8797
        2c0236e022c8bed4af695a96504ee7ece2b43e0d57f5a4d4181edb028539f061
        8debde333d3885abf03cb7f954017744c8e2fc6708f89f0a4771b0bf367fd2a6
        97bf54227c91cd93b540785942e4884ac6a7352fcea84afc3e0b8d174c7e4825
        ac1a09142f25a8df6742a3da290008f99c05c7b83d2754d9928f1a081714eaed
        b236948bd3caaf31c00af82fbeeb0816962e39a96ad6c5cf9bfc1a547cdbe005
        b6959af6ef61ae8ff729a9e5e41a8fdf5f318bd6ef1d21d38dc796d0db7e7627
        f4b9d24c631cd1bcc303cedd3b9b3c6d32556b1975ac4262e823bf5d9affbc9f
    trails                      # state JSON of each revision with objects hardlinks
        0                         # Pantavisor checkout for revision 0
            _sigs
                bsp.json            # .json files are shipped inline, no hardlink
            bsp
                drivers.json
                firmware.squashfs    # this is a hardlink to an object
                modules_6.1.77+.squashfs
                modules_6.1.77-v7+.squashfs
                modules_6.1.77-v7l+.squashfs
                modules_6.1.77-v8+.squashfs
                modules_6.1.77-v8-16k+.squashfs
                pantavisor-rpi.img.gz
                run.json
                trail.config
            device.json
            .pvr/json               # full revision state JSON
            <container-1-name>
            <container-2-name>
            ...
...
```

- boot
- cache
- config
- disks
- logs
- objects
- trails

6.6.1 Boot

Contains revision number information written by Pantavisor and consumed by the bootloader so it can run the proper revision after powering off and rebooting.

6.6.2 Cache

Mainly to store information that is not fixed to any given revision.

Metadata

Metadata is the information that is exchanged between Pantavisor and the outside world, either remotely or locally, that is not attached to a revision itself.

Metadata is just a list of key-value pairs of string type. This list is persistent and stored on disk. By default, it is saved in plain text, but this can be changed in favour of volumes on a crypt disk.

DEVICE METADATA

This type of metadata is meant to be generated by Pantavisor or any of the running containers to expose internal information.

The list of device metadata is persistent on disk and can be managed locally by mgmt containers. Their values will be uploaded to the cloud in remote mode.

Check the list of default key-value pairs that Pantavisor sends.

USER METADATA

This type of metadata is meant to be generated by the user so Pantavisor or any of the containers can consume it.

The list of user metadata is persistent on disk, can be managed locally by mgmt containers. Their values will be deleted or overwritten by the list stored in the cloud in remote mode.

Check out the list of default user metadata consumed by Pantavisor.

6.6.3 Config

Stores the `PH_` prefixed keys of the configuration. This includes the Pantacor Hub client credentials.

These configuration keys are persistent. By default, they are saved in plain text, but this can be changed if set to a volume assigned to a crypt disk.

6.6.4 Disks

Pantavisor offers a way to define the physical storage medium by setting up disks and disk v2.

There are currently 4 disks types supported for disk:

- Non-encrypted directory
- Device Mapper crypt using without hardware acceleration (versatile)
- Device Mapper crypt using i.Mx CAAM
- Device Mapper crypt using i.Mx DCP

The new disk v2 supports all the previous ones and adds 2 new types:

- Swap disk based on zram
- Volume disk based on zram

Each disk, with the exception of swap disk, that is defined in the state JSON can then be privately used by the containers. They can also be internally used by Pantavisor volumes, as in the case of metadata.

Volumes

Additional volumes can be specified to be used internally by Pantavisor. These are some special cases that can be configured as volumes:

- pv--devmeta: if defined, Pantavisor will use this volume to save and load device metadata.
- pv--usrmeta: if defined, Pantavisor will use this volume to save and load user metadata.
- pv--phconfig: when defined, Pantavisor will reserve this volume to save and load Pantacor Hub credentials. To make Pantavisor use it, it has to be forced from configuration with the key `PV_STORAGE_PHCONFIG_VOL`.

6.6.5 Logs

Pantavisor can centralize all of your container logs, separated by revision, in one place on-disk. It does so by running a small server (Log Server) which offers a couple of sockets to the containers:

Output types

- pv-ctrl-log: to send log traces.
- pv-fd-log: to suscribe file descriptors.

There are a number of parameters that can be tweaked from the configuration that will affect Log Server, such as `log.capture`, `log.maxsize`, `log.level`, etc. Between those, there is an important parameter called `log.server.outputs` that allows to change how the logs will be stored:

- filetree
- singlefile
- stdout
- stdout.pantavisor
- stdout.containers
- stdout_direct
- nullsink

These options are not mutually exclusive and can be combined in any fashion.

FILE TREE

This is the default option. In this case the log output will be delivered in different files by containers. Pantavisor logs will also be stored in its own directory.

SINGLE FILE

With this option, all logs will be centralized in one file. These logs are delivered in a JSON following this structure:

```
{
    "tsec":9,
    "tnano":0,
    "plat":"pantavisor",
    "lvl":"DEBUG",
    "src":"logserver",
    "msg": "starting logserver loop"
}
```

STANDARD OUTPUT

Outputs logs into stdout, which can be useful for debugging. Unlike stdout_direct, this sink is processed by the log server. This will provide more control on the output than its direct counterpart and a way of working that is more similar to the other sinks.

> **Note**
>
> In embedded mode, this is the same as sending the logs to dmesg. You can set ignore_loglevel and printk.devkmsg=on in cmdline to get all messages on console.

STANDARD OUTPUT PANTAVISOR

Same as Standard Ouput but filtering all messages except Pantavisor ones.

STANDARD OUTPUT CONTAINERS

Same as Standard Ouput but filtering all messages except container ones.

STANDARD OUTPUT DIRECT

Outputs logs into stdout. Unlinke stdout, this sink is not processed by the log server. This means the logs will be directly printed without going through the server. It will have a faster response and will provide logs during device initialization and teardown, when plain stdout would not be available.

This sink will be automatically enabled when stdout is enabled and log server is not available. That is, before its initialization and after its shutdown. Furthermore, it will always let pass all FATAL level log messages through.

NULL SINK

Send logs to /dev/null.

Timestamp format

All the log outputs, except for the NULL Sink, will print the timestamp along the log lines. It is possible to configure the format of the timestamp with the configuration key `log.<output>.timestamp.format`, which can be set with the following values:

value	example
golang:Layout	"01/02 03:04:05PM '06 -0700"
golang:RubyDate	"Mon Jan 02 15:04:05 -0700 2006"
golang:ANSIC	"Mon Jan _2 15:04:05 2006"
golang:RFC822Z	"02 Jan 06 15:04 -0700"
golang:RFC1123Z	"Mon, 02 Jan 2006 15:04:05 -0700"

Those formats are based in the Golang time constants. For other formats, the strftime formatters can be used setting the `strftime:` prefix. For example: `strftime:%d, %T %Y`. Please check the strftime manual for more information about formatters.

Log Directory Size Management

Pantavisor manages log storage at two levels:

1. **Per-file rotation** — individual log files are compressed in place once they reach the per-file rotation threshold (derived from `PV_LOG_DIR_MAXSIZE` and `PV_LOG_ROTATE_FACTOR`).
2. **Directory-level management** — the total size of the log directory is kept within a configurable budget using a watermark-based cleanup algorithm.

AUTO-SIZING (ZERO-CONFIG)

When `PV_LOG_DIR_MAXSIZE` is set to `0` (the default), Pantavisor automatically determines the log budget at startup by inspecting `/proc/mounts` to identify the filesystem backing the log directory:

- **tmpfs**: budget = 100% of the partition size (the log partition is in RAM, so all of it can be used for logs)
- **Block device**: budget = 10% of the partition size (conservative default to share storage with other data)

Partition size is read from `/sys/class/block/<dev>/size` (kernel sectors × 512 B) with a `statvfs()` fallback.

MANUAL SIZING

Set `PV_LOG_DIR_MAXSIZE` to an explicit value using one of these formats:

Format	Example	Meaning
Plain integer (bytes)	`16777216`	16 MiB
With unit suffix	`256MB`, `1GB`, `512K`	Absolute size
Percentage	`20%`	20% of the underlying partition

Values exceeding the available partition capacity are rejected and fall back to the auto-calculated default.

WATERMARK-BASED CLEANUP

Pantavisor tracks the total size of the log directory and applies a high/low watermark algorithm to keep disk usage in check without constant thrashing:

```
hysteresis_gap = PV_LOG_DIR_MAXSIZE / PV_LOG_HYSTERESIS_FACTOR
high_watermark = PV_LOG_DIR_MAXSIZE × 0.90
low_watermark  = PV_LOG_DIR_MAXSIZE - hysteresis_gap
rotation_size  = hysteresis_gap / PV_LOG_ROTATE_FACTOR
```

How cleanup works:

1. Every 100 log messages the current log directory size is recalculated.
2. If `current_size > high_watermark`, deletion is triggered.
3. The oldest compressed log files are deleted from the largest subdirectories first.
4. Deletion continues until `current_size < low_watermark`.

Example with defaults and an auto-calculated budget of 512 MB:

```
hysteresis_gap = 512 MB / 4 = 128 MB
high_watermark = 512 MB × 0.90   460 MB     cleanup starts here
low_watermark  = 512 MB - 128 MB = 384 MB    cleanup stops here
rotation_size  = 128 MB / 5   25 MB         per-file size before rotation
```

CONFIGURATION REFERENCE

Key	Default	Purpose
`PV_LOG_DIR_MAXSIZE`	`0` (auto)	Total log directory budget
`PV_LOG_HYSTERESIS_FACTOR`	`4`	Controls the gap between high and low watermarks
`PV_LOG_ROTATE_FACTOR`	`5`	Per-file rotation size divisor

6.6.6 Trails and objects

Pantavisor trails, with their state JSONs are stored for each installed revision. The rest of artifacts that can be shared between revisions, are called objects and are stored separately so objects can belong to different revisions at the same time.

Update progress

A progress JSON is stored with update info relevant to each revision. This JSON contains persistent information useful for remote experience reporting but also for local experience debugging of failed updates:


```
{
    "status":"ERROR",
    "status-msg":"Status goal not reached",
    "progress":100,
    "retries":0,
    "logs":
        "{\"tsec\":14457,\"tnano\":0,\"platform\":\"pantavisor\",\"lvl\":\"ERROR\",\"src\":\"platforms\",\"msg\":\"platform 'pvr-sdk_ready' status goal timed
out after waiting for more than 30 secs\"}\n
        {\"tsec\":14457,\"tnano\":0,\"platform\":\"pantavisor\",\"lvl\":\"ERROR\",\"src\":\"controller\",\"msg\":\"timed out before all goals were met. Rolling
back...\"}\n"
}
```


Update progress can be consulted locally using the ctrl steps endpoint.

Integrity

Pantavisor offers **secureboot**, a security mechanism to ensure integrity of the artifacts that are part of a revision. As revisions are composed of a state JSON and a series of artifacts pointed by that JSON, Pantavisor can perform a double level validation: artifact checksum and state signature.

ARTIFACT CHECKSUM

Objects belonging to the revision that is about to be run after boot up are checked against the checksums stored in their state JSON by default.

As running a full SHA hashsum of all artifacts can be time consuming for low spec targets and thus increase boot up time, a lazy integrity validation on container storage can be performed using handlers, which are specified at the state JSON. Handlers are scripts to mount/unmount volumes. The only supported handler so far is dm-verity.

Checksum can also be totally disabled at the configuration level if boot up speed needs to be boosted further, with the `PV_SECUREBOOT_CHECKSUM` key.

STATE SIGNATURE

Additionally to artifact checksum, the state JSON can be signed. This can be done from your host computer or from an automated CI, and it is then validated from Pantavisor using a cryptographic hash algorithm. It can be tuned according to one of these three levels of severity:

- `disabled` : no signature validation will be performed.
- `audit` : all validations from lenient and strict will be performed, errors will be reflected in log but nothing will fail.
- `lenient` : only the signed artifacts in the revision will be validated. Enabled by default.
- `strict` : all artifacts in the revision must be signed and will be validated.

Validation is performed in Pantavisor at two points:

- when booting up, right after Pantavisor is initialized and before the current revision is loaded. In case of failure, the update will be reported as ERROR if possible and the device will either rollback or just reset.
- when a new remote update is received or a new local update is run. In case of failure, the update will be reported as WONTGO.

In either case, Pantavisor will parse all signatures, parse the protected JSON, calculate the hash depending on the elements included in the JSON and compare them with the signatures.

Right now, RS256, ES256, ES384 and ES512 algorithms are supported and Pantavisor must know the public key either by directly storing it on disk or by using a certificate chain.

On-disk public key

> **Note**
>
> We recommend to use the certificate chain of trust for your field devices.

In this option, Pantavisor will need the RSA public key to be stored on disk. You can store it during build time from the ca/ directory.

It will need the signature and information about how to generate it in the revision checkout. We offer pvr sig for convenience.

Certificate chain of trust

In this case, the public key will travel in the revision checkout in a x509 certificate. The certificate will be validated by Pantavisor against a root certificate located on disk .

The root certificate is placed during Pantavisor build. The build system will extract the files from the ca/ directory.

Depending on the subject CN of the signature certificate, we validate it with the following criteria:

- If the subject CN is equal to the value set in `PV_OEM_NAME` from your configuration and the signature does not includes any artifact in the BSP or OEM configuration directories, as those are can only be secured by the default root certificate, we validate it against the OEM root certificate.
- For every other case, the signature is validated against the default root certificate.

It will also need the signature and information about how to generate it in the revision checkout. For this, you can use pvr sig command.

6.6.7 Garbage Collector

The Pantavisor garbage collector is in charge of cleaning up the /storage by automatically removing unused Pantavisor artifacts.

It works by removing logs, trails and stored disks that belong to old revisions. After all this, all orphan objects that were linked to removed revisions are deleted too.

The garbage collector will not affect any of the files related to the running revision, the revision that is being updated or the latest DONE revision. These revisions that are not affected by the garbage collector can be expanded to the factory revision using the PV_STORAGE_GC_KEEP_FACTORY configuration parameter.

Currently, it can be triggered by three different events:

- by a remote update that requires more disk space than available
- if a threshold of disk usage is surpassed
- with a command

Remote update

If a remote update requires more disk space than available, the garbage collector will be activated. This can be adjusted with the configuration PV_STORAGE_GC_RESERVED parameter.

> **Note**
>
> This type of trigger will not work with local updates. For those, you will need to use one of the following two options.

Threshold

Disk usage will be checked periodically and garbage collector will be activated if that threshold is reached. By default it is disabled. This can be changed with the PV_STORAGE_GC_THRESHOLD parameter.

If an object is put using the objects endpoint, the threshold will be disabled temporarily, which is done to avoid removing objects that could be linked to the upcoming new revision. This parameter can be changed from the configuration.

On Demand

Finally, our garbage collector can be triggered on demand issuing a command through Pantavisor control socket.

6.7 Remote Control

This page is for explaining cloud-oriented ways of controlling your Pantavisor devices.

6.7.1 Pantacor Hub

Pantacor Hub is our remote device state management system.

To interact with Pantacor Hub, you can either use the web user interface, the API or the pvr CLI tool.

Pantacor Hub Client

Pantavisor includes a build-in Pantacor Hub client which is enabled by default. To begin remote control and monitoring of devices with Pantacor Hub, it is necessary to create an account and claim the device to it. From that moment on, the device will be attached to that account and will try to keep the connection with Pantacor Hub opened, except if a local revision is installed. In that case, the device will turn to local control until a go remote command is issued. This behavior can be avoided with the remote always configuration.

The main features offered by remote control are:

- Keeping the device up to date: the revisions form a trail of steps in Pantacor Hub. This trail is ordered according to the time of deploying and will be consumed by Pantavisor step by step. After founding a new step, Pantavisor will parse the revision State JSON, download the artifacts into the storage and transition to the new revision.
- Sending device metadata and receiving user metadata.
- Sending logs: sending stored logs up.
- Sending logs: sending stored logs up.

STATE MACHINE

The client state machine goes through the following stages:

> **Note**
>
> All states are independent from the Pantavisor state machine, but all their associated processes are run from its WAIT state.

> **Note**
>
> The client state can be consulted at any moment via device metadata.

- **init**: means we are in remote mode and the client has been initialized.
- **register**: device is trying to register as a non-claimed device in the cloud. This credentials will be stored after reboot.
- **claim**: device is registered and is waiting to be claimed.
- **sync**: device has been claimed and will try to upload the objects and state JSON to the cloud.
- **login**: device is using its credentials to start a session with Hub.
- **wait hub**: device is waiting for Hub to reply. We also use this to decide whether the device is claimed or not.
- **report**: an update finished and device will continue uploading devmeta, downloading usrmeta while reporting its update status.
- **idle**: device has checked all objects and state JSON is in the cloud. This is where the device will spend most of its time, uploading devmeta, downloading usrmeta and waiting for new updates.
- **prep download**: a new update (same as a new state JSON) has been received and the device will now download the object metadata information for each of its objects. This includes object size, sha and download URL.
- **download**: all object metadata is now on memory and we can proceed dowloading all objects that are not already installed in disk. Then we will continue with progressing to the new revision.

6.7.2 Other Remote Controllers

It is also possible to control Pantavisor using any other server. For that, it is necessary to implement a container that performs the communication between the server itself and Pantavisor.

One example of this kind of setup is our Azure IoT Hub client.

6.8 Local Control

With local control, we mean the control that is performed from one of the containers running in the revision using the control socket, which offers an HTTP REST API to control and monitor Pantavisor.

This kind of control can be performed even if the device is already claimed in Pantacor Hub. It can also be the only option if you disable remote control or if you install a local revision at any given time, which will interrupt Pantacor Hub remote control unless a go remote command is issued. This behavior can be avoided with the remote always configuration.

6.8.1 Available Operations

The control socket exposes a REST API for managing the device from within a container. Key operations include:

- List and manage containers — query status, stop/start/restart individual containers with restart_policy "container"
- Send signals — report readiness to Pantavisor
- Issue commands — reboot, run revisions, trigger updates
- Manage metadata — read and write user/device metadata
- Manage steps — install and query revisions
- Manage daemons — start/stop internal Pantavisor daemons
- Query service mesh — inspect the xconnect graph

6.8.2 Pantabox

Pantabox is the top level control tool that can be run inside of a container. It offers a ncurses user interface that lets you interact with Pantavisor (install new revisions, exchange metadata, reboot or shutdown your device...).

It is included in pvr-sdk, our development platform that is included with the initial devices. To get more info or try it out, ssh your pvr-sdk container and just type the command:

```
pantabox
```

Pantabox is built on top of pvcontrol.

6.8.3 pvcontrol

pvcontrol is the CLI control tool that communicates with Pantavisor control socket using cURL. As it is a Pantabox dependency, it generally gets advantage of the latest features of the control socket first.

It is also included in pvr-sdk, included with the initial devices. To try it out, ssh to your pvr-sdk container and just type the command:

```
pvcontrol
```

6.8.4 Other Local Controllers

In the end, Pantabox and pvcontrol are just HTTP clients that are making use of Pantavisor control socket.

If you want to take advantage of the local control in your own container, first make sure mgmt role is selected in your container. Then, consider importing Pantabox and/or pvcontrol into your container. Besides this option, you can always directly use cURL or any other HTTP client to attack the control socket endpoints.

6.9 Inter-Container Communication

Pantavisor includes `pv-xconnect`, a built-in service mesh that manages communication between containers at runtime. It runs as a managed daemon alongside Pantavisor, active in all init modes.

6.9.1 Why xconnect?

In a Pantavisor system, containers are isolated by design. When one container needs to talk to a service in another — a D-Bus system bus, a REST API, a graphics device — this traditionally required manual socket coordination, custom bind mounts, and trusting each container to self-identify correctly.

`pv-xconnect` replaces this with a **mediation layer**: providers declare what services they expose, consumers declare what they need, and Pantavisor injects the correct virtual resources into the consumer's namespace at runtime. Container identity is resolved by Pantavisor itself, not by the container.

6.9.2 How It Works

The service mesh operates through a graph of connections maintained by `pv-xconnect`:

1. **Providers** declare services in a `services.json` file in their container manifest.
2. **Consumers** declare requirements in their `run.json`, including service name, type, and where the resource should be injected inside the container.
3. `pv-xconnect` periodically reads the xconnect-graph from the control socket, resolves provider/consumer pairs, and injects resources into consumer namespaces.

This is entirely declarative: no code changes are needed in containers to expose or consume services.

6.9.3 Supported Service Types

Type	Description
`unix`	Direct Unix socket proxy (supports FD passing and shared memory)
`rest`	HTTP-over-Unix-socket with automatic identity injection (`X-PV-Client`, `X-PV-Role` headers)
`dbus`	D-Bus proxy with role-based identity masquerading using provider-side `/etc/passwd`
`drm`	DRM/KMS device node injection for display servers
`wayland`	Wayland protocol mediation for isolated UI rendering

6.9.4 Security Model

Pantavisor acts as the security broker. Containers use logical service names rather than raw socket paths. Access must be explicitly declared in the revision state JSON. The identity presented to the provider is resolved and injected by `pv-xconnect` from the revision's role configuration, not asserted by the consumer.

6.9.5 Configuration and Control

The xconnect service mesh can be inspected at runtime through the /xconnect-graph endpoint of the Pantavisor control socket. The `pv-xconnect` daemon can be started and stopped via the /daemons endpoint.

For a full reference of service manifest formats and mediation patterns, see the xconnect reference.

6.10 Pantavisor Configuration

> **Note**
>
> The configuration syntax is common for all levels, but not all levels support the same keys. Our reference contains the list of keys and the allowed levels for each one.

There are several ways to set Pantavisor configuration, depending on when it can be modified in the Pantavisor life cycle. Bear in mind that not all configuration parameters will be available for all levels. Furthermore, each level will overwrite whatever is configured in the previous ones, following this order:

1. pantavisor.config
2. pantahub.config
3. Policies
4. OEM
5. cmdline
6. Environment Variables
7. User Metadata
8. Commands

6.10.1 pantavisor.config

Configuration file for Pantavisor. It can only be changed at build time.

6.10.2 pantahub.config

Build time configuration file for Pantavisor built-in Pantacor Hub client.

6.10.3 Policies

Policies are added at build time from the vendor skel directory, but loaded during boot up time.

To select a policy among the installed ones, we need to set its name to the `PV_POLICY` key either from pantavisor.config or environment variables.

6.10.4 OEM

For setups where we want to modify the configuration based on device updates, we offer the possibility to attach a configuration file to a revision.

Its location inside the revision will be defined by the configuration values of the keys `PV_OEM_NAME` and `PV_POLICY` from pantavisor.config, environment variables or policies levels.

6.10.5 cmdline

> **Warning**
>
> This method is *DEPRECATED* but still supported for backwards compatibility reasons. It is recommended to use env variables instead.

Right after loading the configuration files, Pantavisor reads `/proc/cmdline` in search for `key=value` pairs that use the prefix `ph_` or `pv_`. This can be done from the bootloader console.

6.10.6 Environment Variables

Linux environment variables can be used to configure Pantavisor. To do that, the rules to set env variables have to be followed:

- Use `key=value` format
- Do not use `.`
- If ` characted is needed, you can escape them by using `"` between the config item. For example: `"PV_SYSCTL_KERNEL_CORE_PATTERN=|/lib/pv/pvcrash --skip"`.

These variables need to be set during boot time, and setting them after that will have no effect on Pantavisor. This can be achieved from the bootloader console.

6.10.7 User Metadata

> **Note**
>
> If the user metadata volume is assigned to a permanent volume, as it is by default, these changes will persist over device reboots.

User metadata can be used to override any of the previously presented configuration mechanisms.

There is a number of ways of setting user metadata, depending on the device management method choice. Go to our how-to use Pantavisor guide for more information.

6.10.8 Commands

> **Note**
>
> It is important to notice that these changes will not persist after a device reboot in any case.

The Pantavisor control socket, or consequently the PVControl tool, offers another way to change a very limited subset of configuration values. Specifically, using the command endpoint.

6.11 Init Mode

Pantavisor offers several configurable init modes for convenience: embedded, standalone and appengine.

6.11.1 Embedded Mode

This is the way Pantavisor was meant to be run. In this case, the bootloader will directly start up Pantavisor, which will run alongside a minimal rootfs with all its dependencies.

6.11.2 Standalone Mode

This mode was created for debugging Pantavisor. It works the same way as *embedded*, with the bootloader starting Pantavisor up. The difference is Pantavisor will not launch any container or perform any of its regular operations. To do so, Pantavisor has to be manually run from console inside of the device.

6.11.3 App Engine mode

App Engine mode is meant for prototyping on already setup devices running any Linux distro. In this case, Pantavisor will run as a daemon that can be started from your init system or directly from console.

> **Note**
>
> You can get more information about how to run Pantavisor in our how-to guides.

6.12 Watchdog

Pantavisor also offers some configurable convenience to set up and ping the Linux Kernel watchdog.

6.12.1 Mode

The watchdog mode will determine when the watchdog is pinged by Pantavisor.

Disabled

In this mode, Pantavisor will never ping the watchdog.

Shutdown

This is the default mode. Pantavisor will start pinging the watchdog when a reboot or poweroff order is issued. This way, we make sure the device reboots in a shutdown freeze event.

Startup

This mode combines Pantavisor pinging the watchdog during initialization and shutdown. As the watchdog is activated with a ping, it is expected that a container will take that role during the device regular operation.

Always

In this mode, Pantavisor will ping the watchdog during all the device operation.

7. Reference Pages

7.1 Pantavisor

7.1.1 Legacy

Build Components

Build components at compile time.

BUILD INDIVIDUAL COMPONENTS

You can build individual components of the system by calling the module directly as secondary target.

This builds the Pantavisor binary and all its dependencies:

```
./build.docker.sh malta-qemu init
```

This builds the pv_lxc container runtime plugin for Pantavisor:

```
./build.docker.sh malta-qemu pv_lxc
```

This assembles the initrd image '0base.cpio.xz' from all build assets:

```
./build.docker.sh malta-qemu image
```

You can also clean individual components by appending `-clean` to the build subtarget:

```
./build.docker.sh malta-qemu init-clean
```

This runs the 'trail' assembly step, which results in the final PV-trail for storage:

```
./build.docker.sh malta-qemu trail
```

MODULE MANAGEMENT

Modules can be added or removed just by adding a new atom.mk file inside the build tree and modifying the alchemy module configuration for a given target.

To check if the current configuration corresponds to the state of the tree:

```
./build.docker.sh malta-qemu config-check
```

To actually make changes on the target configuration, efectively adding or removing modules:

```
./build.docker.sh malta-qemu config-update
```

CODE CHECK

To run the clang format check on Pantavisor source code, just append the `-codecheck` suffix to the init subtarget:

```
./build.docker.sh malta-qemu init-codecheck
```

Build Options

Build options reference page for build.docker.sh.

Key	Value	Default	Description
PVR_MERGE_SRC	Clone URL or cloned device path with */.pvr* suffix	empty	Reference device. All containers from that device will be used for the new image
PANTAVISOR_DEBUG	*yes* or *no*	*no*	Include debug tools in Pantavisor binary
MAKEFLAGS	build options	empty	Additional build options to get a more verbose build log. For example: "V=s -j1"
PV_FACTORY_AUTOTOK	autotok	disabled	Set new image to be automatically claimed
PV_BUILD_INTERACIVE	*true* or *false*	*true*	Mean to disable interactive builds for automation
PVR_USE_SRC_BSP	*yes* or *no*	*no*	Avoid building the BSP and use the one from *PVR_MERGE_SRC* in the final image
PV_PVS_PUB_PEM	path to public key	empty	Copy public key to the initrd rootfs
PV_PVS_CERT_PEM	path to CA certificate	empty	Copy CA certificate to the initrd rootfs
PV_BUILD_PANTAVISOR_BUILDER_PREFIX	pantavisor-builder prefix	registry.gitlab.com/pantacor/ci	Use a different builder prefix for pantavisor-builder. Used for docker mirrors
PV_BUILD_PANTAVISOR_BUILDER_VERSION	pantavisor-builder version	001	Use a different version for pantavisor-builder. Commits, tags and branches can be used
PV_BUILD_TLS_USE_HOST_CACERTS	*default* or path to CA certificate	empty	Mount CA certificate to pantavisor-builder container. Setting to default will use /etc/ssl/certs/ca-certificates.crt
PV_MERGE_SBOM	Path to folder that contains SPDX JSON formated SBOM to merge	empty	Allows extracting packages from several SBOM documents (SPDX in JSON format) and merge it with Pantavisor SBOM

Customize your Build

Once you have synced the manifest, you have at your disposal all the source code that you need to build Pantavisor:

```
alchemy  bootloader  build.docker.sh  build.sh  ca  config
external  firmware  internal  kernel  prebuilt  run.docker.sh
scripts  test.docker.sh  tests  vendor
```

There are some recurrent locations you might need to play with in order to customize Pantavisor to fit your needs. If you want to create your own customized build, forking these repositories will be something you might need at some point:

- ca
- config
- vendor

CA/

In this directory, we store the secureboot certificates and keys. Some of files are:

- `ca/pvs/pvs.defaultkeys.tar.gz` : contains the default root certificate.
- `ca/pvs/pvs.oemkeys.tar.gz` : contains the OEM root certificate.

You can check our `ca/` repository for our public devices.

CONFIG/

This directory is meant for various platform specific configuration files.

- `config/<platform>/pantavisor.config` : config file for the `PV_` keys.
- `config/<platform>/pantahub.config` : config file for the `PH_` keys.

You can check our `config/` repository for our public devices.

VENDOR/

This directory is meant for various platform specific vendor files.

- `vendor/<platform>/skel/` : this directory contains files that will be added to the Pantavisor rootfs.
- `vendor/<platform>/stepskel/` : this directory contains files that will be added to revision 0.

You can check our `vendor/` repository for our public devices.

Pantavisor Log sockets

The Pantavisor logging system works with 2 sockets `pv-ctrl-log`, for receive logs from any application who needs to send logs to Pantavisor log system, and `pv-fd-log` used by Pantavisor to subscribe file descriptors (fd) to be reading and add to the log system as appropriate.

PV-CTRL-LOG

All messages are sent to the `pv-ctrl-log` socket inside a structure called logserver_msg, defined as follows:

```
struct logserver_msg {
    int version;
    int len;
    char buffer[0];
};
```

The fields in the structure are:

- version: always 0 (no other version exists currently)
- len: length of the buffer
- buffer: log data. This data is organized in a simple way using 0 as separator:

```
level\0platform\0source\0data\0
```

Where:

- level: log level for the message, a numeric value. See below.
- platform: container name
- source: specific place where the log belongs (e.g: function name, module, etc)
- data: message to log

The log levels are defined using an `enum`:

```
enum log_level {
    FATAL, // 0
    ERROR, // 1
    WARN,  // 2
    INFO,  // 3
    DEBUG, // 4
    ALL    // 5
};
```

PV-FD-LOG

Pantavisor provides a Unix socket to add and remove a fd. Those fd will be polling and if there are data available it will be added to the system.

Subscribe

For the subscription mechanism `sendmsg` system call must be used. This function is present on several languages like C and C++, Go, Rust and Python, among others.

As part of the protocol, the platform (container name) and the source of the fd (e.g. console.log, syslog, etc.) must be specified in the header of the message, using the `iov structure`, where `iov[0]` has the platform and the `iov[1]` contains the source, each field have a maximum of 50 characters length.

Once added, Pantavisor creates a file using the platform and source files inside the `/pantavisor/logs/current/` directory. For example, if the platform is `Foo` and the source is `var/log/Bar.log` then the directory `/pantavisor/logs/current/Foo/var/log/Bar.log` will be created.

Only one file descriptor can be subscribed using a specific pair platform-source, where platform refers to the container where the logs belong and the src denote where the log is originated (e.g: console.log)

Unsubscribe

To unsubscribe a fd a new call to `sendmsg` must be done with `fd = -1` using the pair platform-source to identify the fd.

Pantavisor Control Socket

The pv-ctrl socket enables communication between the containers and Pantavisor during runtime.

The following subsections describe the behaviour of the HTTP API for the different endpoints, which you can test either using any HTTP client or our pvcontrol tool from the default pvr-sdk container.

> **Note**
>
> The examples provided use cURL, but any HTTP client inside of your container should work.

/CONTAINERS

This endpoint can be used to list the containers (with their status) that are installed in the current revision.

An example of use:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/containers"
```

/GROUPS

These requests can be used to list groups.

To list all groups in the current revision:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/groups"
```

/SIGNAL

This type of command can be issued to alter the container status in Pantavisor.

To send the one-shot readiness signal:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"type\":\"ready\",\"payload\":\"\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
signal
```

This request will fail if the signal type is not supported, or if that status goal is not expected.

/COMMANDS

These commands can perform changes in Pantavisor container engine itself, so the result will not be inmediate to the request.

An example of a command that tells Pantavisor to transition to revision 4:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"op\":\"LOCAL_RUN\",\"payload\":\"4\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
commands
```

As you can see, the body of the request contains the command itself in JSON format.

These are the different commands that are supported. You can test them by substituting `op` and `payload` in the above command:

op	payload	Description
LOCAL_RUN	revision	transition to specified revision
POWEROFF_DEVICE	message	poweroff device with optional message
REBOOT_DEVICE	message	reboot device with optional message
MAKE_FACTORY	revision	make the revision the factory revision. If revision is not set, Pantavisor will use the current one. Device needs to be not claimed
UPDATE_METADATA	{key, value}	upload the json as a new pair of device metadata to Pantacor Hub
RUN_GC	N/A	run garbage collector
ENABLE_SSH	N/A	enable SSH server ignoring config until reboot
DISABLE_SSH	N/A	disable SSH server ignoring config until reboot
GO_REMOTE	N/A	go remote when running on a locals/ revision if allowed by config

/OBJECTS

This endpoint can be used to list, send and receive objects to and from Pantavisor. Bear in mind that objects are the artifacts that form a Pantavisor revision.

An example of listing the objects that are stored in the device:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects"
```

Here, an example for getting one of those objects:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

You can even put new objects in Pantavisor. Notice that the sha256sum of object has to match the specified sha in the URI:

```
curl -X PUT --upload-file object --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

/STEPS

This endpoint can be used to list, send and receive step jsons, as well as get the update progress and set the commit message for any of them.

Let us go with the examples, first, to list all steps installed in the device:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps"
```

To get an existing step json:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

To send a new json, the format of the new revision in the URI has to contain the "locals/" prefix. The name after the prefix must be under 64 characters and must not contain any other "/" character. These revisions that are installed using the socket (locals) are treated in a different way than the ones installed from Pantacor Hub (remotes), as you will have to manually request the transition to locals using the run command. Most importantly, locals will not attempt any communication with Pantacor Hub during runtime unless a go remote command is issued.

```
curl -X PUT --upload-file json --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/locals/example"
```

To get the update progress (DONE, TESTING, INPROGRESS...) and some related information of a revision:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/progress"
```

Finally, you can set a commit message that will be stored along a revision and showed when listing revisions so the user can idendifcate each one of them:

```
curl -X PUT --data "message" --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/commitmsg"
```

/USER-META

The user-meta endpoint offers the ability to list, save and delete user metadata.

To list all user meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta"
```

An example of creating or updating a new user metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

/DEVICE-META

The device-meta endpoint offers the ability to list device metadata.

To list all device meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta"
```

An example of creating or updating a new device metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

/BUILDINFO

For debugging porpuses, it is possible to get the repo manifest that was used to build this Pantavisor binary.

With cURL, this would look like:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/buildinfo"
```

/DRIVERS

The drivers endpoint lets you list load and unload managed drivers.

To list drivers referenced by container and their load state:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl http://localhost/drivers
```

To load manual drivers at bulk from within container:

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/load
```

To load individual manual drivers, in this case one driver named "wifi":

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/load
```

Same for unloading, it can be done at bulk:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/unload
```

And individually:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/unload
```

Pantavisor Configuration

> **Warning**
>
> This is deprecated but still supported for backwards compatibility reasons. To get to the newly unified configuration key syntax, go here.

There are four ways to tweak Pantavisor configuration, depending on when it can be modified: at compile time, at boot up time, after an update or during runtime. Bear in mind that each level in this list will overwrite whatever is configured in the previous ones.

AT COMPILE TIME

You can take a look at the default configuration files we use to build our supported platforms here.

pantavisor.config

Key	Value	Default
policy	string without '/' character	empty
cache.usrmetadir	path	/storage/cache/meta
cache.devmetadir	path	/storage/cache/devmeta
system.apparmor.profiles	comma-separated list of profile names	empty
system.init.mode	embedded, standalone or appengine	embedded
system.libdir	path	/lib
system.etcdir	path	/etc
system.rundir	path	/pv
system.mediadir	path	/media
system.confdir	path	/configs
system.drivers.load_early.auto	0 or 1	0
system.mount.securityfs	0 or 1	0
debug.shell	0 or 1	1
debug.shell.autologin	0 or 1	0
debug.ssh	0 or 1	1
debug.ssh_authorized_keys	path or "default"	/pv/user-meta/pvr-sdk.authorized_keys
dropbear.cache.dir	path	/storage/cache/dropbear
bootloader.type	uboot, uboot-pvk or grub	uboot
bootloader.mtd_only	0 or 1	0
bootloader.mtd_env	path	N/A
libthttp.certsdir	path	"/certs"
secureboot.mode	disabled, audit, lenient or strict	lenient

Key	Value	Default
secureboot.truststore	string	ca-certificates
secureboot.checksum	0 or 1	1
secureboot.handlers	0 or 1	1
storage.device	LABEL=XXXX, UUID=XXXX or string	N/A (mandatory)
storage.fstype	ext4, ubifs or jffs2	N/A (mandatory)
storage.opts	N/A	N/A
storage.mntpoint	path	N/A (mandatory)
storage.mnttype	ext4	disabled
storage.logtempsize	size	disabled
storage.wait	integer	5
storage.gc.reserved	integer	5
storage.gc.keep_factory	0 or 1	0
storage.gc.threshold	integer	0

Key	Value	Default
storage.gc.threshold.defertime	integer	600
disk.voldir	path	/volumes
updater.goals.timeout	integer	120
updater.use_tmp_objects	0 or 1	0
updater.commit.delay	integer	25
revision.retries	integer	10
revision.retries.timeout	integer	120
wdt.enabled	0 or 1	1
wdt.mode	disabled, shutdown, startup or always (experimental)	shutdown
wdt.timeout	integer	15
net.brdev	string	lxcbr0
net.braddress4	ip	10.0.3.1
net.brmask4	ip	255.255.255.0
log.dir	path	/storage/logs/
log.server.outputs		filetree

Key	Value comma-sepated list of the following values: filetree, singlefile, stdout, stdout.pantavisor, stdout.containers, stdout_direct and nullsink	Default
log.maxsize	integer	2097152 B (2 MB)
log.level	0 (FATAL), 1 (ERROR), 2 (WARN), 3 (INFO), 4 (DEBUG) or 5 (ALL)	0
log.buf_nitems	integer	128 KB
log.capture	0 or 1	1
log.capture.dmesg	0 or 1	0
log.loggers	0 or 1	1
log.filetree.timestamp.format	golang:constant or strftime:format	empty
log.singlefile.timestamp.format	golang:constant or strftime:format	empty
log.stdout.timestamp.format	golang:constant or strftime:format	empty
log.stdout	0 or 1	0
libthttp.log.level	0 (FATAL), 1 (ERROR), 2 (WARN), 3 (INFO), 4 (DEBUG) or 5 (ALL)	3
lxc.log.level	integer	2
control.remote	0 or 1	1
control.remote.always	0 or 1	0
sysctl.*	value	empty

Example of pantavisor.config:

```
bootloader.type=normal-uboot
storage.device=LABEL=pvroot
storage.fstype=ext4
storage.opts=
storage.mntpoint=/storage
wdt.enabled=1
wdt.timeout=30
log.level=5
log.buf_nitems=128
sysctl.vm.watermark_boost_level=0
```

pantahub.config

Key	Value	Default	Description
creds.type	builtin and ext-*	builtin	set type of authentication
creds.host	ip	192.168.53.1	Pantacor Hub IP
creds.port	port	12365	Pantacor Hub port
creds.proxy.host	proxy host	NULL	Proxy Host (if enabled)
creds.proxy.port	proxy port	3128	Proxy Port
creds.proxy.noproxyconnect	int	0	Disable Proxyconnect protocol (basic http proxy for TLS)
creds.id	string	unset (set after claim)	device id
creds.prn	string	unset (set after claim)	device prn
creds.secret	string	unset (set after claim)	device secret
creds.tpm.key	string	disabled	TMP key
creds.tpm.cert	string	disabled	TMP certificate
factory.autotok	string	disabled	token for auto claiming
updater.interval	integer	60	time between Pantacor Hub update requests
updater.network_timeout	integer	120	wait time in seconds before rollback if device does not get network connectivity
log.push	0 or 1	1	push stored logs to cloud
metadata.devmeta.interval	integer	10	time between Pantacor Hub devive metadata requests
metadata.usrmeta.interval	integer	5	time between Pantacor Hub user metadata requests

Example of pantahub.config:

```
updater.interval=5
updater.network_timeout=60
updater.commit.delay=40
creds.host=api.pantahub.com
creds.port=443
creds.id=
creds.prn=
creds.secret=
```

AT BOOT UP TIME

There are two options here: using cmdline and policies. Inside of the boot up time options we also have a hierarchy, making cmdline override policies.

Using cmdline

The values from the previous sections can be overriden at boot up time, just writing the key=value pair in `/proc/cmdline` with a prefix before the key. This is tipically done from the bootloader menu.

Any key from pantavisor.config can be overriden appending the *pv_* prefix to the key, while any pantahub.config can do the same with the *ph_* prefix.

Example of cmdline configuration:

```
pv_log.logger=0 ph_log.push=0
```

> **Note**
>
> When using any of the stdout outputs `pv_log.server.outputs=stdout | stdout.pantavisor | stdout.containers`, you can also pass `ignore_loglevel and printk.devkmsg=on` to kernel through cmdline to make all messages go to console

Using policies

Policies can be added to the platform vendor repo. To set a policy, we use the policy key or the pv_policy in case cmdline is used.

A policy is just a file with the same format as pantavisor.config that is added as an addiitonal layer during boot up time. All keys are allowed except the following ones:

- policy
- system.init.mode
- cache.usrmetadir
- cache.devmetadir
- system.libdir
- system.etcdir
- system.rundir
- system.mediadir
- system.confdir

AFTER AN UPDATE

Some of the values from the previous sections can be overridden when making a new revision. To do so, you must create a new file, following the same format we use for compile time configuration and reference it from bsp/run.json.

These are the keys that are accepted at this level of configuration. See the compile time section for reference:

- creds.proxy.host
- creds.proxy.port
- creds.proxy.noproxyconnect
- storage.gc.reserved
- storage.gc.keep_factory
- storage.gc.threshold
- storage.gc.threshold.defertime
- updater.goals.timeout
- updater.use_tmp_objects
- updater.keep_factory
- updater.interval
- updater.network_timeout
- updater.commit.delay
- revision.retries
- revision.retries.timeout
- log.maxsize
- log.level
- log.buf_nitems
- log.push
- log.capture
- log.loggers
- log.stdout
- log.server.outputs
- log.capture.dmesg
- log.filetree.timestamp.format
- log.singlefile.timestamp.format
- log.stdout.timestamp.format
- libthttp.log.level
- lxc.log.level
- metadata.devmeta.interval
- metadata.usrmeta.interval
- wdt.enabled
- wdt.mode
- wdt.timeout

For example, this is how this file would look like if we wanted to overwrite the log level and the garbage collector threshold:

```
log.level=3
storage.gc.threshold=30
```

DURING RUNTIME

User Metadata

> **Note**
>
> If the user metadata volume is assigned to a permanent volume, these changes will persist over device reboots.

User metadata can be used to override the values from the previous sections.

These are the keys that can be overrriden from user metadata configuration. See the compile time section for reference on them:

- debug.ssh
- storage.gc.reserved
- storage.gc.keep_factory
- storage.gc.threshold
- storage.gc.threshold.defertime
- updater.interval
- log.level
- log.push
- libthttp.log.level
- metadata.devmeta.interval
- metadata.usrmeta.interval

Pantavisor Control Socket

> **Note**
>
> The configuration changes made by this method are volatile and thus will not persist after a device reboot. In that case, the value will fallback to the previously presented highest configuration level.

In this case, there is only one possible value that can be overridden by Pantavisor Commands. See the compile time section for reference:

- debug.ssh

Pantavisor Configuration

> **Note**
>
> The configuration syntax is common for all levels, but not all levels support the same keys. Our reference contains the list of keys and the allowed levels for each one.

There are several ways to set Pantavisor configuration, depending on when it can be modified in the Pantavisor life cycle. Bear in mind that not all configuration parameters will be available for all levels. Furthermore, each level will overwrite whatever is configured in the previous ones, following this order:

1. pantavisor.config
2. pantahub.config
3. Policies
4. OEM
5. cmdline
6. Environment Variables
7. User Metadata
8. Commands

PANTAVISOR.CONFIG

Configuration file for Pantavisor. It can only be changed at build time.

PANTAHUB.CONFIG

Build time configuration file for Pantavisor built-in Pantacor Hub client.

POLICIES

Policies are added at build time from the vendor skel directory, but loaded during boot up time.

To select a policy among the installed ones, we need to set its name to the `PV_POLICY` key either from pantavisor.config or environment variables.

OEM

For setups where we want to modify the configuration based on device updates, we offer the possibility to attach a configuration file to a revision.

Its location inside the revision will be defined by the configuration values of the keys `PV_OEM_NAME` and `PV_POLICY` from pantavisor.config, environment variables or policies levels.

CMDLINE

> **Warning**
>
> This method is *DEPRECATED* but still supported for backwards compatibility reasons. It is recommended to use env variables instead.

Right after loading the configuration files, Pantavisor reads `/proc/cmdline` in search for `key=value` pairs that use the prefix `ph_` or `pv_`. This can be done from the bootloader console.

ENVIRONMENT VARIABLES

Linux environment variables can be used to configure Pantavisor. To do that, the rules to set env variables have to be followed:

- Use `key=value` format
- Do not use `.`
- If `"` characted is needed, you can escape them by using `\"` between the config item. For example: `"PV_SYSCTL_KERNEL_CORE_PATTERN=|/lib/pv/pvcrash --skip"`.

These variables need to be set during boot time, and setting them after that will have no effect on Pantavisor. This can be achieved from the bootloader console.

USER METADATA

> **Note**
>
> If the user metadata volume is assigned to a permanent volume, as it is by default, these changes will persist over device reboots.

User metadata can be used to override any of the previously presented configuration mechanisms.

There is a number of ways of setting user metadata, depending on the device management method choice. Go to our how-to use Pantavisor guide for more information.

COMMANDS

> **Note**
>
> It is important to notice that these changes will not persist after a device reboot in any case.

The Pantavisor control socket, or consequently the PVControl tool, offers another way to change a very limited subset of configuration values. Specifically, using the command endpoint.

Pantavisor Configuration

> **Note**
>
> This reference page presents the newly unified configuration key syntax. To get to the deprecated but still supported previous format, you will have to go here.

SUMMARY

> **Note**
>
> The key syntax is the same for all configuration levels.

> **Note**
>
> All keys are case insensitive.

> **Note**
>
> Syntax and behavior of keys tagged with (experimental) might change and break backwards compatibility.

This table contains the currently supported list of configuration keys.

Key	Value	Default
`PH_CREDS_HOST`	IP or hostname	`192.168.53.1`
`PH_CREDS_ID`	string	empty
`PH_CREDS_PORT`	port	`12365`
`PH_CREDS_PROXY_HOST`	IP or hostname	empty
`PH_CREDS_PROXY_NOPROXYCONNECT`	`0` or `1`	`0`
`PH_CREDS_PROXY_PORT`	port	`3218`
`PH_CREDS_PRN`	string	empty
`PH_CREDS_SECRET`	string	empty
`PH_CREDS_TYPE`	`builtin`	`builtin`
`PH_FACTORY_AUTOTOK`	token	empty
`PH_LIBEVENT_HTTP_TIMEOUT`	time (in seconds)	`60`
`PH_LIBEVENT_HTTP_RETRIES`	number of retries	`1`
`PH_METADATA_DEVMETA_INTERVAL`	time (in seconds)	`10`
`PH_METADATA_USRMETA_INTERVAL`	time (in seconds)	`10`
`PH_ONLINE_REQUEST_THRESHOLD`	number of failures	`0`
`PH_UPDATER_INTERVAL`	time (in seconds)	`60`

Key	Value	Default
`PH_UPDATER_NETWORK_TIMEOUT`	time (in seconds)	`120`
`PH_UPDATER_TRANSFER_MAX_COUNT`	number of transfers	`5`
`PV_BOOTLOADER_FITCONFIG`	string	empty
`PV_BOOTLOADER_MTD_ENV`	string	empty
`PV_BOOTLOADER_MTD_ONLY`	`0` or `1`	`0`
`PV_BOOTLOADER_TYPE`	`uboot`, `uboot-ab`, `uboot-pvk`, `rpiab` or `grub`	`uboot`
`PV_BOOTLOADER_UBOOTAB_A_NAME`	string	fitA
`PV_BOOTLOADER_UBOOTAB_A_NAME`	string	fitB
`PV_BOOTLOADER_UBOOTAB_ENV_NAME`	string	empty
`PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME`	string	empty
`PV_BOOTLOADER_UBOOTAB_ENV_OFFSET`	offset in bytes	0
`PV_BOOTLOADER_UBOOTAB_ENV_SIZE`	size in bytes	0
`PV_CACHE_DEVMETADIR`	path	`/storage/cache/devmeta`
`PV_CACHE_USRMETADIR`	path	`/storage/cache/meta`
`PV_CONTROL_REMOTE`	`0` or `1`	`1`
`PV_CONTROL_REMOTE_ALWAYS`	`0` or `1`	`0`
`PV_DEBUG_SHELL`	`0` or `1`	`1`

Key	Value	Default
`PV_DEBUG_SHELL_AUTOLOGIN`	`0` or `1`	`0`
`PV_DEBUG_SHELL_TIMEOUT`	time (in seconds)	`60`
`PV_DEBUG_SSH`	`0` or `1`	`1`
`PV_DEBUG_SSH_AUTHORIZED_KEYS`	string	empty
`PV_DISK_EXPORTSDIR`	path	`/exports`
`PV_DISK_VOLDIR`	path	`/volumes`
`PV_DISK_WRITABLEDIR`	path	`/writable`
`PV_DROPBEAR_CACHE_DIR`	path	`/storage/cache/dropbear`
`PV_LIBEVENT_DEBUG_MODE`	`0` or `1`	`0`
`PV_LIBTHTTP_CERTSDIR`	path	`/certs`
`PV_LIBTHTTP_LOG_LEVEL`	`0` (FATAL), `1` (ERROR), `2` (WARN), `3` (INFO), `4` (DEBUG) or `5` (ALL)	`3`
`PV_LOG_CAPTURE`	`0` or `1`	`1`
`PV_LOG_CAPTURE_DMESG`	`0` or `1`	`1`
`PV_LOG_BUF_NITEMS`	size (in KB)	`128`
`PV_LOG_DIR`	path	`/storage/logs/`
`PV_LOG_FILETREE_TIMESTAMP_FORMAT`	`golang:` constant or `strftime:` format	empty
`PV_LOG_LEVEL`	`0` (FATAL), `1` (ERROR), `2` (WARN), `3` (INFO), `4` (DEBUG) or `5` (ALL)	`0`
`PV_LOG_LOGGERS`	`0` or `1`	`1`
`PV_LOG_MAXSIZE`	size (in B)	`2097152` (2 MB)
`PV_LOG_PUSH`	`0` or `1`	`1`

Key	Value	Default
`PV_LOG_SERVER_OUTPUTS`	comma-separated list of: `filetree`, `nullsink`, `singlefile`, `stdout`, `stdout_direct`, `stdout.containers` and/or `stdout.pantavisor`	`filetree`
`PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT`	`golang:` constant or `strftime:` format	empty
`PV_LOG_STDOUT_TIMESTAMP_FORMAT`	`golang:` constant or `strftime:` format	empty
`PV_LXC_LOG_LEVEL`	`0` (TRACE), `1` (DEBUG), `2` (INFO), `3` (NOTICE), `4` (WARN), `5` (ERROR), `6` (CRITICAL), `7` (ALERT) or `8` (FATAL)	`2`
`PV_NET_BRADDRESS4`	IP or hostname	`10.0.3.1`
`PV_NET_BRDEV`	string	`lxcbr0`
`PV_NET_BRMASK4`	IP or hostname	`255.255.255.0`
`PV_OEM_NAME`	string	empty
`PV_POLICY`	string without `/` character	empty
`PV_REMOUNT_POLICY`	string	"default"
`PV_REVISION_RETRIES`	number of retries	`10`
`PV_SECUREBOOT_CHECKSUM`	`0` or `1`	`1`
`PV_SECUREBOOT_HANDLERS`	`0` or `1`	`1`
`PV_SECUREBOOT_MODE`	`disabled`, `audit`, `lenient` or `strict`	`lenient`
`PV_SECUREBOOT_OEM_TRUSTSTORE`	string	`ca-oem-certificates`
`PV_SECUREBOOT_TRUSTSTORE`	string	`ca-certificates`

Key	Value	Default
`PV_STORAGE_DEVICE`	`LABEL=` XXXX, `UUID=` XXXX or string	N/A (mandatory)
`PV_STORAGE_FSTYPE`	`ext4`, `ubifs` or `jffs2`	N/A (mandatory)
`PV_STORAGE_GC_KEEP_FACTORY`	`0` or `1`	`0`
`PV_STORAGE_GC_RESERVED`	percentage	`5`
`PV_STORAGE_GC_THRESHOLD_DEFERTIME`	time (in seconds)	`600`
`PV_STORAGE_GC_THRESHOLD`	percentage	`0`
`PV_STORAGE_LOGTEMPSIZE`	size (with k, m g or % suffix)	empty
`PV_STORAGE_MNTPOINT`	path	N/A (mandatory)
`PV_STORAGE_MNTTYPE`	`ext4`	empty
`PV_STORAGE_PHCONFIG_VOL`	`0` or `1`	`0`
`PV_STORAGE_WAIT`	time (in seconds)	`5`
`PV_SYSCTL_KERNEL_CORE_PATTERN`	core pattern	`string|/lib/pv/pvcrash --skip`
`PV_SYSCTL_` *	sysctl.conf format	N/A (mandatory)
`PV_SYSTEM_APPARMOR_PROFILES`	comma-separated list of AppArmor profile names	empty
`PV_SYSTEM_CONFDIR`	path	`/configs`

Key	Value	Default
`PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO`	`0` or `1`	`0`
`PV_SYSTEM_ETCDIR`	path	`/etc`
`PV_SYSTEM_INIT_MODE`	`embedded`, `standalone` or `appengine`	`embedded`
`PV_SYSTEM_LIBDIR`	path	`/lib`
`PV_SYSTEM_MEDIADIR`	path	`/media`
`PV_SYSTEM_MOUNT_SECURITYFS`	`0` or `1`	`0`
`PV_SYSTEM_RUNDIR`	path	`/pv`
`PV_SYSTEM_USRDIR`	path	`/usr`
`PV_UPDATER_COMMIT_DELAY`	time (in seconds)	`25`
`PV_UPDATER_GOALS_TIMEOUT`	time (in seconds)	`120`
`PV_UPDATER_USE_TMP_OBJECTS`	`0` or `1`	`0`
`PV_VOLMOUNT_DM_EXTRA_ARGS`	veritysetup options	empty
`PV_WDT_MODE`	`disabled`, `shutdown`, `startup` or `always`	`shutdown`
`PV_WDT_TIMEOUT`	time (in seconds)	`15`

LEVELS

This table shows the configuration levels that are allowed for each configuration key.

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PH_CREDS_HOST	✗	✓	✓	✓	✓
PH_CREDS_ID	✗	✓	✓	✓	✓
PH_CREDS_PORT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_HOST	✗	✓	✓	✓	✓
PH_CREDS_PROXY_NOPROXYCONNECT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_PORT	✗	✓	✓	✓	✓
PH_CREDS_PRN	✗	✓	✓	✓	✓
PH_CREDS_SECRET	✗	✓	✓	✓	✓
PH_CREDS_TYPE	✗	✓	✓	✓	✓
PH_FACTORY_AUTOTOK	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_TIMEOUT	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_RETRIES	✗	✓	✓	✓	✓
PH_METADATA_DEVMETA_INTERVAL	✗	✓	✓	✓	✓
PH_METADATA_USRMETA_INTERVAL	✗	✓	✓	✓	✓
PH_ONLINE_REQUEST_THRESHOLD	✗	✓	✓	✓	✓
PH_UPDATER_INTERVAL	✗	✓	✓	✓	✓
PH_UPDATER_NETWORK_TIMEOUT	✗	✓	✓	✓	✓
PH_UPDATER_TRANSFER_MAX_COUNT	✗	✓	✓	✓	✓
PV_BOOTLOADER_FITCONFIG	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ENV	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ONLY	✓	✗	✓	✓	✗
PV_BOOTLOADER_TYPE	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_A_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_A_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_OFFSET	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_SIZE	✓	✗	✓	✓	✗
PV_CACHE_DEVMETADIR	✓	✗	✓	✓	✗
PV_CACHE_USRMETADIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_CONTROL_REMOTE	✓	✗	✓	✓	✓
PV_CONTROL_REMOTE_ALWAYS	✓	✗	✓	✓	✓
PV_DEBUG_SHELL	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_AUTOLOGIN	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_TIMEOUT	✓	✗	✓	✓	✗
PV_DEBUG_SSH	✓	✗	✓	✓	✓
PV_DEBUG_SSH_AUTHORIZED_KEYS	✓	✗	✓	✓	✓
PV_DISK_EXPORTSDIR	✓	✗	✓	✓	✗
PV_DISK_VOLDIR	✓	✗	✓	✓	✗
PV_DISK_WRITABLEDIR	✓	✗	✓	✓	✗
PV_DROPBEAR_CACHE_DIR	✓	✗	✓	✓	✗
PV_LIBEVENT_DEBUG_MODE	✓	✗	✓	✓	✓
PV_LIBTHTTP_CERTSDIR	✓	✗	✓	✓	✗
PV_LIBTHTTP_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_CAPTURE	✓	✗	✓	✓	✓
PV_LOG_CAPTURE_DMESG	✓	✗	✓	✓	✓
PV_LOG_BUF_NITEMS	✓	✗	✓	✓	✓
PV_LOG_DIR	✓	✗	✓	✓	✗
PV_LOG_FILETREE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_LOGGERS	✓	✗	✓	✓	✓
PV_LOG_MAXSIZE	✓	✗	✓	✓	✓
PV_LOG_PUSH	✓	✗	✓	✓	✓
PV_LOG_SERVER_OUTPUTS	✓	✗	✓	✓	✓
PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_STDOUT_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LXC_LOG_LEVEL	✓	✗	✓	✓	✓
PV_NET_BRADDRESS4	✓	✗	✓	✓	✓
PV_NET_BRDEV	✓	✗	✓	✓	✓

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PV_NET_BRMASK4	✓	✗	✓	✓	✓
PV_OEM_NAME	✓	✗	✓	✓	✗
PV_POLICY	✓	✗	✓	✗	✗
PV_REMOUNT_POLICY	✗	✗	✓	✗	✗
PV_REVISION_RETRIES	✓	✗	✓	✓	✓
PV_SECUREBOOT_CHECKSUM	✓	✗	✓	✓	✗
PV_SECUREBOOT_HANDLERS	✓	✗	✓	✓	✗
PV_SECUREBOOT_MODE	✓	✗	✓	✓	✗
PV_SECUREBOOT_OEM_TRUSTSTORE	✓	✗	✓	✓	✗
PV_SECUREBOOT_TRUSTSTORE	✓	✗	✓	✓	✗
PV_STORAGE_DEVICE	✓	✗	✓	✓	✗
PV_STORAGE_FSTYPE	✓	✗	✓	✓	✗
PV_STORAGE_GC_KEEP_FACTORY	✓	✗	✓	✓	✓
PV_STORAGE_GC_RESERVED	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD_DEFERTIME	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD	✓	✗	✓	✓	✓
PV_STORAGE_LOGTEMPSIZE	✓	✗	✓	✓	✗
PV_STORAGE_MNTPOINT	✓	✗	✓	✓	✗
PV_STORAGE_MNTTYPE	✓	✗	✓	✓	✗
PV_STORAGE_WAIT	✓	✗	✓	✓	✗
PV_SYSTEM_APPARMOR_PROFILES	✓	✗	✓	✓	✗
PV_SYSTEM_CONFDIR	✓	✗	✓	✓	✗
PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO	✓	✗	✓	✓	✗
PV_SYSTEM_ETCDIR	✓	✗	✓	✓	✗
PV_SYSTEM_INIT_MODE	✓	✗	✓	✓	✗
PV_SYSTEM_LIBDIR	✓	✗	✓	✓	✗
PV_SYSTEM_MEDIADIR	✓	✗	✓	✓	✗
PV_SYSTEM_MOUNT_SECURITYFS	✓	✗	✓	✓	✗
PV_SYSTEM_RUNDIR	✓	✗	✓	✓	✗
PV_SYSTEM_USRDIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_UPDATER_COMMIT_DELAY	✓	✗	✓	✓	✓
PV_UPDATER_GOALS_TIMEOUT	✓	✗	✓	✓	✓
PV_UPDATER_USE_TMP_OBJECTS	✓	✗	✓	✓	✓
PV_VOLMOUNT_DM_EXTRA_ARGS	✓	✗	✓	✓	✓
PV_WDT_MODE	✓	✗	✓	✓	✗
PV_WDT_TIMEOUT	✓	✗	✓	✓	✗

Pantavisor Metadata

This page contains reference information about Pantavisor metadata.

DEVICE METADATA

This is the device metadata created by Pantavisor that will give you useful information about your device:

Key	Value	Description
interfaces	json	network interfaces of the device
pantahub.address	IP:port	Pantacor Hub address the client is communicating with
pantahub.claimed	0 or 1	1 if claimed in Pantacor Hub
pantahub.online	0 or 1	1 of connection to Pantacor Hub was established
pantahub.state	init, register, claim, sync, login, wait hub, report, idle, prep download or download	see Pantacor Hub states
pantavisor.arch	string	CPU architecture
pantavisor.cpumodel	string	CPU model name
pantavisor.mode	local or remote	see operation modes
pantavisor.revision	string	revision number
pantavisor.status	string	revision status
pantavisor.uname	json	uname output
pantavisor.version	string	Pantavisor build
storage	json	disk usage of the device
sysinfo	json	sysinfo
time	json	time information

User metadata

This is the user metadata that can be set by the user which is parsed and have some actions on Pantavisor:

Key	Value	Description
pvr-sdk.authorized_keys	SSH pub key	set public key to get SSH access
pvr-auto-follow.url	URL	device will automatically pull every change in the device associated to that clone URL
pantahub.log.push	0 or 1	disable/enable log pusshing to Pantacor Hub. Enabled by default. Overrides log.push
config-key	config-value	override some of the configuration values
container/key	value	send user metadata that can be consumed by one of the containers

Pantavisor State Format

Pantavisor state format describes the json format to configure the Pantavisor run-time state. In this page, we are going to go through the System State Format v2, which is the version we currently use.

> **Note**
>
> This reference is for advanced users. Before manually editing these values in your device, check if what you intend to do can be achieved with the help of pvr.

SPEC FORMAT

A complete system is made up of one BSP as well as one to many containers. The state allows this following this format:

```
{
    "#spec": "pantavisor-service-system@1",

    "README.md": "xxx",

    "bsp/run.json": {..},
    "bsp/file1....": {..},

    "<container1>/run.json": {..},
    "<container1>/file1...": "xxx",

    "disks.json": {..},

    "_sigs/<container1>.json": {..},

    "_config/<container1>/file1...": "xxx"
}
```

This table defines all possible keys for the state format:

Key	Value	Default	Description
#spec	*pantavisor-service-system@1*	**mandatory**	version of the state format
README.md	string	empty	readme file in markdown format
bsp/run.json	bsp/run.json	**mandatory**	bsp run-time configuration file
bsp/drivers.json	bsp/drivers.json	empty	managed drivers
<container>/run.json	container/run.json	**mandatory**	container run-time configuration file
disks.json	disks.json	empty	definition of physical store medium
groups.json	groups.json	empty	definition of container groups
device.json	device.json	empty	definition of disks and groups
_sigs/container.json	_sigs/container.json	empty	signature file
_config/<container>/<path>	_config/container/	empty	configuration files

bsp/run.json

This file allows to configure the BSP. The JSON follows this format:

```
{
    "addons": [
        "addon-plymouth.cpio.xz4"
    ],
    "firmware": "firmware.squashfs",
    "initrd": "pantavisor",
    "linux": "kernel.img",
    "modules": "modules.squashfs"
}
```

This table defines all possible keys for the BSP run.json:

Key	Value	Default	Description
addons	list of paths	**mandatory**	list of relative paths to the initrd rootfs addons
firmware	path	**mandatory**	relative path to Linux firmware file
initrd	path	**mandatory**	relative path to Pantavisor binary file
linux	path	**mandatory**	relative path to Linux kernel file
modules	path	**mandatory**	relative path to Linux modules file

bsp/drivers.json

This JSON file can be used to set a list of managed drivers, following this format:

```
{
    "#spec": "driver-aliases@1",
    "all": {
        "bluetooth": [
            "hci_uart",
            "btintel"
        ],
        "wifi": [
            "iwlwifi ${user-meta:drivers.iwlwifi.opts}"
        ]
    },
    "dtb:broadcom/bcm2711-rpi-4-b.dtb": {
        "wifi": [
            "cfg80211 ${user-meta:drivers.cfg80211.opts}",
            "brcmfmac ${user-meta:drivers.brcmfmac.opts}"
        ]
    }
}
```

This table defines all supported keys for the BSP drivers.json:

Key	Value	Default	Description
#spec	*driver-aliases@1*	**mandatory**	version of the drivers JSON
all	list of managed drivers	**mandatory**	list of managed driver JSONs
dtb:	list of managed drivers specific to dtb info provided by bootloader through pv_bootloader.dtb= cmdline paramaneter	**optional**	list of managed driver JSONs
ovl:	list of managed drivers specific to overlay info provided by bootloader through pv_bootloader.ovl= cmdline paramaneter	**experimental**	list of managed driver JSONs (experimental - semantic change or go away)

managed driver

Each driver JSON is composed by a list of Kernel modules. It follows this format:

```
"bluetooth": [
    "hci_uart",
    "btintel ${user-meta:drivers.btintel.opts}"
]
```

This table defines all supported keys for the BSP driver JSON:

Key	Value	Default	Description
name	string	**mandatory**	name of the driver
modules	list of strings	empty	module names with optional user or device metadata key where the load arguments are stored

multi board BSP drivers

To facilitate an ecosystem where a single BSP can support multiple boards effectively we are offering a mechanism to map abstract drivers to different modules or properties based on the dtb and overlay information provided by the bootloader to the kernel.

The initial example from above shows how to overload the definition of the abstract `wifi` driver for a given dtb info provided by the bootloader to the kernel:

```
    "dtb:broadcom/bcm2711-rpi-4-b.dtb": {
        "wifi": [
            "cfg80211 ${user-meta:drivers.cfg80211.opts}",
            "brcmfmac ${user-meta:drivers.brcmfmac.opts}"
        ]
    }
```

This will ensure that a kernel booted with the dtb `broadcom/bcm2711-rpi-4-b.dtb` will use the brcmfmac driver and not the iwlwifi driver from the `all` section.

OEM configuration

This file allows to overrride some configuration keys. It will only be processed if `PV_OEM_NAME` is configured, with the following behavior:

- If `PV_POLICY` is set too, the file is loaded from the revision at `<PV_OEM_NAME>/<PV_POLICY>.config`.
- If `PV_POLICY` is not set, the file is is loaded from `<PV_OEM_NAME>/default.config`.
- If the file is not found in the revision, we continue normally with a warning on logs.

container/run.json

This file allows to individually configure each container. This JSON follows this format:

```
{
    "#spec": "service-manifest-run@1",
    "config": "lxc.container.conf",
    "name": "awconnect",
    "root-volume": "root.squashfs",
    "status_goal": "STARTED",
    "storage": {
        "docker--etc-NetworkManager-system-connections": {
            "persistence": "permanent"
        },
        "lxc-overlay": {
            "persistence": "boot"
        }
    },
    "type": "lxc",
    "volumes": [],
    "drivers": {
        "required": [ "ethernet" ]
    }
}
```

This table defines all possible keys for the container run.json:

Key	Value	Default	Description
#spec	*service-manifest-run@1*	**mandatory**	version of the run JSON
config	path	**mandatory**	relative path to the container configuration file
name	string	**mandatory**	name of the container
root-volume	path	**mandatory**	relative path to the container rootfs volume file
storage	storage	**mandatory**	container storage configuration
type	*lxc*	**mandatory**	type of container
volumes	list of paths	**mandatory**	relative paths to the additional container volumes
logs	list of loggers	empty	list of log configuration JSONS
group	string	empty	name of group
runlevel	*data*, *root*, *platform* or *app*	*root* or _platform	_DEPRECATED: use group instead
roles	*mgmt* or *nobody*	empty	list of roles for the container
status_goal	*MOUNTED*, *STARTED* or *READY*	*MOUNTED* or *STARTED*	status goal
restart_policy	*system* or *container*	*system* or *container*	restart policy
drivers	list of container drivers	empty	list of referenced managed drivers for the container
exports	list of paths	empty	list of paths to be mounted to the host

volume

This string can define container volumes. It follows this format:

```
[_handler_:]path
```

The path is the relative path for the volume file in the state JSON. The optional *handler* allows to mount/umnount volumes using the referenced handler script.

For example, to set a root file system for a container that will be validated using dm-verity:

```
"root-volume": "dm:rootfs.squashfs"
```

storage

This JSON allows to configure the storage of a container or to define additional Pantavisor volumes. It is important to notice that you can configure the overlay or any of the container rootfs paths:

```
{
    "lxc-overlay":{
        "persistence":"boot"
    },
    "etc/example/": {
        "persistence": "permanent"
    },
    "docker--etc-modules-load.d": {
        "disk": "built-in",
        "persistence": "permanent"
    },
    "docker--var-pv": {
        "disk": "dm-versatile",
        "persistence": "permanent"
    },
```

```
    ...
}
```

This table defines all possible keys for the container storage JSON:

Key	Value	Default	Description
persistence	*permanent*, *revision* or *boot*	**mandatory**	
disk	string	empty	name of disk defined in [disks.json]

logger

This allows to configure a logger inside of a container. It looks like this:

```
{
    "file":"/var/log/syslog",
    "maxsize":102485760,
    "truncate":true,
    "name":"alpine-logger"
}
```

This table defines all possible keys for the container logger JSON:

Key	Value	Default	Description
file, *lxc* or *console*	string	**mandatory**	path of a file in the rootfs in case of *file* key. *enable* in case of *lxc* or *console* key
maxsize	integer	**mandatory**	max file size in bytes before truncating
truncate	*true* or *false*	**mandatory**	truncate the file when reaching maxsize
name	string	**mandatory**	logger daemon name

container driver

This JSON allows to configure a container driver. An example would look like:

```
"optional": [ "usbnet", "wifi" ]
```

Key	Value	Default	Description
loading	*required*, *optional* or *manual*	**mandatory**	how and when to load the driver
drivers	list of managed driver names	empty	the managed drivers to be loaded

disks.json

This JSON allows to add new disks. An example of this JSON shows it is just a list of JSONs with the following format:

```
[
    {
        "default": "yes",
        "name": "built-in",
        "path": "/storage/disks/",
        "type": "directory"
    },
    {
        "name": "dm-versatile",
        "options": "....",
        "path": "/storage/dm-crypt-file/disk1/versatile.img,8,versatile_key",
        "type": "dm-crypt-versatile"
    },
    {
        "name": "dm-caam",
        "options": "....",
        "path": "/storage/dm-crypt-file/disk1/caam.img,8,caam_key",
        "type": "dm-crypt-caam"
    },
    {
        "name": "dm-dcp",
        "options": "....",
        "path": "/storage/dm-crypt-file/disk1/dcp.img,8,dcp_key",
        "type": "dm-crypt-dcp"
```

```
    }
]
```

This table defines the keys for each disk:

Key	Value	Default	Description
default	*yes* or *no*	*no*	all volumes without specific disk name will use this disk
name	string	**mandatory**	unique name of the disk to be used in container storage
path	path	**mandatory**	destination mount path in case of directory. For crypt disk, please refer below
type	*directory*, *dm-crypt-versatile*, *dm-crypt-caam* or *dm-crypt-dcp*	**mandatory**	type of disk
options	string	empty	options to be passed for mount command -o

Device mapper crypt disk follows a special pattern in path:

```
image,size,key
```

image: disk path which will be encrypted (created during first boot if not present) size: size in MB used to create the disk file key: name of the key file which needs to be created using respective tools (e.g. caam-keygen for i.Mx CAAM)

For example:

```
"path": "/storage/dm-crypt-file/disk1/file.img,8,key"
```

disks_v2.json

The new JSON supports all the previous disk and adds two new types:

```
[
    {
        "name": "zram-swap",
        "type": "swap-disk",
        "format": "swap",
         "format_options": "",
        "provision": "zram",
        "provision_options": "...",
         "mount_options": "..."
    },
    {
        "name": "zram-volume",
        "type": "volume-disk",
        "format": "ext4",
         "format_options": "",
        "provision": "zram",
        "provision_options": "...",
        "mount_target": "...",
         "mount_options": "..."
    }
]
```

The first one is a swap disk based on zram, while the second is a volume disk, which could be used to mount volumes for containers or for Pantavasor itself.

To define this new types, the following keys are available:

key	value	default	description
name	string	**mandatory**	unique name of the disk to be used in container storage
type	string	**mandatory**	type of disk to create: `volume-disk` or `swap-disk`
format	string	**mandatory**	new disk format, possible values are: ext3, ext4 or swap
format_options	string	empty	format options for the mkfs command (see the man page)
provision	string	**mandatory**	define the underlying resource to create the disk. At the moment only `zram` is available
provision_options	string	empty	comma separate key=value pairs to set options for the current provision
mount_target	string (path)	empty	mount point for the current disk. Only necessary for `volume-disk`
mount_options	string	empty	comma separated flags for mount syscall. See `mountflags` in the manual for more information

groups.json

This JSON can be used to overload the default groups. This is the default JSON that will be used by Pantavisor if groups.json is not present in the state, but can be used to illustrate how any groups.json would be formed:

```
[
    {
        "name":"data",
        "description":"Containers which volumes we want to mount but not to be started",
        "restart_policy": "system",
        "timeout": 30,
        "status_goal": "MOUNTED"
    },
    {
        "name":"root",
        "description":"Container or containers that are in charge of setting network connectivity up for the board",
        "restart_policy": "system",
        "timeout": 30,
        "status_goal": "STARTED"
    },
    {
        "name":"platform",
        "description": "Middleware and utility containers",
        "restart_policy": "system",
        "timeout": 30,
        "status_goal": "STARTED"
    },
    {
        "name":"app",
        "description": "Application level containers",
        "restart_policy": "container",
        "timeout": 30,
        "status_goal": "STARTED"
    }
]
```

This table defines the keys for each group:

Key	Value	Default	Description
name	string	**mandatory**	unique name of the group to be used from container
description	string	empty	human readable description
restart_policy	*system* or *container*	*container*	default restart policy
status_goal	*MOUNTED*, *STARTED* or *READY*	*STARTED*	default status goal
timeout	integer	PV_UPDATER_GOALS_TIMEOUT	default timeout for the group

device.json

This JSON substitutes both disks.json and groups.json. Using it along any of the two will result in an invalid state JSON, as it encompasses their contents.

```
{
    "disks": [
        {
            "name": "dm-internal-secrets",
            "path": "/storage/dm-crypt-files/dm-internal-secrets/versatile.img,2,versatile_key-internal_secrets",
            "type": "dm-crypt-versatile"
        }
    ],
    "groups": [
        {
            "description": "Containers which volumes we want to mount but not to be started",
            "name": "data",
            "restart_policy": "system",
            "status_goal": "MOUNTED",
            "timeout": 30
        },
        {
            "description": "Container or containers that are in charge of setting network connectivity up for the board",
            "name": "root",
            "restart_policy": "system",
            "status_goal": "STARTED",
            "timeout": 30
        },
        {
            "description": "Middleware and utility containers",
            "name": "platform",
            "restart_policy": "system",
            "status_goal": "STARTED",
            "timeout": 30
        },
        {
            "description": "Application level containers",
            "name": "app",
            "restart_policy": "container",
            "status_goal": "STARTED",
            "timeout": 30
        }
    ],
    "volumes": {
        "pv--devmeta": {
            "disk": "dm-internal-secrets",
            "persistence": "permanent"
        },
        "pv--usrmeta": {
            "disk": "dm-internal-secrets",
            "persistence": "permanent"
        },
        "pv--phconfig": {
            "disk": "dm-internal-secrets",
            "persistence": "permanent"
        }
    }
}
```

This table defines the keys for each group:

Key	Value	Default	Description
disks	list of disks	empty	definition of physical store medium
groups	list of groups	empty	definition of container groups
volumes	storage	empty	definition of additional volumes

_sigs/container.json

This JSON allows to sign a group of artifacts from the state. It must follow the standard JWS format:

```
{
    "#spec": "pvs@2",
    "protected": "XXXX",
    "signature": "XXXX"
}
```

This table defines the keys for the signature JSON:

Key	Value	Default	Description
#spec	*pvs@2*	**mandatory**	version of the signature JSON
protected	base64 encoded protected	**mandatory**	information about how the signature was created
signature	base64 encoded hash	**mandatory**	the signature itself

protected

This JSON allows to reproduce the signature hash from _sigs/container.json. It must follow the standard JWS format:

```
{
    "alg":"RS256",
    "typ":"PVS",
    "pvs": {
        "include":["pv-avahi/**","_config/pv-avahi/**"],
        "exclude":["pv-avahi/src.json"]
    },
    "x5c": [
        "XXXX",
        "XXXX"
    ]
}
```

Key	Value	Default	Description
alg	*RS256*, *ES256*, *ES384* or *ES512*	**mandatory**	cryptography algorithm to generate the signature
typ	*PVS*	**mandatory**	type of signature
pvs	lists of *included* and *excluded* paths	**mandatory**	relative paths to the files that were used to generate the signature
x5c	list of certificate strings	empty	certificate that contains the public key plus more certificates from the chain of trust

_config/container/

This directory can be used to attach additional files to the container rootfs.

For example, if we wanted to add or overwrite the main pvr-sdk configuration file, we would do it by adding the desired file to the _config directory in the JSON state:

```
_config/pvr-sdk/etc/pvr-sdk/config.json
```

Which would set that file in the pvr-sdk container rootfs like this:

```
/etc/pvr-sdk/config.json
```

Those files will be attached using the default owner and permissions:

- Owner: 0:0 (global root:root)
- Mod: 0644 (u+rw,g+r,o+r)

FORMAT EXCEPTIONS

The files in this section are meant to be used for tooling (like pvr) and thus ignored by Pantavisor.

bsp/src.json

```
{
  "#spec": "bsp-manifest-src@1",
  "pvr": "https://pvr.pantahub.com/pantahub-ci/arm_rpi64_bsp_latest#bsp"
}
```

bsp/build.json

```
{
  "altrepogroups": "",
  "branch": "master",
  "commit": "e2a4911eb35de2032e85f74c8f239de81c6f622b",
  "gitdescribe": "014-rc14-18-ge2a4911",
  "pipeline": "436189414",
  "platform": "rpi64",
  "project": "pantacor/pv-manifest",
  "pvrversion": "pvr version 026-52-gbf3bd5d6",
  "target": "arm-rpi64",
  "time": "2021-12-24 01:25:27 +0000"
}
```

container/src.json

```
{
  "#spec": "service-manifest-src@1",
  "docker_config": {
    "AttachStderr": false,
    "AttachStdin": false,
    "AttachStdout": false,
    "Cmd": [
      "/sbin/init"
    ],
    "Domainname": "",
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Hostname": "",
    "Image": "sha256:701f4bbc1d5b707fc7daabf637cb2fa12532d05ceea6be24f8bb15a55805843b",
    "OpenStdin": false,
    "StdinOnce": false,
    "Tty": false,
    "User": "",
    "WorkingDir": ""
  },
  "docker_digest": "registry.gitlab.com/pantacor/pv-platforms/pv-avahi@sha256:895b2af2b5d407235f2b8c7c568532108a44946898694282340ef0315e2afb28",
  "docker_name": "registry.gitlab.com/pantacor/pv-platforms/pv-avahi",
  "docker_source": "remote,local",
  "docker_tag": "arm32v6",
  "persistence": {},
  "template": "builtin-lxc-docker"
}
```

Pantavisor xconnect

The `pv-xconnect` service mesh facilitates efficient container-to-container and container-to-host interactions. It uses a plugin-driven architecture to inject resources (Unix sockets, D-Bus proxies, DRM nodes) into consumer containers on demand.

For information on how to inspect or manage the service mesh via the Pantavisor Control API, see the Pantavisor Control Socket reference.

ARCHITECTURE

To manage interactions between containers, a dedicated process called `pv-xconnect` handles the mediation logic via on-demand plugins. It runs as a managed daemon spawned by Pantavisor init and is enabled for all init modes (embedded, standalone, and appengine).

Core Responsibilities

- **Discovery & Reconciliation**: Periodically consumes an `xconnect-graph` from Pantavisor's `pv-ctrl` socket and maintains the state of active connects.
- **Plumbing**: Provides namespace-aware helpers to inject virtual resources (sockets/device nodes) inside the consumer's namespace.
- **Security**: Acts as the single point of truth for role-based access control.

SERVICE MANIFESTS

Provider (`services.json`)

A container declares the services it provides in a `services.json` file. This file must use the `#spec` format for identification by Pantavisor's parser.

Example `services.json` :

```
{
  "#spec": "service-manifest-xconnect@1",
  "services": [
    {
      "name": "network-manager",
      "type": "rest",
      "socket": "/run/network-manager/api.sock"
    },
    {
      "name": "system-bus",
      "type": "dbus",
      "socket": "/run/dbus/system_bus_socket"
    }
  ]
}
```

Consumer (`args.json` / `run.json`)

Containers that consume services define their requirements in `args.json` during creation (e.g., with `pvr app add --arg-json args.json`). These are then rendered into the final `run.json` manifest.

Example `run.json` requirement:

```
{
  "#spec": "service-manifest-run@1",
  "name": "my-app",
  "services": {
    "required": [
      {
        "name": "system-bus",
        "type": "dbus",
        "interface": "org.pantavisor.Example",
        "target": "/run/dbus/system_bus_socket"
      }
    ]
  },
  "type": "lxc"
}
```

- **`interface`** : Protocol-specific identifier (e.g., D-Bus interface name).
- **`target`** : The path where `pv-xconnect` should inject the proxied resource inside the consumer container.

MEDIATION PATTERNS

Raw Unix Sockets

Provides direct proxying of Unix Domain Sockets between containers. It supports high-performance features like FD passing (SCM_RIGHTS) and Shared Memory handles.

REST

Identity-injected HTTP over UDS. `pv-xconnect` automatically injects `X-PV-Client` and `X-PV-Role` headers into the first request, allowing the provider to identify the consumer.

D-Bus

Policy-aware proxy for the system bus. It performs **Role-Based Identity Masquerading**: 1. `pv-xconnect` intercepts the D-Bus SASL authentication phase. 2. It takes the **Role** from the link and looks up the corresponding **UID** in the provider container's `/etc/passwd`. 3. It replaces the consumer's identity with the resolved UID.

This allows the provider's standard `dbus-daemon` to enforce fine-grained permissions using standard XML policy files based on the assigned role.

DRM / Graphics

- **Master Role**: Injects `/dev/dri/cardX` for display servers (KMS access).
- **Render Role**: Injects `/dev/dri/renderDX` for accelerated applications.

Wayland

Mediates the Wayland protocol for isolated UI rendering, allowing a containerized compositor to serve multiple isolated clients.

TOOLS

pvcurl

A lightweight shell script wrapping `nc` for HTTP-over-Unix-socket communication. It is preferred in App Engine environments where standard `curl` might not be available.

pvcontrol

A high-level CLI wrapper around `pvcurl` for common Pantavisor control operations.

7.1.2 027

Pantavisor Log Sockets

The Pantavisor logging system uses two Unix sockets for inter-process log management: `pv-ctrl-log` for receiving direct log messages and `pv-fd-log` for subscribing file descriptors to be polled by Pantavisor.

PV-CTRL-LOG

This socket is used by applications and containers to send log messages directly to the Pantavisor Log Server. All messages must follow the `logserver_msg` structure:

```
struct logserver_msg {
    int code; // Protocol code: 0 for LEGACY, 256 for CMD
    int len;  // Length of the following buffer
    char buf[0]; // Log data buffer
};
```

Legacy Protocol (code = 0)

The `buf` contains the log metadata and message, separated by null terminators (`\0`):

```
level\0platform\0source\0data
```

- **level**: Log level as a string (e.g., "3" for INFO).
- **platform**: Name of the container or "pantavisor".
- **source**: The specific source of the log (e.g., a process name or module).
- **data**: The actual log message content.

The supported log levels are: * `0` : FATAL * `1` : ERROR * `2` : WARN * `3` : INFO * `4` : DEBUG * `5` : ALL

PV-FD-LOG

This socket allows containers to delegate the polling of their file descriptors (e.g., pipes, sockets, or open files) to Pantavisor.

Subscription Protocol

To subscribe or unsubscribe a file descriptor, you must use the `sendmsg` system call with `SCM_RIGHTS` to pass the file descriptor.

The `msghdr` must contain an `iovec` array with 4 elements:

iov Index	Type	Max Length	Description
`iov[0]`	string	50 bytes	Platform name (container name)
`iov[1]`	string	50 bytes	Source name (e.g., "stdout", "syslog")
`iov[2]`	int	4 bytes	Log level for the messages from this FD
`iov[3]`	int	4 bytes	Action: `1` to subscribe, `0` to unsubscribe

Subscribe

- Send the file descriptor using `SCM_RIGHTS` .
- Set `iov[3]` to `1` .
- Pantavisor will poll this FD and create a corresponding log file at `/storage/logs/current/<platform>/<source>` .

Unsubscribe

- Set `iov[3]` to `0` .
- The file descriptor passed in `SCM_RIGHTS` can be `-1` .
- Pantavisor will stop polling and close its internal reference to the FD for that platform/source pair.

Note

Only one file descriptor can be subscribed per platform-source pair. Subscribing a new FD for an existing pair will replace the previous one.

Pantavisor Control Socket

The pv-ctrl socket enables communication between the containers and Pantavisor during runtime.

The following subsections describe the behaviour of the HTTP API for the different endpoints, which you can test either using any HTTP client or our pvcontrol tool from the default pvr-sdk container.

> **Note**
>
> The examples provided use cURL, but any HTTP client inside of your container should work.

/CONTAINERS

This endpoint can be used to list and manage containers in the current revision.

List containers

Returns all containers with their status, restart policy, auto-recovery state, and service mesh information.

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/containers"
```

Lifecycle control

Containers with `restart_policy: "container"` can be stopped, started, and restarted via the API. Containers with `restart_policy: "system"` are protected and will return HTTP 403.

To stop a container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"stop\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Stopping a container sets the `user_stopped` flag, which prevents auto-recovery from restarting it. The container's auto-recovery configuration is preserved — no retry counters are consumed and no backoff is triggered.

To start a previously stopped container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"start\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Starting clears the `user_stopped` flag and returns the container to the normal engine lifecycle. Auto-recovery is fully restored.

To restart a running container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"restart\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Restart force-stops the container and resets the retry counter to zero. For containers with auto-recovery configured, the recovery engine restarts it naturally through the standard path. For containers without auto-recovery (`RECOVERY_NO`), the container is restarted directly.

Action	State change	Auto-recovery effect
stop	Running STOPPED	`user_stopped` set; recovery skipped
start	STOPPED INSTALLED start	`user_stopped` cleared; recovery restored
restart (with recovery)	Running STOPPED recovery engine restarts	Retry counter reset to 0
restart (no recovery)	Running STOPPED INSTALLED start	Direct restart

/DAEMONS

This endpoint can be used to list and manage Pantavisor internal daemons.

To list all daemons and their current status:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons"
```

To stop a specific daemon (e.g., `pv-xconnect`):

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"stop\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons/pv-xconnect"
```

To start a specific daemon:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"start\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons/pv-xconnect"
```

/GROUPS

These requests can be used to list groups.

To list all groups in the current revision:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/groups"
```

/SIGNAL

This type of command can be issued to alter the container status in Pantavisor.

To send the one-shot readiness signal:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"type\":\"ready\",\"payload\":\"\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
signal
```

This request will fail if the signal type is not supported, or if that status goal is not expected.

/COMMANDS

These commands can perform changes in Pantavisor container engine itself, so the result will not be inmediate to the request.

An example of a command that tells Pantavisor to transition to revision 4:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"op\":\"LOCAL_RUN\",\"payload\":\"4\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
commands
```

As you can see, the body of the request contains the command itself in JSON format.

These are the different commands that are supported. You can test them by substituting `op` and `payload` in the above command:

op	payload	Description
UPDATE_METADATA	{key, value}	upload the json as a new pair of device metadata to Pantacor Hub
REBOOT_DEVICE	message	reboot device with optional message
POWEROFF_DEVICE	message	poweroff device with optional message
TRY_ONCE	revision	try a revision once (will rollback on failure or next reboot)
LOCAL_RUN	revision	transition to specified revision
MAKE_FACTORY	revision	make the revision the factory revision. If revision is not set, Pantavisor will use the current one. Device needs to be not claimed
RUN_GC	N/A	run garbage collector
ENABLE_SSH	N/A	enable SSH server ignoring config until reboot
DISABLE_SSH	N/A	disable SSH server ignoring config until reboot
GO_REMOTE	N/A	go remote when running on a locals/ revision if allowed by config
DEFER_REBOOT	N/A	defer reboot when debug shell is active
LOCAL_RUN_COMMIT	revision	transition to revision and commit it automatically
LOCAL_APPLY	revision	apply revision changes without a full reboot
XCONNECT_GRAPH	N/A	trigger an immediate xconnect graph reconciliation

/OBJECTS

This endpoint can be used to list, send and receive objects to and from Pantavisor. Bear in mind that objects are the artifacts that form a Pantavisor revision.

An example of listing the objects that are stored in the device:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects"
```

Here, an example for getting one of those objects:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

You can even put new objects in Pantavisor. Notice that the sha256sum of object has to match the specified sha in the URI:

```
curl -X PUT --upload-file object --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

/STEPS

This endpoint can be used to list, send and receive step jsons, as well as get the update progress and set the commit message for any of them.

Let us go with the examples, first, to list all steps installed in the device:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps"
```

To get an existing step json:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

To send a new json, the format of the new revision in the URI has to contain the "locals/" prefix. The name after the prefix must be under 64 characters and must not contain any other "/" character. These revisions that are installed using the socket (locals) are treated in a different way than the ones installed from Pantacor Hub (remotes), as you will have to manually request the transition to locals using the run

command. Most importantly, locals will not attempt any communication with Pantacor Hub during runtime unless a go remote command is issued.

```
curl -X PUT --upload-file json --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/locals/example"
```

To get the update progress (DONE, TESTING, INPROGRESS...) and some related information of a revision:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/progress"
```

Finally, you can set a commit message that will be stored along a revision and showed when listing revisions so the user can idendifcate each one of them:

```
curl -X PUT --data "message" --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/
commitmsg"
```

/USER-META

The user-meta endpoint offers the ability to list, save and delete user metadata.

To list all user meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta"
```

An example of creating or updating a new user metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

/DEVICE-META

The device-meta endpoint offers the ability to list device metadata.

To list all device meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta"
```

An example of creating or updating a new device metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

/XCONNECT-GRAPH

This endpoint returns the current xconnect service mesh graph in JSON format. For details on how the service mesh operates and how to define manifests, see the Pantavisor xconnect reference.

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/xconnect-graph"
```

/BUILDINFO

For debugging porpuses, it is possible to get the repo manifest that was used to build this Pantavisor binary.

With cURL, this would look like:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/buildinfo"
```

/DRIVERS

The drivers endpoint lets you list load and unload managed drivers.

To list drivers referenced by container and their load state:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl http://localhost/drivers
```

To load manual drivers at bulk from within container:

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/load
```

To load individual manual drivers, in this case one driver named "wifi":

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/load
```

Same for unloading, it can be done at bulk:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/unload
```

And individually:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/unload
```

Pantavisor Configuration

> **Note**
>
> This reference page presents the newly unified configuration key syntax. To get to the deprecated but still supported previous format, you will have to go here.

SUMMARY

> **Note**
>
> The key syntax is the same for all configuration levels.

> **Note**
>
> All keys are case insensitive.

> **Note**
>
> Syntax and behavior of keys tagged with (experimental) might change and break backwards compatibility.

This table contains the currently supported list of configuration keys, sorted alphabetically.

Key	Value	Default	Description
`PH_CREDS_HOST`	IP or hostname	`api.pantahub.com`	set Pantacor Hub address
`PH_CREDS_ID`	string	empty	set Pantacor Hub device ID
`PH_CREDS_PORT`	port	`443`	set port for communication Pantacor Hub
`PH_CREDS_PROXY_HOST`	IP or hostname	empty	set Pantacor Hub proxy ad
`PH_CREDS_PROXY_NOPROXYCONNECT`	`0` or `1`	`0`	disable proxy communicat Pantacor Hub
`PH_CREDS_PROXY_PORT`	port	`3218`	set port for proxy commun with Pantacor Hub
`PH_CREDS_PRN`	string	empty	set Pantacor Hub device P
`PH_CREDS_SECRET`	string	empty	set Pantacor Hub credenti
`PH_CREDS_TYPE`	`builtin`	`builtin`	set Pantacor Hub credenti
`PH_FACTORY_AUTOTOK`	token	empty	set factory auto token for communication with Panta
`PH_LIBEVENT_HTTP_RETRIES`	number of retries	`1`	set HTTP request number for communication with Pa Hub
`PH_LIBEVENT_HTTP_TIMEOUT`	time (in seconds)	`60`	set HTTP request timeout communication with Panta
`PH_METADATA_DEVMETA_INTERVAL`	time (in seconds)	`10`	set push interval for device metadata to Pantacor Hub
`PH_METADATA_USRMETA_INTERVAL`	time (in seconds)	`5`	set refresh interval for use metadata from Pantacor H
`PH_ONLINE_REQUEST_THRESHOLD`	number of failures	`0`	number of failed requests Pantacor Hub allowed to s consider device online
`PH_UPDATER_INTERVAL`	time (in seconds)	`60`	set time between Pantacor update requests
`PH_UPDATER_NETWORK_TIMEOUT`	time (in seconds)	`120`	set time before rollback if cannot communicate with Hub
`PH_UPDATER_TRANSFER_MAX_COUNT`	number of transfers	`5`	set maximum number of o transfers to and from Panta during updates
`PV_BOOTLOADER_FITCONFIG`	string	empty	set FIT configuration name
`PV_BOOTLOADER_MTD_ENV`	string	empty	set MTD name for bootloa
`PV_BOOTLOADER_MTD_ONLY`	`0` or `1`	`0`	enable MTD for bootloade
`PV_BOOTLOADER_TYPE`	`uboot`, `uboot-ab`, `uboot-pvk`, `rpiab` or `grub`	`uboot`	set bootloader type
`PV_BOOTLOADER_UBOOTAB_A_NAME`	string	`fitA`	name of the partition to us uboot-ab mode
`PV_BOOTLOADER_UBOOTAB_B_NAME`	string	`fitB`	name of the partition to us uboot-ab mode

Key	Value	Default	Description
`PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME`	string	empty	name of the partition wher uboot environment is back
`PV_BOOTLOADER_UBOOTAB_ENV_NAME`	string	empty	name of the partition wher uboot environment is store
`PV_BOOTLOADER_UBOOTAB_ENV_OFFSET`	offset in bytes	`0`	environment offset from th beginning of the partition
`PV_BOOTLOADER_UBOOTAB_ENV_SIZE`	size in bytes	`0`	size of the uboot environm
`PV_CACHE_DEVMETADIR`	path	`/storage/cache/devmeta`	set persistent device meta
`PV_CACHE_USRMETADIR`	path	`/storage/cache/meta`	set persistent user metada
`PV_CONTROL_REMOTE`	`0` or `1`	`1`	allow remote control from Hub
`PV_CONTROL_REMOTE_ALWAYS`	`0` or `1`	`0`	keep communication with Hub even when a local rev running
`PV_DEBUG_SHELL`	`0` or `1`	`1`	enable local debug shell
`PV_DEBUG_SHELL_AUTOLOGIN`	`0` or `1`	`0`	enable autologin for debug
`PV_DEBUG_SHELL_TIMEOUT`	time (in seconds)	`60`	time that Pantavisor waits rebooting if debug shell co opened
`PV_DEBUG_SSH`	`0` or `1`	`1`	enable SSH debug access
`PV_DEBUG_SSH_AUTHORIZED_KEYS`	string	empty	set authorized keys for SS access
`PV_DISK_EXPORTSDIR`	path	`/exports`	set exports directory
`PV_DISK_VOLDIR`	path	`/volumes`	set volumes directory
`PV_DISK_WRITABLEDIR`	path	`/writable`	set writable directory
`PV_DROPBEAR_CACHE_DIR`	path	`/storage/cache/dropbear`	set debug ssh server cach directory
`PV_LIBEVENT_DEBUG_MODE`	`0` or `1`	`0`	enable event loop debug lo
`PV_LIBTHTTP_CERTSDIR`	path	`/certs`	set certificates directory fo
`PV_LIBTHTTP_LOG_LEVEL`	`0` to `5`	`3`	set libthttp log verbosity le
`PV_LOG_BUF_NITEMS`	integer	`128`	set in-memory logs buffer
`PV_LOG_CAPTURE`	`0` or `1`	`1`	capture logs from containe
`PV_LOG_CAPTURE_DMESG`	`0` or `1`	`1`	capture dmesg logs
`PV_LOG_DIR`	path	`/storage/logs/`	set logs directory
`PV_LOG_DIR_MAXSIZE`	integer with optional suffix `B` (default), `K`, `KB`, `M`, `MB`, `G`, `GB`, `T`, `TB`, `%`; `0` for auto 10% (100% if tmpfs)	`16777216`	max size of log directory
`PV_LOG_FILETREE_TIMESTAMP_FORMAT`	format string	empty	timestamp format for filetr

Key	Value	Default	Description
`PV_LOG_HYSTERESIS_FACTOR`	positive integer	`4`	controls the gap between l low watermarks for log dir cleanup
`PV_LOG_LEVEL`	`0` to `5`	`0`	set Pantavisor log level (0: 5: ALL)
`PV_LOG_LOGGERS`	`0` or `1`	`1`	enable loggers for contain
`PV_LOG_PUSH`	`0` or `1`	`1`	push logs to Pantacor Hub
`PV_LOG_ROTATE_FACTOR`	integer	`5`	determines per-file rotatio threshold for log directory
`PV_LOG_SERVER_OUTPUTS`	string	`filetree`	set log server outputs (con separated)
`PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT`	format string	empty	timestamp format for singl
`PV_LOG_STDOUT_TIMESTAMP_FORMAT`	format string	empty	timestamp format for stdou
`PV_LXC_LOG_LEVEL`	`0` to `5`	`2`	set LXC log level
`PV_NET_BRADDRESS4`	IP address	`10.0.3.1`	set bridge IPv4 address
`PV_NET_BRDEV`	interface name	`lxcbr0`	set bridge device name
`PV_NET_BRMASK4`	IP mask	`255.255.255.0`	set bridge IPv4 mask
`PV_OEM_NAME`	string	empty	set OEM name for configu overrides
`PV_POLICY`	string	empty	set policy name for config
`PV_REMOUNT_POLICY`	string	empty	set remount policy name f filesystem remounting
`PV_REVISION_RETRIES`	integer	`10`	number of retries for revisi transitions
`PV_SECUREBOOT_CHECKSUM`	`0` or `1`	`1`	enable artifact checksum v
`PV_SECUREBOOT_HANDLERS`	`0` or `1`	`1`	enable handlers verificatio
`PV_SECUREBOOT_MODE`	`disabled`, `audit`, `lenient` or `strict`	`lenient`	set secureboot mode
`PV_SECUREBOOT_OEM_TRUSTSTORE`	path	`/etc/pantavisor/certs/oem`	set path to OEM truststore
`PV_SECUREBOOT_TRUSTSTORE`	path	`/etc/pantavisor/certs`	set path to Pantavisor trus
`PV_STORAGE_DEVICE`	string	empty	set storage device name
`PV_STORAGE_FSTYPE`	string	empty	set storage filesystem type
`PV_STORAGE_GC_KEEP_FACTORY`	`0` or `1`	`0`	keep factory revision durin
`PV_STORAGE_GC_RESERVED`	percentage	`5`	reserved storage percenta
`PV_STORAGE_GC_THRESHOLD`	percentage	`0`	storage GC threshold perc
`PV_STORAGE_GC_THRESHOLD_DEFERTIME`	time (in seconds)	`600`	defer time for GC threshol
`PV_STORAGE_LOGTEMPSIZE`	size string	empty	set size for temporary log

Key	Value	Default	Description
`PV_STORAGE_MNTPOINT`	path	empty	set storage mount point
`PV_STORAGE_MNTTYPE`	string	empty	set storage mount type
`PV_STORAGE_PHCONFIG_VOL`	`0` or `1`	`0`	use volume for Pantahub configuration
`PV_STORAGE_WAIT`	time (in seconds)	`5`	time to wait for storage de
`PV_SYSCTL_*`	string	—	set any kernel sysctl at run maps to `/proc/sys/` (e.g. `PV_SYSCTL_KERNEL_CORE_PA` `proc/sys/kernel/core_pat`
`PV_SYSCTL_KERNEL_CORE_PATTERN`	string	`|/lib/pv/pvcrash --skip`	set kernel core dump patte
`PV_SYSTEM_APPARMOR_PROFILES`	string	empty	AppArmor profiles to load
`PV_SYSTEM_CONFDIR`	path	`/configs`	set directory for system configurations
`PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO`	`0` or `1`	`0`	enable early auto-loading
`PV_SYSTEM_ETCDIR`	path	`/etc`	set system etc directory
`PV_SYSTEM_ETCPANTAVISORDIR`	path	`/etc/pantavisor`	set Pantavisor etc director
`PV_SYSTEM_INIT_MODE`	`embedded`, `standalone` or `appengine`	`embedded`	set system init mode
`PV_SYSTEM_LIBDIR`	path	`/lib`	set system library directory
`PV_SYSTEM_MEDIADIR`	path	`/media`	set system media directory
`PV_SYSTEM_MOUNT_SECURITYFS`	`0` or `1`	`0`	mount securityfs
`PV_SYSTEM_RUNDIR`	path	`/run/pantavisor/pv`	set system run directory
`PV_SYSTEM_USRDIR`	path	`/usr`	set system usr directory
`PV_UPDATER_COMMIT_DELAY`	time (in seconds)	`25`	delay before committing a
`PV_UPDATER_GOALS_TIMEOUT`	time (in seconds)	`120`	timeout for reaching updat
`PV_UPDATER_USE_TMP_OBJECTS`	`0` or `1`	`0`	use temporary objects dur updates
`PV_VOLMOUNT_DM_EXTRA_ARGS`	string	empty	extra arguments for DM vo mounting
`PV_WDT_MODE`	`disabled`, `shutdown`, `startup` or `always`	`shutdown`	set watchdog mode
`PV_WDT_TIMEOUT`	time (in seconds)	`15`	set watchdog timeout

LEVELS

This table shows the configuration levels that are allowed for each configuration key.

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PH_CREDS_HOST	✗	✓	✓	✓	✓
PH_CREDS_ID	✗	✓	✓	✓	✓
PH_CREDS_PORT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_HOST	✗	✓	✓	✓	✓
PH_CREDS_PROXY_NOPROXYCONNECT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_PORT	✗	✓	✓	✓	✓
PH_CREDS_PRN	✗	✓	✓	✓	✓
PH_CREDS_SECRET	✗	✓	✓	✓	✓
PH_CREDS_TYPE	✗	✓	✓	✓	✓
PH_FACTORY_AUTOTOK	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_TIMEOUT	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_RETRIES	✗	✓	✓	✓	✓
PH_METADATA_DEVMETA_INTERVAL	✗	✓	✓	✓	✓
PH_METADATA_USRMETA_INTERVAL	✗	✓	✓	✓	✓
PH_ONLINE_REQUEST_THRESHOLD	✗	✓	✓	✓	✓
PH_UPDATER_INTERVAL	✗	✓	✓	✓	✓
PH_UPDATER_NETWORK_TIMEOUT	✗	✓	✓	✓	✓
PH_UPDATER_TRANSFER_MAX_COUNT	✗	✓	✓	✓	✓
PV_BOOTLOADER_FITCONFIG	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ENV	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ONLY	✓	✗	✓	✓	✗
PV_BOOTLOADER_TYPE	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_A_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_B_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_OFFSET	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_SIZE	✓	✗	✓	✓	✗
PV_CACHE_DEVMETADIR	✓	✗	✓	✓	✗
PV_CACHE_USRMETADIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_CONTROL_REMOTE	✓	✗	✓	✓	✓
PV_CONTROL_REMOTE_ALWAYS	✓	✗	✓	✓	✓
PV_DEBUG_SHELL	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_AUTOLOGIN	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_TIMEOUT	✓	✗	✓	✓	✗
PV_DEBUG_SSH	✓	✗	✓	✓	✓
PV_DEBUG_SSH_AUTHORIZED_KEYS	✓	✗	✓	✓	✓
PV_DISK_EXPORTSDIR	✓	✗	✓	✓	✗
PV_DISK_VOLDIR	✓	✗	✓	✓	✗
PV_DISK_WRITABLEDIR	✓	✗	✓	✓	✗
PV_DROPBEAR_CACHE_DIR	✓	✗	✓	✓	✗
PV_LIBEVENT_DEBUG_MODE	✓	✗	✓	✓	✓
PV_LIBTHTTP_CERTSDIR	✓	✗	✓	✓	✗
PV_LIBTHTTP_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_BUF_NITEMS	✓	✗	✓	✓	✓
PV_LOG_CAPTURE	✓	✗	✓	✓	✓
PV_LOG_CAPTURE_DMESG	✓	✗	✓	✓	✓
PV_LOG_DIR	✓	✗	✓	✓	✗
PV_LOG_DIR_MAXSIZE	✓	✗	✓	✓	✓
PV_LOG_FILETREE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_HYSTERESIS_FACTOR	✓	✗	✓	✓	✓
PV_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_LOGGERS	✓	✗	✓	✓	✓
PV_LOG_PUSH	✓	✗	✓	✓	✓
PV_LOG_ROTATE_FACTOR	✓	✗	✓	✓	✓
PV_LOG_SERVER_OUTPUTS	✓	✗	✓	✓	✓
PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_STDOUT_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LXC_LOG_LEVEL	✓	✗	✓	✓	✓

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PV_NET_BRADDRESS4	✓	✗	✓	✓	✓
PV_NET_BRDEV	✓	✗	✓	✓	✓
PV_NET_BRMASK4	✓	✗	✓	✓	✓
PV_OEM_NAME	✓	✗	✓	✓	✗
PV_POLICY	✓	✗	✓	✗	✗
PV_REMOUNT_POLICY	✗	✗	✓	✗	✗
PV_REVISION_RETRIES	✓	✗	✓	✓	✓
PV_SECUREBOOT_CHECKSUM	✓	✗	✓	✓	✗
PV_SECUREBOOT_HANDLERS	✓	✗	✓	✓	✗
PV_SECUREBOOT_MODE	✓	✗	✓	✓	✗
PV_SECUREBOOT_OEM_TRUSTSTORE	✓	✗	✓	✓	✗
PV_SECUREBOOT_TRUSTSTORE	✓	✗	✓	✓	✗
PV_STORAGE_DEVICE	✓	✗	✓	✓	✗
PV_STORAGE_FSTYPE	✓	✗	✓	✓	✗
PV_STORAGE_GC_KEEP_FACTORY	✓	✗	✓	✓	✓
PV_STORAGE_GC_RESERVED	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD_DEFERTIME	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD	✓	✗	✓	✓	✓
PV_STORAGE_LOGTEMPSIZE	✓	✗	✓	✓	✗
PV_STORAGE_MNTPOINT	✓	✗	✓	✓	✗
PV_STORAGE_MNTTYPE	✓	✗	✓	✓	✗
PV_STORAGE_WAIT	✓	✗	✓	✓	✗
PV_SYSCTL_*	✓	✗	✓	✓	✓
PV_SYSCTL_KERNEL_CORE_PATTERN	✓	✗	✓	✓	✓
PV_SYSTEM_APPARMOR_PROFILES	✓	✗	✓	✓	✗
PV_SYSTEM_CONFDIR	✓	✗	✓	✓	✗
PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO	✓	✗	✓	✓	✗
PV_SYSTEM_ETCDIR	✓	✗	✓	✓	✗
PV_SYSTEM_INIT_MODE	✓	✗	✓	✓	✗
PV_SYSTEM_LIBDIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_SYSTEM_MEDIADIR	✓	✗	✓	✓	✗
PV_SYSTEM_MOUNT_SECURITYFS	✓	✗	✓	✓	✗
PV_SYSTEM_RUNDIR	✓	✗	✓	✓	✗
PV_SYSTEM_USRDIR	✓	✗	✓	✓	✗
PV_UPDATER_COMMIT_DELAY	✓	✗	✓	✓	✓
PV_UPDATER_GOALS_TIMEOUT	✓	✗	✓	✓	✓
PV_UPDATER_USE_TMP_OBJECTS	✓	✗	✓	✓	✓
PV_VOLMOUNT_DM_EXTRA_ARGS	✓	✗	✓	✓	✓
PV_WDT_MODE	✓	✗	✓	✓	✗
PV_WDT_TIMEOUT	✓	✓	✓	✓	✗

Pantavisor Metadata

This page contains reference information about Pantavisor metadata.

DEVICE METADATA

This is the device metadata created by Pantavisor that will give you useful information about your device:

Key	Value	Description
`interfaces`	json	network interfaces of the device
`pantahub.address`	IP:port	Pantacor Hub address the client is communicating with
`pantahub.claimed`	0 or 1	1 if claimed in Pantacor Hub
`pantahub.online`	0 or 1	1 if connection to Pantacor Hub was established
`pantahub.state`	string	see Pantacor Hub states (init, register, claim, sync, login, wait hub, report, idle, prep download or download)
`pantavisor.arch`	string	CPU architecture
`pantavisor.claimed`	0 or 1	1 if device has ever been claimed (local or remote)
`pantavisor.cpumodel`	string	CPU model name
`pantavisor.dtmodel`	string	Device Tree model name
`pantavisor.mode`	local or remote	see operation modes
`pantavisor.revision`	string	revision number
`pantavisor.status`	string	revision status
`pantavisor.uname`	json	uname output
`pantavisor.version`	string	Pantavisor build version
`storage`	json	disk usage of the device
`sysinfo`	json	sysinfo
`time`	json	time information

User metadata

This is the user metadata that can be set by the user which is parsed and have some actions on Pantavisor:

Key	Value	Description
`pvr-sdk.authorized_keys`	SSH pub key	set public key to get SSH access
`pvr-auto-follow.url`	URL	device will automatically pull every change in the device associated to that clone URL
`pantahub.log.push`	0 or 1	disable/enable log pushing to Pantacor Hub. Overrides PV_LOG_PUSH
`<config-key>`	config-value	override any configuration keys that allow RUN level
`<container>/<key>`	value	send user metadata that can be consumed by one of the containers

Pantavisor State Format (state.json)

A Pantavisor revision is defined by a single JSON object called `state.json`. It acts as a **virtual filesystem manifest** where every key represents a relative file path within the revision, and every value is either a nested configuration object or a SHA256 identifier for a binary artifact.

1. ROOT LEVEL (`state.json`)

These keys represent the files at the root of a revision.

Key	Value Type	Mandatory	Description
`#spec`	string	Yes	Parser version. Must be `"pantavisor-service-system@1"`.
`README.md`	string	No	Documentation for the revision in Markdown format.
`bsp/run.json`	BSP Manifest	Yes	Board Support Package configuration.
`bsp/drivers.json`	Drivers Manifest	No	Abstract driver mapping for the kernel.
`device.json`	Infrastructure Manifest	No	Unified physical storage and logical group definition.
`groups.json`	Groups Manifest	No	(Legacy) Logical container orchestration groups.
`disks.json`	Disks Manifest	No	(Legacy) Physical storage medium definitions.
`<container>/run.json`	Container Manifest	Yes	Individual container configuration.
`<container>/services.json`	Service Exports	No	Services exported to the xconnect mesh.
`_sigs/<container>.json`	Signature Manifest	No	Security signature for container artifacts.
`_config/<container>/<path>`	string	No	Injects data into `<path>` inside the container's rootfs.
`<any/other/path>`	string	No	SHA256 identifier for a binary artifact at that path.

2. BSP (`bsp/run.json`)

Defines the core system boot assets.

Key	Value Type	Description
`linux`	path string	Path to the Linux kernel image.
`initrd`	path string	Path to the Pantavisor initrd binary.
`modules`	path string	Path to the modules squashfs image.
`firmware`	path string	Path to the firmware squashfs image.
`fdt`	path string	Path to the Flattened Device Tree binary.
`fit`	path string	Path to a FIT image (replaces linux/initrd/fdt).
`rpiab`	path string	Path to a Raspberry Pi boot image.
`addons`	array of paths	List of CPIO addons to merge into the initrd rootfs.
`initrd_config`	path string	Custom configuration for the initrd process.

3. DRIVERS (`bsp/drivers.json`)

Maps abstract driver names to kernel modules based on hardware.

Key	Value Type	Description
`#spec`	string	Must be `"driver-aliases@1"`.
`all`	object	Default module mappings for all hardware.
`dtb:<name>`	object	Module mappings specific to a Device Tree model.
`ovl:<name>`	object	Module mappings specific to a DT Overlay.

Example module list:

```
"wifi": [ "cfg80211", "brcmfmac ${user-meta:wifi.opts}" ]
```

---

4. INFRASTRUCTURE (`device.json`)

The unified hardware and orchestration manifest.

Key	Value Type	Description
`disks`	array	List of Disk Definitions.
`groups`	array	List of Orchestration Groups.
`volumes`	object	List of Persistent Volumes for Pantavisor itself.

---

5. ORCHESTRATION (`groups.json`)

Defines how containers are grouped and started.

Key	Value Type	Default	Description
`name`	string	**Mandatory**	Unique logical name for the group.
`description`	string	empty	Human-readable description.
`status_goal`	enum	`STARTED`	Goal for all members: `MOUNTED`, `STARTED`, `READY`.
`restart_policy`	enum	`container`	Policy on failure: `system`, `container`.
`timeout`	integer	30	Seconds to wait for members to reach `status_goal`.
`auto_recovery`	object	none	Default auto-recovery for containers in this group. Inherited all-or-nothing by containers without their own `auto_recovery`.

---

6. STORAGE (`disks.json`)

Defines physical storage mediums.

Key	Value Type	Default	Description
`name`	string	**Mandatory**	Unique name used in `run.json` storage keys.
`type`	enum	**Mandatory**	`directory`, `dm-crypt-versatile`, `swap-disk`, `volume-disk`.
`path`	string	**Mandatory**	Path to block device or image file.
`format`	enum	`ext4`	Filesystem format (`ext3`, `ext4`, `swap`).
`provision`	string	empty	Provisioning source (e.g., `zram`).
`mount_target`	path	empty	Where to mount the disk on the host.
`mount_options`	string	empty	Comma-separated mount flags.
`format_options`	string	empty	Arguments for the `mkfs` command.
`default`	string	`"no"`	If `"yes"`, this disk is used for all volumes without a `disk` key.

---

7. CONTAINER (`<container>/run.json`)

Configures an individual container runtime.

Key	Value Type	Mandatory	Description
`#spec`	string	Yes	Must be `"service-manifest-run@1"`.
`name`	string	Yes	Logical name of the container.
`type`	enum	Yes	Runtime type (currently only `lxc`).
`config`	path string	Yes	Path to the LXC configuration file.
`root-volume`	path string	Yes	Path to the rootfs squashfs artifact.
`volumes`	array	No	Additional artifacts to mount as volumes.
`group`	string	No	Orchestration group name (from `device.json`).
`status_goal`	enum	No	Target state: `MOUNTED`, `STARTED`, `READY`.
`restart_policy`	enum	No	`system` (reboot on crash) or `container` (restart LXC).
`roles`	array	No	Capability roles: `mgmt` (control API access) or `nobody`.
`storage`	object	Yes	Persistence settings for rootfs paths.
`drivers`	object	No	Requirements: `required`, `optional`, or `manual`.
`services`	object	No	Service mesh requirements.
`logs`	array	No	Logger configurations.
`exports`	array	No	(Boolean flag in code) Marks container as an exporter.
`auto_recovery`	object	No	Auto-recovery configuration. If absent, inherited from group.

Auto-Recovery Object

Configures automatic restart behavior when a container crashes. See Auto-Recovery overview for the broader context.

Key	Value Type	Default	Description
`policy`	enum	`no`	Recovery policy: `no`, `always`, `on-failure`, `unless-stopped`. Note: the current implementation does not distinguish exit codes — `on-failure` behaves the same as `always`.
`max_retries`	integer	0	Maximum restart attempts. 0 = unlimited.
`retry_delay`	integer	0	Initial delay in seconds before first restart.
`backoff_factor`	number	1.0	Multiplier applied to `retry_delay` on each subsequent retry.
`reset_window`	integer	0	Seconds of continuous uptime after which the retry counter resets to 0.
`stable_timeout`	integer	0	Seconds the container must survive after reaching its status goal to be considered stable. Used to gate TESTING commit.
`backoff_policy`	string	`reboot`	Action after `max_retries` exhausted in steady state: `reboot`, `never`, or a duration string (`10min`, `1h`, `30s`).

Storage Object

Defines persistence for specific directories. Keys are paths relative to container root. * **`persistence`**: `permanent` (survives updates), `revision` (survives reboots), `boot` (volatile). * **`disk`**: Logical disk name from `device.json`.

Service Requirements

Under the `services` key in `run.json`. * **`required`** / **`optional`**: Arrays of service requirement objects. * `name`: Logical name of the service to find. * `type`: Protocol (`rest`, `dbus`, `unix`, `drm`, `wayland`). * `target`: Path where Pantavisor should inject the socket/resource. * `role`: Masquerade as this role when connecting. * `interface`: Protocol-specific identifier.

---

8. SERVICE MESH (`<container>/services.json`)

Declares services this container provides to others.

Key	Value Type	Description
`#spec`	string	Must be `"service-manifest-xconnect@1"`.
`services`	array	List of service objects (`name`, `type`, `socket`).

---

9. SECURITY (`_sigs/<container>.json`)

JWS-based artifact verification.

Key	Value	Description
`#spec`	`"pvs@2"`	Parser version.
`protected`	base64 string	Encoded headers including `alg`, `typ`, and `pvs` path filters.
`signature`	base64 string	The cryptographic signature of the protected header and payload.
`x5c`	array	(In protected) Certificate chain for verification.
`jwk`	object	(In protected) JSON Web Key for verification.

Pantavisor xconnect

The `pv-xconnect` service mesh facilitates efficient container-to-container and container-to-host interactions. It uses a plugin-driven architecture to inject resources (Unix sockets, D-Bus proxies, DRM nodes) into consumer containers on demand.

For information on how to inspect or manage the service mesh via the Pantavisor Control API, see the Pantavisor Control Socket reference.

ARCHITECTURE

To manage interactions between containers, a dedicated process called `pv-xconnect` handles the mediation logic via on-demand plugins. It runs as a managed daemon spawned by Pantavisor init and is enabled for all init modes (embedded, standalone, and appengine).

Core Responsibilities

- **Discovery & Reconciliation**: Periodically consumes an `xconnect-graph` from Pantavisor's `pv-ctrl` socket and maintains the state of active connects.
- **Plumbing**: Provides namespace-aware helpers to inject virtual resources (sockets/device nodes) inside the consumer's namespace.
- **Security**: Acts as the single point of truth for role-based access control.

SERVICE MANIFESTS

Provider (`services.json`)

A container declares the services it provides in a `services.json` file. This file must use the `#spec` format for identification by Pantavisor's parser.

Example `services.json` :

```
{
  "#spec": "service-manifest-xconnect@1",
  "services": [
    {
      "name": "network-manager",
      "type": "rest",
      "socket": "/run/network-manager/api.sock"
    },
    {
      "name": "system-bus",
      "type": "dbus",
      "socket": "/run/dbus/system_bus_socket"
    }
  ]
}
```

Consumer (`args.json` / `run.json`)

Containers that consume services define their requirements in `args.json` during creation (e.g., with `pvr app add --arg-json args.json`). These are then rendered into the final `run.json` manifest.

Example `run.json` requirement:

```
{
  "#spec": "service-manifest-run@1",
  "name": "my-app",
  "services": {
    "required": [
      {
        "name": "system-bus",
        "type": "dbus",
        "interface": "org.pantavisor.Example",
        "target": "/run/dbus/system_bus_socket"
      }
    ]
  },
  "type": "lxc"
}
```

- **`interface`** : Protocol-specific identifier (e.g., D-Bus interface name).
- **`target`** : The path where `pv-xconnect` should inject the proxied resource inside the consumer container.

MEDIATION PATTERNS

Raw Unix Sockets

Provides direct proxying of Unix Domain Sockets between containers. It supports high-performance features like FD passing (SCM_RIGHTS) and Shared Memory handles.

REST

Identity-injected HTTP over UDS. `pv-xconnect` automatically injects `X-PV-Client` and `X-PV-Role` headers into the first request, allowing the provider to identify the consumer.

D-Bus

Policy-aware proxy for the system bus. It performs **Role-Based Identity Masquerading**: 1. `pv-xconnect` intercepts the D-Bus SASL authentication phase. 2. It takes the **Role** from the link and looks up the corresponding **UID** in the provider container's `/etc/passwd`. 3. It replaces the consumer's identity with the resolved UID.

This allows the provider's standard `dbus-daemon` to enforce fine-grained permissions using standard XML policy files based on the assigned role.

DRM / Graphics

- **Master Role**: Injects `/dev/dri/cardX` for display servers (KMS access).
- **Render Role**: Injects `/dev/dri/renderDX` for accelerated applications.

Wayland

Mediates the Wayland protocol for isolated UI rendering, allowing a containerized compositor to serve multiple isolated clients.

TOOLS

pvcurl

A lightweight shell script wrapping `nc` for HTTP-over-Unix-socket communication. It is preferred in App Engine environments where standard `curl` might not be available.

pvcontrol

A high-level CLI wrapper around `pvcurl` for common Pantavisor control operations.

7.1.3 029

Pantavisor Log Sockets

The Pantavisor logging system uses two Unix sockets for inter-process log management: `pv-ctrl-log` for receiving direct log messages and `pv-fd-log` for subscribing file descriptors to be polled by Pantavisor.

PV-CTRL-LOG

This socket is used by applications and containers to send log messages directly to the Pantavisor Log Server. All messages must follow the `logserver_msg` structure:

```
struct logserver_msg {
    int code; // Protocol code: 0 for LEGACY, 256 for CMD
    int len;  // Length of the following buffer
    char buf[0]; // Log data buffer
};
```

Legacy Protocol (code = 0)

The `buf` contains the log metadata and message, separated by null terminators (`\0`):

```
level\0platform\0source\0data
```

- **level**: Log level as a string (e.g., "3" for INFO).
- **platform**: Name of the container or "pantavisor".
- **source**: The specific source of the log (e.g., a process name or module).
- **data**: The actual log message content.

The supported log levels are: * `0` : FATAL * `1` : ERROR * `2` : WARN * `3` : INFO * `4` : DEBUG * `5` : ALL

PV-FD-LOG

This socket allows containers to delegate the polling of their file descriptors (e.g., pipes, sockets, or open files) to Pantavisor.

Subscription Protocol

To subscribe or unsubscribe a file descriptor, you must use the `sendmsg` system call with `SCM_RIGHTS` to pass the file descriptor.

The `msghdr` must contain an `iovec` array with 4 elements:

iov Index	Type	Max Length	Description
`iov[0]`	string	50 bytes	Platform name (container name)
`iov[1]`	string	50 bytes	Source name (e.g., "stdout", "syslog")
`iov[2]`	int	4 bytes	Log level for the messages from this FD
`iov[3]`	int	4 bytes	Action: `1` to subscribe, `0` to unsubscribe

Subscribe

- Send the file descriptor using `SCM_RIGHTS` .
- Set `iov[3]` to `1` .
- Pantavisor will poll this FD and create a corresponding log file at `/storage/logs/current/<platform>/<source>` .

Unsubscribe

- Set `iov[3]` to `0` .
- The file descriptor passed in `SCM_RIGHTS` can be `-1` .
- Pantavisor will stop polling and close its internal reference to the FD for that platform/source pair.

Note

Only one file descriptor can be subscribed per platform-source pair. Subscribing a new FD for an existing pair will replace the previous one.

Pantavisor Control Socket

The pv-ctrl socket enables communication between the containers and Pantavisor during runtime.

The following subsections describe the behaviour of the HTTP API for the different endpoints, which you can test either using any HTTP client or our pvcontrol tool from the default pvr-sdk container.

> **Note**
>
> The examples provided use cURL, but any HTTP client inside of your container should work.

/CONTAINERS

This endpoint can be used to list and manage containers in the current revision.

List containers

Returns all containers with their status, restart policy, auto-recovery state, and service mesh information.

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/containers"
```

Lifecycle control

Containers with `restart_policy: "container"` can be stopped, started, and restarted via the API. Containers with `restart_policy: "system"` are protected and will return HTTP 403.

To stop a container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"stop\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Stopping a container sets the `user_stopped` flag, which prevents auto-recovery from restarting it. The container's auto-recovery configuration is preserved — no retry counters are consumed and no backoff is triggered.

To start a previously stopped container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"start\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Starting clears the `user_stopped` flag and returns the container to the normal engine lifecycle. Auto-recovery is fully restored.

To restart a running container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"restart\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Restart force-stops the container and resets the retry counter to zero. For containers with auto-recovery configured, the recovery engine restarts it naturally through the standard path. For containers without auto-recovery (`RECOVERY_NO`), the container is restarted directly.

Action	State change	Auto-recovery effect
stop	Running STOPPED	`user_stopped` set; recovery skipped
start	STOPPED INSTALLED start	`user_stopped` cleared; recovery restored
restart (with recovery)	Running STOPPED recovery engine restarts	Retry counter reset to 0
restart (no recovery)	Running STOPPED INSTALLED start	Direct restart

/DAEMONS

This endpoint can be used to list and manage Pantavisor internal daemons.

To list all daemons and their current status:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons"
```

To stop a specific daemon (e.g., `pv-xconnect`):

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"stop\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons/pv-xconnect"
```

To start a specific daemon:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"start\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons/pv-xconnect"
```

/GROUPS

These requests can be used to list groups.

To list all groups in the current revision:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/groups"
```

/SIGNAL

This type of command can be issued to alter the container status in Pantavisor.

To send the one-shot readiness signal:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"type\":\"ready\",\"payload\":\"\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
signal
```

This request will fail if the signal type is not supported, or if that status goal is not expected.

/COMMANDS

These commands can perform changes in Pantavisor container engine itself, so the result will not be inmediate to the request.

An example of a command that tells Pantavisor to transition to revision 4:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"op\":\"LOCAL_RUN\",\"payload\":\"4\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
commands
```

As you can see, the body of the request contains the command itself in JSON format.

These are the different commands that are supported. You can test them by substituting `op` and `payload` in the above command:

op	payload	Description
UPDATE_METADATA	{key, value}	upload the json as a new pair of device metadata to Pantacor Hub
REBOOT_DEVICE	message	reboot device with optional message
POWEROFF_DEVICE	message	poweroff device with optional message
TRY_ONCE	revision	try a revision once (will rollback on failure or next reboot)
LOCAL_RUN	revision	transition to specified revision
MAKE_FACTORY	revision	make the revision the factory revision. If revision is not set, Pantavisor will use the current one. Device needs to be not claimed
RUN_GC	N/A	run garbage collector
ENABLE_SSH	N/A	enable SSH server ignoring config until reboot
DISABLE_SSH	N/A	disable SSH server ignoring config until reboot
GO_REMOTE	N/A	go remote when running on a locals/ revision if allowed by config
DEFER_REBOOT	N/A	defer reboot when debug shell is active
LOCAL_RUN_COMMIT	revision	transition to revision and commit it automatically
LOCAL_APPLY	revision	apply revision changes without a full reboot
XCONNECT_GRAPH	N/A	trigger an immediate xconnect graph reconciliation

/OBJECTS

This endpoint can be used to list, send and receive objects to and from Pantavisor. Bear in mind that objects are the artifacts that form a Pantavisor revision.

An example of listing the objects that are stored in the device:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects"
```

Here, an example for getting one of those objects:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

You can even put new objects in Pantavisor. Notice that the sha256sum of object has to match the specified sha in the URI:

```
curl -X PUT --upload-file object --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

/STEPS

This endpoint can be used to list, send and receive step jsons, as well as get the update progress and set the commit message for any of them.

Let us go with the examples, first, to list all steps installed in the device:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps"
```

To get an existing step json:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

To send a new json, the format of the new revision in the URI has to contain the "locals/" prefix. The name after the prefix must be under 64 characters and must not contain any other "/" character. These revisions that are installed using the socket (locals) are treated in a different way than the ones installed from Pantacor Hub (remotes), as you will have to manually request the transition to locals using the run

command. Most importantly, locals will not attempt any communication with Pantacor Hub during runtime unless a go remote command is issued.

```
curl -X PUT --upload-file json --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/locals/example"
```

To get the update progress (DONE, TESTING, INPROGRESS...) and some related information of a revision:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/progress"
```

Finally, you can set a commit message that will be stored along a revision and showed when listing revisions so the user can idendifcate each one of them:

```
curl -X PUT --data "message" --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/
commitmsg"
```

/USER-META

The user-meta endpoint offers the ability to list, save and delete user metadata.

To list all user meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta"
```

An example of creating or updating a new user metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

/DEVICE-META

The device-meta endpoint offers the ability to list device metadata.

To list all device meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta"
```

An example of creating or updating a new device metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

/XCONNECT-GRAPH

This endpoint returns the current xconnect service mesh graph in JSON format. For details on how the service mesh operates and how to define manifests, see the Pantavisor xconnect reference.

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/xconnect-graph"
```

/BUILDINFO

For debugging porpuses, it is possible to get the repo manifest that was used to build this Pantavisor binary.

With cURL, this would look like:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/buildinfo"
```

/DRIVERS

The drivers endpoint lets you list load and unload managed drivers.

To list drivers referenced by container and their load state:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl http://localhost/drivers
```

To load manual drivers at bulk from within container:

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/load
```

To load individual manual drivers, in this case one driver named "wifi":

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/load
```

Same for unloading, it can be done at bulk:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/unload
```

And individually:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/unload
```

Pantavisor Configuration

> **Note**
>
> This reference page presents the newly unified configuration key syntax. To get to the deprecated but still supported previous format, you will have to go here.

SUMMARY

> **Note**
>
> The key syntax is the same for all configuration levels.

> **Note**
>
> All keys are case insensitive.

> **Note**
>
> Syntax and behavior of keys tagged with (experimental) might change and break backwards compatibility.

This table contains the currently supported list of configuration keys, sorted alphabetically.

Key	Value	Default	Description
`PH_CREDS_HOST`	IP or hostname	`api.pantahub.com`	set Pantacor Hub address
`PH_CREDS_ID`	string	empty	set Pantacor Hub device ID
`PH_CREDS_PORT`	port	`443`	set port for communication Pantacor Hub
`PH_CREDS_PROXY_HOST`	IP or hostname	empty	set Pantacor Hub proxy ad
`PH_CREDS_PROXY_NOPROXYCONNECT`	`0` or `1`	`0`	disable proxy communicat Pantacor Hub
`PH_CREDS_PROXY_PORT`	port	`3218`	set port for proxy commun with Pantacor Hub
`PH_CREDS_PRN`	string	empty	set Pantacor Hub device P
`PH_CREDS_SECRET`	string	empty	set Pantacor Hub credenti
`PH_CREDS_TYPE`	`builtin`	`builtin`	set Pantacor Hub credenti
`PH_FACTORY_AUTOTOK`	token	empty	set factory auto token for communication with Panta
`PH_LIBEVENT_HTTP_RETRIES`	number of retries	`1`	set HTTP request number for communication with Pa Hub
`PH_LIBEVENT_HTTP_TIMEOUT`	time (in seconds)	`60`	set HTTP request timeout communication with Panta
`PH_METADATA_DEVMETA_INTERVAL`	time (in seconds)	`10`	set push interval for device metadata to Pantacor Hub
`PH_METADATA_USRMETA_INTERVAL`	time (in seconds)	`5`	set refresh interval for use metadata from Pantacor H
`PH_ONLINE_REQUEST_THRESHOLD`	number of failures	`0`	number of failed requests Pantacor Hub allowed to s consider device online
`PH_UPDATER_INTERVAL`	time (in seconds)	`60`	set time between Pantacor update requests
`PH_UPDATER_NETWORK_TIMEOUT`	time (in seconds)	`120`	set time before rollback if cannot communicate with Hub
`PH_UPDATER_TRANSFER_MAX_COUNT`	number of transfers	`5`	set maximum number of o transfers to and from Panta during updates
`PV_BOOTLOADER_FITCONFIG`	string	empty	set FIT configuration name
`PV_BOOTLOADER_MTD_ENV`	string	empty	set MTD name for bootloa
`PV_BOOTLOADER_MTD_ONLY`	`0` or `1`	`0`	enable MTD for bootloade
`PV_BOOTLOADER_TYPE`	`uboot`, `uboot-ab`, `uboot-pvk`, `rpiab` or `grub`	`uboot`	set bootloader type
`PV_BOOTLOADER_UBOOTAB_A_NAME`	string	`fitA`	name of the partition to us uboot-ab mode
`PV_BOOTLOADER_UBOOTAB_B_NAME`	string	`fitB`	name of the partition to us uboot-ab mode

Key	Value	Default	Description
`PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME`	string	empty	name of the partition wher uboot environment is back
`PV_BOOTLOADER_UBOOTAB_ENV_NAME`	string	empty	name of the partition wher uboot environment is store
`PV_BOOTLOADER_UBOOTAB_ENV_OFFSET`	offset in bytes	`0`	environment offset from th beginning of the partition
`PV_BOOTLOADER_UBOOTAB_ENV_SIZE`	size in bytes	`0`	size of the uboot environm
`PV_CACHE_DEVMETADIR`	path	`/storage/cache/devmeta`	set persistent device meta
`PV_CACHE_USRMETADIR`	path	`/storage/cache/meta`	set persistent user metada
`PV_CONTROL_REMOTE`	`0` or `1`	`1`	allow remote control from Hub
`PV_CONTROL_REMOTE_ALWAYS`	`0` or `1`	`0`	keep communication with Hub even when a local rev running
`PV_DEBUG_SHELL`	`0` or `1`	`1`	enable local debug shell
`PV_DEBUG_SHELL_AUTOLOGIN`	`0` or `1`	`0`	enable autologin for debug
`PV_DEBUG_SHELL_TIMEOUT`	time (in seconds)	`60`	time that Pantavisor waits rebooting if debug shell co opened
`PV_DEBUG_SSH`	`0` or `1`	`1`	enable SSH debug access
`PV_DEBUG_SSH_AUTHORIZED_KEYS`	string	empty	set authorized keys for SS access
`PV_DISK_EXPORTSDIR`	path	`/exports`	set exports directory
`PV_DISK_VOLDIR`	path	`/volumes`	set volumes directory
`PV_DISK_WRITABLEDIR`	path	`/writable`	set writable directory
`PV_DROPBEAR_CACHE_DIR`	path	`/storage/cache/dropbear`	set debug ssh server cach directory
`PV_LIBEVENT_DEBUG_MODE`	`0` or `1`	`0`	enable event loop debug lo
`PV_LIBTHTTP_CERTSDIR`	path	`/certs`	set certificates directory fo
`PV_LIBTHTTP_LOG_LEVEL`	`0` to `5`	`3`	set libthttp log verbosity le
`PV_LOG_BUF_NITEMS`	integer	`128`	set in-memory logs buffer
`PV_LOG_CAPTURE`	`0` or `1`	`1`	capture logs from containe
`PV_LOG_CAPTURE_DMESG`	`0` or `1`	`1`	capture dmesg logs
`PV_LOG_DIR`	path	`/storage/logs/`	set logs directory
`PV_LOG_DIR_MAXSIZE`	integer with optional suffix `B` (default), `K`, `KB`, `M`, `MB`, `G`, `GB`, `T`, `TB`, `%`; `0` for auto 10% (100% if tmpfs)	`16777216`	max size of log directory
`PV_LOG_FILETREE_TIMESTAMP_FORMAT`	format string	empty	timestamp format for filetr

Key	Value	Default	Description
`PV_LOG_HYSTERESIS_FACTOR`	positive integer	`4`	controls the gap between l low watermarks for log dir cleanup
`PV_LOG_LEVEL`	`0` to `5`	`0`	set Pantavisor log level (0: 5: ALL)
`PV_LOG_LOGGERS`	`0` or `1`	`1`	enable loggers for contain
`PV_LOG_PUSH`	`0` or `1`	`1`	push logs to Pantacor Hub
`PV_LOG_ROTATE_FACTOR`	integer	`5`	determines per-file rotatio threshold for log directory
`PV_LOG_SERVER_OUTPUTS`	string	`filetree`	set log server outputs (con separated)
`PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT`	format string	empty	timestamp format for singl
`PV_LOG_STDOUT_TIMESTAMP_FORMAT`	format string	empty	timestamp format for stdou
`PV_LXC_LOG_LEVEL`	`0` to `5`	`2`	set LXC log level
`PV_NET_BRADDRESS4`	IP address	`10.0.3.1`	set bridge IPv4 address
`PV_NET_BRDEV`	interface name	`lxcbr0`	set bridge device name
`PV_NET_BRMASK4`	IP mask	`255.255.255.0`	set bridge IPv4 mask
`PV_OEM_NAME`	string	empty	set OEM name for configu overrides
`PV_POLICY`	string	empty	set policy name for config
`PV_REMOUNT_POLICY`	string	empty	set remount policy name f filesystem remounting
`PV_REVISION_RETRIES`	integer	`10`	number of retries for revisi transitions
`PV_SECUREBOOT_CHECKSUM`	`0` or `1`	`1`	enable artifact checksum v
`PV_SECUREBOOT_HANDLERS`	`0` or `1`	`1`	enable handlers verificatio
`PV_SECUREBOOT_MODE`	`disabled`, `audit`, `lenient` or `strict`	`lenient`	set secureboot mode
`PV_SECUREBOOT_OEM_TRUSTSTORE`	path	`/etc/pantavisor/certs/oem`	set path to OEM truststore
`PV_SECUREBOOT_TRUSTSTORE`	path	`/etc/pantavisor/certs`	set path to Pantavisor trus
`PV_STORAGE_DEVICE`	string	empty	set storage device name
`PV_STORAGE_FSTYPE`	string	empty	set storage filesystem type
`PV_STORAGE_GC_KEEP_FACTORY`	`0` or `1`	`0`	keep factory revision durin
`PV_STORAGE_GC_RESERVED`	percentage	`5`	reserved storage percenta
`PV_STORAGE_GC_THRESHOLD`	percentage	`0`	storage GC threshold perc
`PV_STORAGE_GC_THRESHOLD_DEFERTIME`	time (in seconds)	`600`	defer time for GC threshol
`PV_STORAGE_LOGTEMPSIZE`	size string	empty	set size for temporary log

Key	Value	Default	Description
`PV_STORAGE_MNTPOINT`	path	empty	set storage mount point
`PV_STORAGE_MNTTYPE`	string	empty	set storage mount type
`PV_STORAGE_PHCONFIG_VOL`	`0` or `1`	`0`	use volume for Pantahub configuration
`PV_STORAGE_WAIT`	time (in seconds)	`5`	time to wait for storage de
`PV_SYSCTL_*`	string	—	set any kernel sysctl at run maps to `/proc/sys/` (e.g. `PV_SYSCTL_KERNEL_CORE_PA` `proc/sys/kernel/core_pat`
`PV_SYSCTL_KERNEL_CORE_PATTERN`	string	`|/lib/pv/pvcrash --skip`	set kernel core dump patte
`PV_SYSTEM_APPARMOR_PROFILES`	string	empty	AppArmor profiles to load
`PV_SYSTEM_CONFDIR`	path	`/configs`	set directory for system configurations
`PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO`	`0` or `1`	`0`	enable early auto-loading
`PV_SYSTEM_ETCDIR`	path	`/etc`	set system etc directory
`PV_SYSTEM_ETCPANTAVISORDIR`	path	`/etc/pantavisor`	set Pantavisor etc director
`PV_SYSTEM_INIT_MODE`	`embedded`, `standalone` or `appengine`	`embedded`	set system init mode
`PV_SYSTEM_LIBDIR`	path	`/lib`	set system library directory
`PV_SYSTEM_MEDIADIR`	path	`/media`	set system media directory
`PV_SYSTEM_MOUNT_SECURITYFS`	`0` or `1`	`0`	mount securityfs
`PV_SYSTEM_RUNDIR`	path	`/run/pantavisor/pv`	set system run directory
`PV_SYSTEM_USRDIR`	path	`/usr`	set system usr directory
`PV_UPDATER_COMMIT_DELAY`	time (in seconds)	`25`	delay before committing a
`PV_UPDATER_GOALS_TIMEOUT`	time (in seconds)	`120`	timeout for reaching updat
`PV_UPDATER_USE_TMP_OBJECTS`	`0` or `1`	`0`	use temporary objects dur updates
`PV_VOLMOUNT_DM_EXTRA_ARGS`	string	empty	extra arguments for DM vo mounting
`PV_WDT_MODE`	`disabled`, `shutdown`, `startup` or `always`	`shutdown`	set watchdog mode
`PV_WDT_TIMEOUT`	time (in seconds)	`15`	set watchdog timeout

LEVELS

This table shows the configuration levels that are allowed for each configuration key.

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PH_CREDS_HOST	✗	✓	✓	✓	✓
PH_CREDS_ID	✗	✓	✓	✓	✓
PH_CREDS_PORT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_HOST	✗	✓	✓	✓	✓
PH_CREDS_PROXY_NOPROXYCONNECT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_PORT	✗	✓	✓	✓	✓
PH_CREDS_PRN	✗	✓	✓	✓	✓
PH_CREDS_SECRET	✗	✓	✓	✓	✓
PH_CREDS_TYPE	✗	✓	✓	✓	✓
PH_FACTORY_AUTOTOK	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_TIMEOUT	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_RETRIES	✗	✓	✓	✓	✓
PH_METADATA_DEVMETA_INTERVAL	✗	✓	✓	✓	✓
PH_METADATA_USRMETA_INTERVAL	✗	✓	✓	✓	✓
PH_ONLINE_REQUEST_THRESHOLD	✗	✓	✓	✓	✓
PH_UPDATER_INTERVAL	✗	✓	✓	✓	✓
PH_UPDATER_NETWORK_TIMEOUT	✗	✓	✓	✓	✓
PH_UPDATER_TRANSFER_MAX_COUNT	✗	✓	✓	✓	✓
PV_BOOTLOADER_FITCONFIG	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ENV	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ONLY	✓	✗	✓	✓	✗
PV_BOOTLOADER_TYPE	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_A_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_B_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_OFFSET	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_SIZE	✓	✗	✓	✓	✗
PV_CACHE_DEVMETADIR	✓	✗	✓	✓	✗
PV_CACHE_USRMETADIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_CONTROL_REMOTE	✓	✗	✓	✓	✓
PV_CONTROL_REMOTE_ALWAYS	✓	✗	✓	✓	✓
PV_DEBUG_SHELL	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_AUTOLOGIN	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_TIMEOUT	✓	✗	✓	✓	✗
PV_DEBUG_SSH	✓	✗	✓	✓	✓
PV_DEBUG_SSH_AUTHORIZED_KEYS	✓	✗	✓	✓	✓
PV_DISK_EXPORTSDIR	✓	✗	✓	✓	✗
PV_DISK_VOLDIR	✓	✗	✓	✓	✗
PV_DISK_WRITABLEDIR	✓	✗	✓	✓	✗
PV_DROPBEAR_CACHE_DIR	✓	✗	✓	✓	✗
PV_LIBEVENT_DEBUG_MODE	✓	✗	✓	✓	✓
PV_LIBTHTTP_CERTSDIR	✓	✗	✓	✓	✗
PV_LIBTHTTP_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_BUF_NITEMS	✓	✗	✓	✓	✓
PV_LOG_CAPTURE	✓	✗	✓	✓	✓
PV_LOG_CAPTURE_DMESG	✓	✗	✓	✓	✓
PV_LOG_DIR	✓	✗	✓	✓	✗
PV_LOG_DIR_MAXSIZE	✓	✗	✓	✓	✓
PV_LOG_FILETREE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_HYSTERESIS_FACTOR	✓	✗	✓	✓	✓
PV_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_LOGGERS	✓	✗	✓	✓	✓
PV_LOG_PUSH	✓	✗	✓	✓	✓
PV_LOG_ROTATE_FACTOR	✓	✗	✓	✓	✓
PV_LOG_SERVER_OUTPUTS	✓	✗	✓	✓	✓
PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_STDOUT_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LXC_LOG_LEVEL	✓	✗	✓	✓	✓

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PV_NET_BRADDRESS4	✓	✗	✓	✓	✓
PV_NET_BRDEV	✓	✗	✓	✓	✓
PV_NET_BRMASK4	✓	✗	✓	✓	✓
PV_OEM_NAME	✓	✗	✓	✓	✗
PV_POLICY	✓	✗	✓	✗	✗
PV_REMOUNT_POLICY	✗	✗	✓	✗	✗
PV_REVISION_RETRIES	✓	✗	✓	✓	✓
PV_SECUREBOOT_CHECKSUM	✓	✗	✓	✓	✗
PV_SECUREBOOT_HANDLERS	✓	✗	✓	✓	✗
PV_SECUREBOOT_MODE	✓	✗	✓	✓	✗
PV_SECUREBOOT_OEM_TRUSTSTORE	✓	✗	✓	✓	✗
PV_SECUREBOOT_TRUSTSTORE	✓	✗	✓	✓	✗
PV_STORAGE_DEVICE	✓	✗	✓	✓	✗
PV_STORAGE_FSTYPE	✓	✗	✓	✓	✗
PV_STORAGE_GC_KEEP_FACTORY	✓	✗	✓	✓	✓
PV_STORAGE_GC_RESERVED	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD_DEFERTIME	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD	✓	✗	✓	✓	✓
PV_STORAGE_LOGTEMPSIZE	✓	✗	✓	✓	✗
PV_STORAGE_MNTPOINT	✓	✗	✓	✓	✗
PV_STORAGE_MNTTYPE	✓	✗	✓	✓	✗
PV_STORAGE_WAIT	✓	✗	✓	✓	✗
PV_SYSCTL_*	✓	✗	✓	✓	✓
PV_SYSCTL_KERNEL_CORE_PATTERN	✓	✗	✓	✓	✓
PV_SYSTEM_APPARMOR_PROFILES	✓	✗	✓	✓	✗
PV_SYSTEM_CONFDIR	✓	✗	✓	✓	✗
PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO	✓	✗	✓	✓	✗
PV_SYSTEM_ETCDIR	✓	✗	✓	✓	✗
PV_SYSTEM_INIT_MODE	✓	✗	✓	✓	✗
PV_SYSTEM_LIBDIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_SYSTEM_MEDIADIR	✓	✗	✓	✓	✗
PV_SYSTEM_MOUNT_SECURITYFS	✓	✗	✓	✓	✗
PV_SYSTEM_RUNDIR	✓	✗	✓	✓	✗
PV_SYSTEM_USRDIR	✓	✗	✓	✓	✗
PV_UPDATER_COMMIT_DELAY	✓	✗	✓	✓	✓
PV_UPDATER_GOALS_TIMEOUT	✓	✗	✓	✓	✓
PV_UPDATER_USE_TMP_OBJECTS	✓	✗	✓	✓	✓
PV_VOLMOUNT_DM_EXTRA_ARGS	✓	✗	✓	✓	✓
PV_WDT_MODE	✓	✗	✓	✓	✗
PV_WDT_TIMEOUT	✓	✓	✓	✓	✗

Pantavisor Metadata

This page contains reference information about Pantavisor metadata.

DEVICE METADATA

This is the device metadata created by Pantavisor that will give you useful information about your device:

Key	Value	Description
`interfaces`	json	network interfaces of the device
`pantahub.address`	IP:port	Pantacor Hub address the client is communicating with
`pantahub.claimed`	0 or 1	1 if claimed in Pantacor Hub
`pantahub.online`	0 or 1	1 if connection to Pantacor Hub was established
`pantahub.state`	string	see Pantacor Hub states (init, register, claim, sync, login, wait hub, report, idle, prep download or download)
`pantavisor.arch`	string	CPU architecture
`pantavisor.claimed`	0 or 1	1 if device has ever been claimed (local or remote)
`pantavisor.cpumodel`	string	CPU model name
`pantavisor.dtmodel`	string	Device Tree model name
`pantavisor.mode`	local or remote	see operation modes
`pantavisor.revision`	string	revision number
`pantavisor.status`	string	revision status
`pantavisor.uname`	json	uname output
`pantavisor.version`	string	Pantavisor build version
`storage`	json	disk usage of the device
`sysinfo`	json	sysinfo
`time`	json	time information

User metadata

This is the user metadata that can be set by the user which is parsed and have some actions on Pantavisor:

Key	Value	Description
`pvr-sdk.authorized_keys`	SSH pub key	set public key to get SSH access
`pvr-auto-follow.url`	URL	device will automatically pull every change in the device associated to that clone URL
`pantahub.log.push`	0 or 1	disable/enable log pushing to Pantacor Hub. Overrides PV_LOG_PUSH
`<config-key>`	config-value	override any configuration keys that allow RUN level
`<container>/<key>`	value	send user metadata that can be consumed by one of the containers

Pantavisor State Format (state.json)

A Pantavisor revision is defined by a single JSON object called `state.json`. It acts as a **virtual filesystem manifest** where every key represents a relative file path within the revision, and every value is either a nested configuration object or a SHA256 identifier for a binary artifact.

1. ROOT LEVEL (`state.json`)

These keys represent the files at the root of a revision.

Key	Value Type	Mandatory	Description
`#spec`	string	Yes	Parser version. Must be `"pantavisor-service-system@1"`.
`README.md`	string	No	Documentation for the revision in Markdown format.
`bsp/run.json`	BSP Manifest	Yes	Board Support Package configuration.
`bsp/drivers.json`	Drivers Manifest	No	Abstract driver mapping for the kernel.
`device.json`	Infrastructure Manifest	No	Unified physical storage and logical group definition.
`groups.json`	Groups Manifest	No	(Legacy) Logical container orchestration groups.
`disks.json`	Disks Manifest	No	(Legacy) Physical storage medium definitions.
`<container>/run.json`	Container Manifest	Yes	Individual container configuration.
`<container>/services.json`	Service Exports	No	Services exported to the xconnect mesh.
`_sigs/<container>.json`	Signature Manifest	No	Security signature for container artifacts.
`_config/<container>/<path>`	string	No	Injects data into `<path>` inside the container's rootfs.
`<any/other/path>`	string	No	SHA256 identifier for a binary artifact at that path.

2. BSP (`bsp/run.json`)

Defines the core system boot assets.

Key	Value Type	Description
`linux`	path string	Path to the Linux kernel image.
`initrd`	path string	Path to the Pantavisor initrd binary.
`modules`	path string	Path to the modules squashfs image.
`firmware`	path string	Path to the firmware squashfs image.
`fdt`	path string	Path to the Flattened Device Tree binary.
`fit`	path string	Path to a FIT image (replaces linux/initrd/fdt).
`rpiab`	path string	Path to a Raspberry Pi boot image.
`addons`	array of paths	List of CPIO addons to merge into the initrd rootfs.
`initrd_config`	path string	Custom configuration for the initrd process.

3. DRIVERS (`bsp/drivers.json`)

Maps abstract driver names to kernel modules based on hardware.

Key	Value Type	Description
`#spec`	string	Must be `"driver-aliases@1"`.
`all`	object	Default module mappings for all hardware.
`dtb:<name>`	object	Module mappings specific to a Device Tree model.
`ovl:<name>`	object	Module mappings specific to a DT Overlay.

Example module list:

```
"wifi": [ "cfg80211", "brcmfmac ${user-meta:wifi.opts}" ]
```

---

4. INFRASTRUCTURE (`device.json`)

The unified hardware and orchestration manifest.

Key	Value Type	Description
`disks`	array	List of Disk Definitions. Strict parsing — unknown types are fatal.
`disks_v2`	array	Same schema as `disks`, additive. Parsed independently.
`disks_v3`	array	Same schema as `disks` with lenient parsing (unknown types are warned and skipped). Required for the `dual` type — old firmware safely ignores this key.
`groups`	array	List of Orchestration Groups.
`volumes`	object	List of Persistent Volumes for Pantavisor itself.

---

5. ORCHESTRATION (`groups.json`)

Defines how containers are grouped and started.

Key	Value Type	Default	Description
`name`	string	**Mandatory**	Unique logical name for the group.
`description`	string	empty	Human-readable description.
`status_goal`	enum	`STARTED`	Goal for all members: `MOUNTED`, `STARTED`, `READY`.
`restart_policy`	enum	`container`	Policy on failure: `system`, `container`.
`timeout`	integer	30	Seconds to wait for members to reach `status_goal`.
`auto_recovery`	object	none	Default auto-recovery for containers in this group. Inherited all-or-nothing by containers without their own `auto_recovery`.

---

6. STORAGE (`disks.json`)

Defines physical storage mediums. Each entry in the `disks` , `disks_v2` , or `disks_v3` array uses this schema. See the Disks overview for type-specific details and examples.

Key	Value Type	Default	Description
`name`	string	**Mandatory**	Unique name used in `run.json` storage keys and mount paths.
`aliases`	string array	empty	Additional names this disk answers to. Volumes referring to any alias resolve to this disk. Aliases must not shadow another disk's `name` or be claimed by more than one disk; conflicts make the state refuse to boot. See Aliases in the overview.
`type`	enum	**Mandatory**	`dm-crypt-caam` , `dm-crypt-dcp` , `dm-crypt-versatile` , `swap-disk` , `volume-disk` , `dual` , `directory` .
`path`	string	**Mandatory** (crypt, swap, vol)	Device/image path. CAAM v2: `-v2 <img>,<size>,<key>` . DCP/versatile: `<img>,<size>,<key>` .
`mode`	enum	**Mandatory** (crypt)	`mainline` or `nxp` . Selects key subsystem.
`format`	enum	`ext4`	Filesystem format: `ext4` , `ext3` , or `swap` .
`provision`	string	empty	Backend provisioning: `zram` , `file` , or a custom value. Required for swap and volume types.
`provision_ops`	string	empty	Backend-specific options (e.g. `disksize=128M comp_algorithm=lz4`).
`mount_target`	path	empty	Where to mount the disk on the host. Required for `volume-disk` .
`mount_options`	string	empty	Comma-separated mount flags (e.g. `MS_NOATIME,MS_NOSUID`).
`format_options`	string	empty	Arguments for the `mkfs` or `mkswap` command.
`default`	string	`"no"`	If `"yes"` , this disk is used for all volumes without a `disk` key.
`uuid`	string	empty	Disk UUID.
`disks`	array	**Mandatory** (dual)	Sub-disk names: `["primary-name", "secondary-name"]` .
`init_order`	array	**Mandatory** (dual)	Actions tried in sequence: `primary` , `secondary` , `create-primary` , `create-secondary` , `copy-once-to-primary` .

7. CONTAINER (`<container>/run.json`)

Configures an individual container runtime.

Key	Value Type	Mandatory	Description
`#spec`	string	Yes	Must be `"service-manifest-run@1"`.
`name`	string	Yes	Logical name of the container.
`type`	enum	Yes	Runtime type (currently only `lxc`).
`config`	path string	Yes	Path to the LXC configuration file.
`root-volume`	path string	Yes	Path to the rootfs squashfs artifact.
`volumes`	array	No	Additional artifacts to mount as volumes.
`group`	string	No	Orchestration group name (from `device.json`).
`status_goal`	enum	No	Target state: `MOUNTED`, `STARTED`, `READY`.
`restart_policy`	enum	No	`system` (reboot on crash) or `container` (restart LXC).
`roles`	array	No	Capability roles: `mgmt` (control API access) or `nobody`.
`storage`	object	Yes	Persistence settings for rootfs paths.
`drivers`	object	No	Requirements: `required`, `optional`, or `manual`.
`services`	object	No	Service mesh requirements.
`logs`	array	No	Logger configurations.
`exports`	array	No	(Boolean flag in code) Marks container as an exporter.
`auto_recovery`	object	No	Auto-recovery configuration. If absent, inherited from group.

Auto-Recovery Object

Configures automatic restart behavior when a container crashes. See Auto-Recovery overview for the broader context.

Key	Value Type	Default	Description
`policy`	enum	`no`	Recovery policy: `no`, `always`, `on-failure`, `unless-stopped`. Note: the current implementation does not distinguish exit codes — `on-failure` behaves the same as `always`.
`max_retries`	integer	0	Maximum restart attempts. 0 = unlimited.
`retry_delay`	integer	0	Initial delay in seconds before first restart.
`backoff_factor`	number	1.0	Multiplier applied to `retry_delay` on each subsequent retry.
`reset_window`	integer	0	Seconds of continuous uptime after which the retry counter resets to 0.
`stable_timeout`	integer	0	Seconds the container must survive after reaching its status goal to be considered stable. Used to gate TESTING commit.
`backoff_policy`	string	`reboot`	Action after `max_retries` exhausted in steady state: `reboot`, `never`, or a duration string (`10min`, `1h`, `30s`).

Storage Object

Defines persistence for specific directories. Keys are paths relative to container root. * **`persistence`**: `permanent` (survives updates), `revision` (survives reboots), `boot` (volatile). * **`disk`**: Logical disk name from `device.json`.

Service Requirements

Under the `services` key in `run.json`. * **`required` / `optional`**: Arrays of service requirement objects. * `name`: Logical name of the service to find. * `type`: Protocol (`rest`, `dbus`, `unix`, `drm`, `wayland`). * `target`: Path where Pantavisor should inject the socket/resource. * `role`: Masquerade as this role when connecting. * `interface`: Protocol-specific identifier.

---

8. SERVICE MESH (`<container>/services.json`)

Declares services this container provides to others.

Key	Value Type	Description
`#spec`	string	Must be `"service-manifest-xconnect@1"`.
`services`	array	List of service objects (`name`, `type`, `socket`).

---

9. SECURITY (`_sigs/<container>.json`)

JWS-based artifact verification.

Key	Value	Description
`#spec`	`"pvs@2"`	Parser version.
`protected`	base64 string	Encoded headers including `alg`, `typ`, and `pvs` path filters.
`signature`	base64 string	The cryptographic signature of the protected header and payload.
`x5c`	array	(In protected) Certificate chain for verification.
`jwk`	object	(In protected) JSON Web Key for verification.

Pantavisor Tools

On-device CLI tools shipped with Pantavisor for development, debugging, and container control.

PVENTER

Enter a running container's namespaces.

```
pventer -c <container-name> [CMD ...]
```

Without a command, drops into the container's default shell. With a command, executes it inside the container's namespace. Uses `fallbear-cmd` under the hood via LXC paths.

```
# Drop into a shell inside the container
pventer -c my-app

# Run a command inside the container
pventer -c my-app ps aux
```

PVCURL

Lightweight HTTP client for the Pantavisor control socket. Drop-in replacement for `curl` in environments where curl is not available. Uses `nc` to send HTTP/1.0 requests over a Unix socket.

```
pvcurl --unix-socket <socket-path> [OPTIONS] <endpoint>
```

Socket paths (tried in order by pvcontrol): - `/run/pantavisor/pv/pv-ctrl` — appengine / embedded mode - `/pv/pv-ctrl` — alternative embedded path - `/pantavisor/pv-ctrl` — default inside containers

```
# Query buildinfo
pvcurl --unix-socket /run/pantavisor/pv/pv-ctrl /buildinfo

# List running daemons
pvcurl --unix-socket /run/pantavisor/pv/pv-ctrl /daemons

# Send a signal to a container
pvcurl -X PUT --unix-socket /run/pantavisor/pv/pv-ctrl /signal \
    --data '{"name":"my-app","signal":15}'
```

Supports `-X`, `-H`, `--data`, `--upload-file`, `-s`, `-i`, `-v`, `-o`, `-w`, `--connect-timeout`.

PVCONTROL

Shell wrapper around `pvcurl` for common control operations. Prefers `curl` if available, falls back to `pvcurl`.

```
pvcontrol [-s <socket-path>] <command> [arguments]
```

Key commands

```
# Container and group status
pvcontrol ls                        # list containers in current revision
pvcontrol containers ls             # list containers with status
pvcontrol containers start <name>
pvcontrol containers stop <name>
pvcontrol containers restart <name>
pvcontrol groups ls                 # list container groups

# xconnect service mesh
pvcontrol graph ls                  # show current xconnect service graph

# Daemon status (REST API daemons)
pvcontrol daemons ls                # list managed daemons
pvcontrol daemons get <name>

# Status signals (sent by a container to signal readiness)
pvcontrol signal ready              # signal container is ready
pvcontrol signal alive              # signal container is alive (watchdog)

# System commands
pvcontrol cmd reboot [message]
pvcontrol cmd poweroff [message]
pvcontrol cmd run-gc                # trigger garbage collection
pvcontrol cmd enable-ssh            # start SSH server until next reboot

# Metadata
pvcontrol devmeta ls
pvcontrol devmeta save <key> <value>
pvcontrol usrmeta ls
```

```
# Build info
pvcontrol buildinfo
```

PVTX

Transaction tool for creating, modifying, and deploying Pantavisor system state revisions. Operates on JSON state documents that describe the desired system configuration (services, BSP, configs), with atomic commit and rollback via the pv-ctrl daemon.

Transaction commands

```
pvtx begin <base> [object]
pvtx add <file> | -
pvtx remove <part>
pvtx abort
pvtx commit
pvtx show
pvtx deploy <directory>
```

Command	Description
`begin <base> [object]`	Start a new transaction. `base` is a revision hash, `current`, or `empty`. Omit `object` for a remote transaction (synced via pv-ctrl); provide a path for a local transaction written to disk.
`add <file> | -`	Add a JSON or tarball (`.json`, `.tar`, `.tar.gz`, `.tgz`, `.bz2`) to the current transaction. Use `-` to read from stdin.
`remove <part>`	Remove a part from the revision. `part` can be a name (`nginx`), a signature path (`_sigs/nginx.json`), or a config path (`_config/nginx`).
`abort`	Discard the current transaction and clean up state.
`commit`	Commit a remote transaction to pv-ctrl; prints the new revision hash.
`show`	Print the current transaction state as JSON to stdout.
`deploy <directory>`	Write a local transaction to disk (creates `.pvr/json`, `.pvr/config`, `bsp/run.json`).

Queue commands

Queue mode builds an ordered sequence of operations, then applies them as a single transaction.

```
pvtx queue new <queue> <object>
pvtx queue remove <part>
pvtx queue unpack <tarball> | -
pvtx queue process [base] [queue] [object]
```

Command	Description
`queue new <queue> <object>`	Initialize a new queue at `<queue>`, saving objects at `<object>`.
`queue remove <part>`	Enqueue a remove operation for `<part>`.
`queue unpack <tarball> | -`	Enqueue an unpack step for the given tarball (or stdin).
`queue process [base] [queue] [object]`	Execute the queue against `base` revision (`current` by default).

Environment variables

Variable	Default	Description
`PVTXDIR`	`/var/pvr-sdk/pvtx`	Temp directory for transaction state
`PVTX_OBJECT_BUF_SIZE`	—	Buffer size for saving objects (512B–10M)
`PVTX_CTRL_BUF_SIZE`	—	Buffer size for pv-ctrl I/O (16K–10M)

Socket paths (auto-detected): `/pv/pv-ctrl` (container root) or `/pantavisor/pv-ctrl` (inside containers).

```
# Remote transaction: modify current revision and commit
pvtx begin current
pvtx add /path/to/package.tar.gz
pvtx remove nginx
pvtx show
pvtx commit

# Local transaction: build a state on disk
pvtx begin empty /tmp/objects
cat package.tgz | pvtx add -
pvtx deploy /deploy/path

# Queue-based workflow: batch operations, then apply
pvtx queue new /tmp/queue /tmp/objects
pvtx queue remove nginx
pvtx queue unpack /path/to/package.tgz
pvtx begin empty
pvtx queue process
pvtx show
```

Pantavisor xconnect

The `pv-xconnect` service mesh facilitates efficient container-to-container and container-to-host interactions. It uses a plugin-driven architecture to inject resources (Unix sockets, D-Bus proxies, DRM nodes) into consumer containers on demand.

For information on how to inspect or manage the service mesh via the Pantavisor Control API, see the Pantavisor Control Socket reference.

ARCHITECTURE

To manage interactions between containers, a dedicated process called `pv-xconnect` handles the mediation logic via on-demand plugins. It runs as a managed daemon spawned by Pantavisor init and is enabled for all init modes (embedded, standalone, and appengine).

Core Responsibilities

- **Discovery & Reconciliation**: Periodically consumes an `xconnect-graph` from Pantavisor's `pv-ctrl` socket and maintains the state of active connects.
- **Plumbing**: Provides namespace-aware helpers to inject virtual resources (sockets/device nodes) inside the consumer's namespace.
- **Security**: Acts as the single point of truth for role-based access control.

SERVICE MANIFESTS

Provider (`services.json`)

A container declares the services it provides in a `services.json` file. This file must use the `#spec` format for identification by Pantavisor's parser.

Example `services.json` :

```
{
  "#spec": "service-manifest-xconnect@1",
  "services": [
    {
      "name": "network-manager",
      "type": "rest",
      "socket": "/run/network-manager/api.sock"
    },
    {
      "name": "system-bus",
      "type": "dbus",
      "socket": "/run/dbus/system_bus_socket"
    }
  ]
}
```

Consumer (`args.json` / `run.json`)

Containers that consume services define their requirements in `args.json` during creation (e.g., with `pvr app add --arg-json args.json`). These are then rendered into the final `run.json` manifest.

Example `run.json` requirement:

```
{
  "#spec": "service-manifest-run@1",
  "name": "my-app",
  "services": {
    "required": [
      {
        "name": "system-bus",
        "type": "dbus",
        "interface": "org.pantavisor.Example",
        "target": "/run/dbus/system_bus_socket"
      }
    ]
  },
  "type": "lxc"
}
```

- **`interface`** : Protocol-specific identifier (e.g., D-Bus interface name).
- **`target`** : The path where `pv-xconnect` should inject the proxied resource inside the consumer container.

MEDIATION PATTERNS

Raw Unix Sockets

Provides direct proxying of Unix Domain Sockets between containers. It supports high-performance features like FD passing (SCM_RIGHTS) and Shared Memory handles.

REST

Identity-injected HTTP over UDS. `pv-xconnect` automatically injects `X-PV-Client` and `X-PV-Role` headers into the first request, allowing the provider to identify the consumer.

D-Bus

Policy-aware proxy for the system bus. It performs **Role-Based Identity Masquerading**: 1. `pv-xconnect` intercepts the D-Bus SASL authentication phase. 2. It takes the **Role** from the link and looks up the corresponding **UID** in the provider container's `/etc/passwd`. 3. It replaces the consumer's identity with the resolved UID.

This allows the provider's standard `dbus-daemon` to enforce fine-grained permissions using standard XML policy files based on the assigned role.

DRM / Graphics

- **Master Role**: Injects `/dev/dri/cardX` for display servers (KMS access).
- **Render Role**: Injects `/dev/dri/renderDX` for accelerated applications.

Wayland

Mediates the Wayland protocol for isolated UI rendering, allowing a containerized compositor to serve multiple isolated clients.

TOOLS

pvcurl

A lightweight shell script wrapping `nc` for HTTP-over-Unix-socket communication. It is preferred in App Engine environments where standard `curl` might not be available.

pvcontrol

A high-level CLI wrapper around `pvcurl` for common Pantavisor control operations.

7.1.4 028 (Development)

Pantavisor Log Sockets

The Pantavisor logging system uses two Unix sockets for inter-process log management: `pv-ctrl-log` for receiving direct log messages and `pv-fd-log` for subscribing file descriptors to be polled by Pantavisor.

PV-CTRL-LOG

This socket is used by applications and containers to send log messages directly to the Pantavisor Log Server. All messages must follow the `logserver_msg` structure:

```
struct logserver_msg {
    int code; // Protocol code: 0 for LEGACY, 256 for CMD
    int len;  // Length of the following buffer
    char buf[0]; // Log data buffer
};
```

Legacy Protocol (code = 0)

The `buf` contains the log metadata and message, separated by null terminators (`\0`):

```
level\0platform\0source\0data
```

- **level**: Log level as a string (e.g., "3" for INFO).
- **platform**: Name of the container or "pantavisor".
- **source**: The specific source of the log (e.g., a process name or module).
- **data**: The actual log message content.

The supported log levels are: * `0` : FATAL * `1` : ERROR * `2` : WARN * `3` : INFO * `4` : DEBUG * `5` : ALL

PV-FD-LOG

This socket allows containers to delegate the polling of their file descriptors (e.g., pipes, sockets, or open files) to Pantavisor.

Subscription Protocol

To subscribe or unsubscribe a file descriptor, you must use the `sendmsg` system call with `SCM_RIGHTS` to pass the file descriptor.

The `msghdr` must contain an `iovec` array with 4 elements:

iov Index	Type	Max Length	Description
`iov[0]`	string	50 bytes	Platform name (container name)
`iov[1]`	string	50 bytes	Source name (e.g., "stdout", "syslog")
`iov[2]`	int	4 bytes	Log level for the messages from this FD
`iov[3]`	int	4 bytes	Action: `1` to subscribe, `0` to unsubscribe

Subscribe

- Send the file descriptor using `SCM_RIGHTS` .
- Set `iov[3]` to `1` .
- Pantavisor will poll this FD and create a corresponding log file at `/storage/logs/current/<platform>/<source>` .

Unsubscribe

- Set `iov[3]` to `0` .
- The file descriptor passed in `SCM_RIGHTS` can be `-1` .
- Pantavisor will stop polling and close its internal reference to the FD for that platform/source pair.

> **Note**
>
> Only one file descriptor can be subscribed per platform-source pair. Subscribing a new FD for an existing pair will replace the previous one.

Pantavisor Control Socket

The pv-ctrl socket enables communication between the containers and Pantavisor during runtime.

The following subsections describe the behaviour of the HTTP API for the different endpoints, which you can test either using any HTTP client or our pvcontrol tool from the default pvr-sdk container.

> **Note**
>
> The examples provided use cURL, but any HTTP client inside of your container should work.

/CONTAINERS

This endpoint can be used to list and manage containers in the current revision.

List containers

Returns all containers with their status, restart policy, auto-recovery state, and service mesh information.

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/containers"
```

Lifecycle control

Containers with `restart_policy: "container"` can be stopped, started, and restarted via the API. Containers with `restart_policy: "system"` are protected and will return HTTP 403.

To stop a container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"stop\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Stopping a container sets the `user_stopped` flag, which prevents auto-recovery from restarting it. The container's auto-recovery configuration is preserved — no retry counters are consumed and no backoff is triggered.

To start a previously stopped container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"start\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Starting clears the `user_stopped` flag and returns the container to the normal engine lifecycle. Auto-recovery is fully restored.

To restart a running container:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"restart\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/containers/my-container"
```

Restart force-stops the container and resets the retry counter to zero. For containers with auto-recovery configured, the recovery engine restarts it naturally through the standard path. For containers without auto-recovery (`RECOVERY_NO`), the container is restarted directly.

Action	State change	Auto-recovery effect
stop	Running STOPPED	`user_stopped` set; recovery skipped
start	STOPPED INSTALLED start	`user_stopped` cleared; recovery restored
restart (with recovery)	Running STOPPED recovery engine restarts	Retry counter reset to 0
restart (no recovery)	Running STOPPED INSTALLED start	Direct restart

/DAEMONS

This endpoint can be used to list and manage Pantavisor internal daemons.

To list all daemons and their current status:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons"
```

To stop a specific daemon (e.g., `pv-xconnect`):

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"stop\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons/pv-xconnect"
```

To start a specific daemon:

```
$ curl -X PUT --header "Content-Type: application/json" --data "{\"action\":\"start\"}" --unix-socket /pantavisor/pv-ctrl "http://localhost/daemons/pv-xconnect"
```

/GROUPS

These requests can be used to list groups.

To list all groups in the current revision:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/groups"
```

/SIGNAL

This type of command can be issued to alter the container status in Pantavisor.

To send the one-shot readiness signal:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"type\":\"ready\",\"payload\":\"\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
signal
```

This request will fail if the signal type is not supported, or if that status goal is not expected.

/COMMANDS

These commands can perform changes in Pantavisor container engine itself, so the result will not be inmediate to the request.

An example of a command that tells Pantavisor to transition to revision 4:

```
$ curl -X POST --header "Content-Type: application/json" --data "{\"op\":\"LOCAL_RUN\",\"payload\":\"4\"}" --unix-socket /pantavisor/pv-ctrl http://localhost/
commands
```

As you can see, the body of the request contains the command itself in JSON format.

These are the different commands that are supported. You can test them by substituting `op` and `payload` in the above command:

op	payload	Description
UPDATE_METADATA	{key, value}	upload the json as a new pair of device metadata to Pantacor Hub
REBOOT_DEVICE	message	reboot device with optional message
POWEROFF_DEVICE	message	poweroff device with optional message
TRY_ONCE	revision	try a revision once (will rollback on failure or next reboot)
LOCAL_RUN	revision	transition to specified revision
MAKE_FACTORY	revision	make the revision the factory revision. If revision is not set, Pantavisor will use the current one. Device needs to be not claimed
RUN_GC	N/A	run garbage collector
ENABLE_SSH	N/A	enable SSH server ignoring config until reboot
DISABLE_SSH	N/A	disable SSH server ignoring config until reboot
GO_REMOTE	N/A	go remote when running on a locals/ revision if allowed by config
DEFER_REBOOT	N/A	defer reboot when debug shell is active
LOCAL_RUN_COMMIT	revision	transition to revision and commit it automatically
LOCAL_APPLY	revision	apply revision changes without a full reboot
XCONNECT_GRAPH	N/A	trigger an immediate xconnect graph reconciliation

/OBJECTS

This endpoint can be used to list, send and receive objects to and from Pantavisor. Bear in mind that objects are the artifacts that form a Pantavisor revision.

An example of listing the objects that are stored in the device:

```
$ curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects"
```

Here, an example for getting one of those objects:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

You can even put new objects in Pantavisor. Notice that the sha256sum of object has to match the specified sha in the URI:

```
curl -X PUT --upload-file object --unix-socket /pantavisor/pv-ctrl "http://localhost/objects/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

/STEPS

This endpoint can be used to list, send and receive step jsons, as well as get the update progress and set the commit message for any of them.

Let us go with the examples, first, to list all steps installed in the device:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps"
```

To get an existing step json:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a"
```

To send a new json, the format of the new revision in the URI has to contain the "locals/" prefix. The name after the prefix must be under 64 characters and must not contain any other "/" character. These revisions that are installed using the socket (locals) are treated in a different way than the ones installed from Pantacor Hub (remotes), as you will have to manually request the transition to locals using the run

command. Most importantly, locals will not attempt any communication with Pantacor Hub during runtime unless a go remote command is issued.

```
curl -X PUT --upload-file json --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/locals/example"
```

To get the update progress (DONE, TESTING, INPROGRESS...) and some related information of a revision:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/progress"
```

Finally, you can set a commit message that will be stored along a revision and showed when listing revisions so the user can idendifcate each one of them:

```
curl -X PUT --data "message" --unix-socket /pantavisor/pv-ctrl "http://localhost/steps/033e779113f2499a2bfb55c0c374803fba9c820361d71bbda616643007cacd5a/
commitmsg"
```

/USER-META

The user-meta endpoint offers the ability to list, save and delete user metadata.

To list all user meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta"
```

An example of creating or updating a new user metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/user-meta/key"
```

/DEVICE-META

The device-meta endpoint offers the ability to list device metadata.

To list all device meta in json format:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta"
```

An example of creating or updating a new device metadata pair in device:

```
curl -X PUT --data value --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

To delete one pair, we would do this, having in mind the same behaviour of operation modes as with putting metadata pairs:

```
curl -X DELETE --unix-socket /pantavisor/pv-ctrl "http://localhost/device-meta/key"
```

/XCONNECT-GRAPH

This endpoint returns the current xconnect service mesh graph in JSON format. For details on how the service mesh operates and how to define manifests, see the Pantavisor xconnect reference.

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/xconnect-graph"
```

/BUILDINFO

For debugging porpuses, it is possible to get the repo manifest that was used to build this Pantavisor binary.

With cURL, this would look like:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl "http://localhost/buildinfo"
```

/DRIVERS

The drivers endpoint lets you list load and unload managed drivers.

To list drivers referenced by container and their load state:

```
curl -X GET --unix-socket /pantavisor/pv-ctrl http://localhost/drivers
```

To load manual drivers at bulk from within container:

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/load
```

To load individual manual drivers, in this case one driver named "wifi":

```
curl -X PUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/load
```

Same for unloading, it can be done at bulk:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/unload
```

And individually:

```
curl -XPUT --unix-socket /pantavisor/pv-ctrl http://localhost/drivers/wifi/unload
```

Pantavisor Configuration

> **Note**
>
> This reference page presents the newly unified configuration key syntax. To get to the deprecated but still supported previous format, you will have to go here.

SUMMARY

> **Note**
>
> The key syntax is the same for all configuration levels.

> **Note**
>
> All keys are case insensitive.

> **Note**
>
> Syntax and behavior of keys tagged with (experimental) might change and break backwards compatibility.

This table contains the currently supported list of configuration keys, sorted alphabetically.

Key	Value	Default	Description
`PH_CREDS_HOST`	IP or hostname	`api.pantahub.com`	set Pantacor Hub address
`PH_CREDS_ID`	string	empty	set Pantacor Hub device ID
`PH_CREDS_PORT`	port	`443`	set port for communication Pantacor Hub
`PH_CREDS_PROXY_HOST`	IP or hostname	empty	set Pantacor Hub proxy ad
`PH_CREDS_PROXY_NOPROXYCONNECT`	`0` or `1`	`0`	disable proxy communicat Pantacor Hub
`PH_CREDS_PROXY_PORT`	port	`3218`	set port for proxy commun with Pantacor Hub
`PH_CREDS_PRN`	string	empty	set Pantacor Hub device P
`PH_CREDS_SECRET`	string	empty	set Pantacor Hub credenti
`PH_CREDS_TYPE`	`builtin`	`builtin`	set Pantacor Hub credenti
`PH_FACTORY_AUTOTOK`	token	empty	set factory auto token for communication with Panta
`PH_LIBEVENT_HTTP_RETRIES`	number of retries	`1`	set HTTP request number for communication with Pa Hub
`PH_LIBEVENT_HTTP_TIMEOUT`	time (in seconds)	`60`	set HTTP request timeout communication with Panta
`PH_METADATA_DEVMETA_INTERVAL`	time (in seconds)	`10`	set push interval for device metadata to Pantacor Hub
`PH_METADATA_USRMETA_INTERVAL`	time (in seconds)	`5`	set refresh interval for use metadata from Pantacor H
`PH_ONLINE_REQUEST_THRESHOLD`	number of failures	`0`	number of failed requests Pantacor Hub allowed to s consider device online
`PH_UPDATER_INTERVAL`	time (in seconds)	`60`	set time between Pantacor update requests
`PH_UPDATER_NETWORK_TIMEOUT`	time (in seconds)	`120`	set time before rollback if cannot communicate with Hub
`PH_UPDATER_TRANSFER_MAX_COUNT`	number of transfers	`5`	set maximum number of o transfers to and from Panta during updates
`PV_BOOTLOADER_FITCONFIG`	string	empty	set FIT configuration name
`PV_BOOTLOADER_MTD_ENV`	string	empty	set MTD name for bootloa
`PV_BOOTLOADER_MTD_ONLY`	`0` or `1`	`0`	enable MTD for bootloade
`PV_BOOTLOADER_TYPE`	`uboot`, `uboot-ab`, `uboot-pvk`, `rpiab` or `grub`	`uboot`	set bootloader type
`PV_BOOTLOADER_UBOOTAB_A_NAME`	string	`fitA`	name of the partition to us uboot-ab mode
`PV_BOOTLOADER_UBOOTAB_B_NAME`	string	`fitB`	name of the partition to us uboot-ab mode

Key	Value	Default	Description
`PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME`	string	empty	name of the partition wher uboot environment is back
`PV_BOOTLOADER_UBOOTAB_ENV_NAME`	string	empty	name of the partition wher uboot environment is store
`PV_BOOTLOADER_UBOOTAB_ENV_OFFSET`	offset in bytes	`0`	environment offset from th beginning of the partition
`PV_BOOTLOADER_UBOOTAB_ENV_SIZE`	size in bytes	`0`	size of the uboot environm
`PV_CACHE_DEVMETADIR`	path	`/storage/cache/devmeta`	set persistent device meta
`PV_CACHE_USRMETADIR`	path	`/storage/cache/meta`	set persistent user metada
`PV_CONTROL_REMOTE`	`0` or `1`	`1`	allow remote control from Hub
`PV_CONTROL_REMOTE_ALWAYS`	`0` or `1`	`0`	keep communication with Hub even when a local rev running
`PV_DEBUG_SHELL`	`0` or `1`	`1`	enable local debug shell
`PV_DEBUG_SHELL_AUTOLOGIN`	`0` or `1`	`0`	enable autologin for debug
`PV_DEBUG_SHELL_TIMEOUT`	time (in seconds)	`60`	time that Pantavisor waits rebooting if debug shell co opened
`PV_DEBUG_SSH`	`0` or `1`	`1`	enable SSH debug access
`PV_DEBUG_SSH_AUTHORIZED_KEYS`	string	empty	set authorized keys for SS access
`PV_DISK_EXPORTSDIR`	path	`/exports`	set exports directory
`PV_DISK_VOLDIR`	path	`/volumes`	set volumes directory
`PV_DISK_WRITABLEDIR`	path	`/writable`	set writable directory
`PV_DROPBEAR_CACHE_DIR`	path	`/storage/cache/dropbear`	set debug ssh server cach directory
`PV_LIBEVENT_DEBUG_MODE`	`0` or `1`	`0`	enable event loop debug lo
`PV_LIBTHTTP_CERTSDIR`	path	`/certs`	set certificates directory fo
`PV_LIBTHTTP_LOG_LEVEL`	`0` to `5`	`3`	set libthttp log verbosity le
`PV_LOG_BUF_NITEMS`	integer	`128`	set in-memory logs buffer
`PV_LOG_CAPTURE`	`0` or `1`	`1`	capture logs from containe
`PV_LOG_CAPTURE_DMESG`	`0` or `1`	`1`	capture dmesg logs
`PV_LOG_DIR`	path	`/storage/logs/`	set logs directory
`PV_LOG_DIR_MAXSIZE`	integer with optional suffix `B` (default), `K`, `KB`, `M`, `MB`, `G`, `GB`, `T`, `TB`, `%`; `0` for auto 10% (100% if tmpfs)	`16777216`	max size of log directory
`PV_LOG_FILETREE_TIMESTAMP_FORMAT`	format string	empty	timestamp format for filetr

Key	Value	Default	Description
`PV_LOG_HYSTERESIS_FACTOR`	positive integer	`4`	controls the gap between l low watermarks for log dir cleanup
`PV_LOG_LEVEL`	`0` to `5`	`0`	set Pantavisor log level (0: 5: ALL)
`PV_LOG_LOGGERS`	`0` or `1`	`1`	enable loggers for contain
`PV_LOG_PUSH`	`0` or `1`	`1`	push logs to Pantacor Hub
`PV_LOG_ROTATE_FACTOR`	integer	`5`	determines per-file rotatio threshold for log directory
`PV_LOG_SERVER_OUTPUTS`	string	`filetree`	set log server outputs (con separated)
`PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT`	format string	empty	timestamp format for singl
`PV_LOG_STDOUT_TIMESTAMP_FORMAT`	format string	empty	timestamp format for stdou
`PV_LXC_LOG_LEVEL`	`0` to `5`	`2`	set LXC log level
`PV_NET_BRADDRESS4`	IP address	`10.0.3.1`	set bridge IPv4 address
`PV_NET_BRDEV`	interface name	`lxcbr0`	set bridge device name
`PV_NET_BRMASK4`	IP mask	`255.255.255.0`	set bridge IPv4 mask
`PV_OEM_NAME`	string	empty	set OEM name for configu overrides
`PV_POLICY`	string	empty	set policy name for config
`PV_REMOUNT_POLICY`	string	empty	set remount policy name f filesystem remounting
`PV_REVISION_RETRIES`	integer	`10`	number of retries for revisi transitions
`PV_SECUREBOOT_CHECKSUM`	`0` or `1`	`1`	enable artifact checksum v
`PV_SECUREBOOT_HANDLERS`	`0` or `1`	`1`	enable handlers verificatio
`PV_SECUREBOOT_MODE`	`disabled`, `audit`, `lenient` or `strict`	`lenient`	set secureboot mode
`PV_SECUREBOOT_OEM_TRUSTSTORE`	path	`/etc/pantavisor/certs/oem`	set path to OEM truststore
`PV_SECUREBOOT_TRUSTSTORE`	path	`/etc/pantavisor/certs`	set path to Pantavisor trus
`PV_STORAGE_DEVICE`	string	empty	set storage device name
`PV_STORAGE_FSTYPE`	string	empty	set storage filesystem type
`PV_STORAGE_GC_KEEP_FACTORY`	`0` or `1`	`0`	keep factory revision durin
`PV_STORAGE_GC_RESERVED`	percentage	`5`	reserved storage percenta
`PV_STORAGE_GC_THRESHOLD`	percentage	`0`	storage GC threshold perc
`PV_STORAGE_GC_THRESHOLD_DEFERTIME`	time (in seconds)	`600`	defer time for GC threshol
`PV_STORAGE_LOGTEMPSIZE`	size string	empty	set size for temporary log

Key	Value	Default	Description
`PV_STORAGE_MNTPOINT`	path	empty	set storage mount point
`PV_STORAGE_MNTTYPE`	string	empty	set storage mount type
`PV_STORAGE_PHCONFIG_VOL`	`0` or `1`	`0`	use volume for Pantahub configuration
`PV_STORAGE_WAIT`	time (in seconds)	`5`	time to wait for storage de
`PV_SYSCTL_*`	string	—	set any kernel sysctl at run maps to `/proc/sys/` (e.g. `PV_SYSCTL_KERNEL_CORE_PA` `proc/sys/kernel/core_pat`
`PV_SYSCTL_KERNEL_CORE_PATTERN`	string	`|/lib/pv/pvcrash --skip`	set kernel core dump patte
`PV_SYSTEM_APPARMOR_PROFILES`	string	empty	AppArmor profiles to load
`PV_SYSTEM_CONFDIR`	path	`/configs`	set directory for system configurations
`PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO`	`0` or `1`	`0`	enable early auto-loading
`PV_SYSTEM_ETCDIR`	path	`/etc`	set system etc directory
`PV_SYSTEM_ETCPANTAVISORDIR`	path	`/etc/pantavisor`	set Pantavisor etc director
`PV_SYSTEM_INIT_MODE`	`embedded`, `standalone` or `appengine`	`embedded`	set system init mode
`PV_SYSTEM_LIBDIR`	path	`/lib`	set system library directory
`PV_SYSTEM_MEDIADIR`	path	`/media`	set system media directory
`PV_SYSTEM_MOUNT_SECURITYFS`	`0` or `1`	`0`	mount securityfs
`PV_SYSTEM_RUNDIR`	path	`/run/pantavisor/pv`	set system run directory
`PV_SYSTEM_USRDIR`	path	`/usr`	set system usr directory
`PV_UPDATER_COMMIT_DELAY`	time (in seconds)	`25`	delay before committing a
`PV_UPDATER_GOALS_TIMEOUT`	time (in seconds)	`120`	timeout for reaching updat
`PV_UPDATER_USE_TMP_OBJECTS`	`0` or `1`	`0`	use temporary objects dur updates
`PV_VOLMOUNT_DM_EXTRA_ARGS`	string	empty	extra arguments for DM vo mounting
`PV_WDT_MODE`	`disabled`, `shutdown`, `startup` or `always`	`shutdown`	set watchdog mode
`PV_WDT_TIMEOUT`	time (in seconds)	`15`	set watchdog timeout

LEVELS

This table shows the configuration levels that are allowed for each configuration key.

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PH_CREDS_HOST	✗	✓	✓	✓	✓
PH_CREDS_ID	✗	✓	✓	✓	✓
PH_CREDS_PORT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_HOST	✗	✓	✓	✓	✓
PH_CREDS_PROXY_NOPROXYCONNECT	✗	✓	✓	✓	✓
PH_CREDS_PROXY_PORT	✗	✓	✓	✓	✓
PH_CREDS_PRN	✗	✓	✓	✓	✓
PH_CREDS_SECRET	✗	✓	✓	✓	✓
PH_CREDS_TYPE	✗	✓	✓	✓	✓
PH_FACTORY_AUTOTOK	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_TIMEOUT	✗	✓	✓	✓	✓
PH_LIBEVENT_HTTP_RETRIES	✗	✓	✓	✓	✓
PH_METADATA_DEVMETA_INTERVAL	✗	✓	✓	✓	✓
PH_METADATA_USRMETA_INTERVAL	✗	✓	✓	✓	✓
PH_ONLINE_REQUEST_THRESHOLD	✗	✓	✓	✓	✓
PH_UPDATER_INTERVAL	✗	✓	✓	✓	✓
PH_UPDATER_NETWORK_TIMEOUT	✗	✓	✓	✓	✓
PH_UPDATER_TRANSFER_MAX_COUNT	✗	✓	✓	✓	✓
PV_BOOTLOADER_FITCONFIG	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ENV	✓	✗	✓	✓	✗
PV_BOOTLOADER_MTD_ONLY	✓	✗	✓	✓	✗
PV_BOOTLOADER_TYPE	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_A_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_B_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_BAK_NAME	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_OFFSET	✓	✗	✓	✓	✗
PV_BOOTLOADER_UBOOTAB_ENV_SIZE	✓	✗	✓	✓	✗
PV_CACHE_DEVMETADIR	✓	✗	✓	✓	✗
PV_CACHE_USRMETADIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_CONTROL_REMOTE	✓	✗	✓	✓	✓
PV_CONTROL_REMOTE_ALWAYS	✓	✗	✓	✓	✓
PV_DEBUG_SHELL	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_AUTOLOGIN	✓	✗	✓	✓	✗
PV_DEBUG_SHELL_TIMEOUT	✓	✗	✓	✓	✗
PV_DEBUG_SSH	✓	✗	✓	✓	✓
PV_DEBUG_SSH_AUTHORIZED_KEYS	✓	✗	✓	✓	✓
PV_DISK_EXPORTSDIR	✓	✗	✓	✓	✗
PV_DISK_VOLDIR	✓	✗	✓	✓	✗
PV_DISK_WRITABLEDIR	✓	✗	✓	✓	✗
PV_DROPBEAR_CACHE_DIR	✓	✗	✓	✓	✗
PV_LIBEVENT_DEBUG_MODE	✓	✗	✓	✓	✓
PV_LIBTHTTP_CERTSDIR	✓	✗	✓	✓	✗
PV_LIBTHTTP_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_BUF_NITEMS	✓	✗	✓	✓	✓
PV_LOG_CAPTURE	✓	✗	✓	✓	✓
PV_LOG_CAPTURE_DMESG	✓	✗	✓	✓	✓
PV_LOG_DIR	✓	✗	✓	✓	✗
PV_LOG_DIR_MAXSIZE	✓	✗	✓	✓	✓
PV_LOG_FILETREE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_HYSTERESIS_FACTOR	✓	✗	✓	✓	✓
PV_LOG_LEVEL	✓	✗	✓	✓	✓
PV_LOG_LOGGERS	✓	✗	✓	✓	✓
PV_LOG_PUSH	✓	✗	✓	✓	✓
PV_LOG_ROTATE_FACTOR	✓	✗	✓	✓	✓
PV_LOG_SERVER_OUTPUTS	✓	✗	✓	✓	✓
PV_LOG_SINGLEFILE_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LOG_STDOUT_TIMESTAMP_FORMAT	✓	✗	✓	✓	✓
PV_LXC_LOG_LEVEL	✓	✗	✓	✓	✓

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
PV_NET_BRADDRESS4	✓	✗	✓	✓	✓
PV_NET_BRDEV	✓	✗	✓	✓	✓
PV_NET_BRMASK4	✓	✗	✓	✓	✓
PV_OEM_NAME	✓	✗	✓	✓	✗
PV_POLICY	✓	✗	✓	✗	✗
PV_REMOUNT_POLICY	✗	✗	✓	✗	✗
PV_REVISION_RETRIES	✓	✗	✓	✓	✓
PV_SECUREBOOT_CHECKSUM	✓	✗	✓	✓	✗
PV_SECUREBOOT_HANDLERS	✓	✗	✓	✓	✗
PV_SECUREBOOT_MODE	✓	✗	✓	✓	✗
PV_SECUREBOOT_OEM_TRUSTSTORE	✓	✗	✓	✓	✗
PV_SECUREBOOT_TRUSTSTORE	✓	✗	✓	✓	✗
PV_STORAGE_DEVICE	✓	✗	✓	✓	✗
PV_STORAGE_FSTYPE	✓	✗	✓	✓	✗
PV_STORAGE_GC_KEEP_FACTORY	✓	✗	✓	✓	✓
PV_STORAGE_GC_RESERVED	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD_DEFERTIME	✓	✗	✓	✓	✓
PV_STORAGE_GC_THRESHOLD	✓	✗	✓	✓	✓
PV_STORAGE_LOGTEMPSIZE	✓	✗	✓	✓	✗
PV_STORAGE_MNTPOINT	✓	✗	✓	✓	✗
PV_STORAGE_MNTTYPE	✓	✗	✓	✓	✗
PV_STORAGE_WAIT	✓	✗	✓	✓	✗
PV_SYSCTL_*	✓	✗	✓	✓	✓
PV_SYSCTL_KERNEL_CORE_PATTERN	✓	✗	✓	✓	✓
PV_SYSTEM_APPARMOR_PROFILES	✓	✗	✓	✓	✗
PV_SYSTEM_CONFDIR	✓	✗	✓	✓	✗
PV_SYSTEM_DRIVERS_LOAD_EARLY_AUTO	✓	✗	✓	✓	✗
PV_SYSTEM_ETCDIR	✓	✗	✓	✓	✗
PV_SYSTEM_INIT_MODE	✓	✗	✓	✓	✗
PV_SYSTEM_LIBDIR					

Key	pv.conf	ph.conf	env,bootargs	Policy	OEM
	✓	✗	✓	✓	✗
PV_SYSTEM_MEDIADIR	✓	✗	✓	✓	✗
PV_SYSTEM_MOUNT_SECURITYFS	✓	✗	✓	✓	✗
PV_SYSTEM_RUNDIR	✓	✗	✓	✓	✗
PV_SYSTEM_USRDIR	✓	✗	✓	✓	✗
PV_UPDATER_COMMIT_DELAY	✓	✗	✓	✓	✓
PV_UPDATER_GOALS_TIMEOUT	✓	✗	✓	✓	✓
PV_UPDATER_USE_TMP_OBJECTS	✓	✗	✓	✓	✓
PV_VOLMOUNT_DM_EXTRA_ARGS	✓	✗	✓	✓	✓
PV_WDT_MODE	✓	✗	✓	✓	✗
PV_WDT_TIMEOUT	✓	✓	✓	✓	✗

Pantavisor Metadata

This page contains reference information about Pantavisor metadata.

DEVICE METADATA

This is the device metadata created by Pantavisor that will give you useful information about your device:

Key	Value	Description
`interfaces`	json	network interfaces of the device
`pantahub.address`	IP:port	Pantacor Hub address the client is communicating with
`pantahub.claimed`	0 or 1	1 if claimed in Pantacor Hub
`pantahub.online`	0 or 1	1 if connection to Pantacor Hub was established
`pantahub.state`	string	see Pantacor Hub states (init, register, claim, sync, login, wait hub, report, idle, prep download or download)
`pantavisor.arch`	string	CPU architecture
`pantavisor.claimed`	0 or 1	1 if device has ever been claimed (local or remote)
`pantavisor.cpumodel`	string	CPU model name
`pantavisor.dtmodel`	string	Device Tree model name
`pantavisor.mode`	local or remote	see operation modes
`pantavisor.revision`	string	revision number
`pantavisor.status`	string	revision status
`pantavisor.uname`	json	uname output
`pantavisor.version`	string	Pantavisor build version
`storage`	json	disk usage of the device
`sysinfo`	json	sysinfo
`time`	json	time information

User metadata

This is the user metadata that can be set by the user which is parsed and have some actions on Pantavisor:

Key	Value	Description
`pvr-sdk.authorized_keys`	SSH pub key	set public key to get SSH access
`pvr-auto-follow.url`	URL	device will automatically pull every change in the device associated to that clone URL
`pantahub.log.push`	0 or 1	disable/enable log pushing to Pantacor Hub. Overrides PV_LOG_PUSH
`<config-key>`	config-value	override any configuration keys that allow RUN level
`<container>/<key>`	value	send user metadata that can be consumed by one of the containers

Pantavisor State Format (state.json)

A Pantavisor revision is defined by a single JSON object called `state.json`. It acts as a **virtual filesystem manifest** where every key represents a relative file path within the revision, and every value is either a nested configuration object or a SHA256 identifier for a binary artifact.

1. ROOT LEVEL (`state.json`)

These keys represent the files at the root of a revision.

Key	Value Type	Mandatory	Description
`#spec`	string	Yes	Parser version. Must be `"pantavisor-service-system@1"`.
`README.md`	string	No	Documentation for the revision in Markdown format.
`bsp/run.json`	BSP Manifest	Yes	Board Support Package configuration.
`bsp/drivers.json`	Drivers Manifest	No	Abstract driver mapping for the kernel.
`device.json`	Infrastructure Manifest	No	Unified physical storage and logical group definition.
`groups.json`	Groups Manifest	No	(Legacy) Logical container orchestration groups.
`disks.json`	Disks Manifest	No	(Legacy) Physical storage medium definitions.
`<container>/run.json`	Container Manifest	Yes	Individual container configuration.
`<container>/services.json`	Service Exports	No	Services exported to the xconnect mesh.
`_sigs/<container>.json`	Signature Manifest	No	Security signature for container artifacts.
`_config/<container>/<path>`	string	No	Injects data into `<path>` inside the container's rootfs.
`<any/other/path>`	string	No	SHA256 identifier for a binary artifact at that path.

---

2. BSP (`bsp/run.json`)

Defines the core system boot assets.

Key	Value Type	Description
`linux`	path string	Path to the Linux kernel image.
`initrd`	path string	Path to the Pantavisor initrd binary.
`modules`	path string	Path to the modules squashfs image.
`firmware`	path string	Path to the firmware squashfs image.
`fdt`	path string	Path to the Flattened Device Tree binary.
`fit`	path string	Path to a FIT image (replaces linux/initrd/fdt).
`rpiab`	path string	Path to a Raspberry Pi boot image.
`addons`	array of paths	List of CPIO addons to merge into the initrd rootfs.
`initrd_config`	path string	Custom configuration for the initrd process.

---

3. DRIVERS (`bsp/drivers.json`)

Maps abstract driver names to kernel modules based on hardware.

Key	Value Type	Description
`#spec`	string	Must be `"driver-aliases@1"`.
`all`	object	Default module mappings for all hardware.
`dtb:<name>`	object	Module mappings specific to a Device Tree model.
`ovl:<name>`	object	Module mappings specific to a DT Overlay.

Example module list:

```
"wifi": [ "cfg80211", "brcmfmac ${user-meta:wifi.opts}" ]
```

---

4. INFRASTRUCTURE (`device.json`)

The unified hardware and orchestration manifest.

Key	Value Type	Description
`disks`	array	List of Disk Definitions.
`groups`	array	List of Orchestration Groups.
`volumes`	object	List of Persistent Volumes for Pantavisor itself.

---

5. ORCHESTRATION (`groups.json`)

Defines how containers are grouped and started.

Key	Value Type	Default	Description
`name`	string	**Mandatory**	Unique logical name for the group.
`description`	string	empty	Human-readable description.
`status_goal`	enum	`STARTED`	Goal for all members: `MOUNTED`, `STARTED`, `READY`.
`restart_policy`	enum	`container`	Policy on failure: `system`, `container`.
`timeout`	integer	30	Seconds to wait for members to reach `status_goal`.
`auto_recovery`	object	none	Default auto-recovery for containers in this group. Inherited all-or-nothing by containers without their own `auto_recovery`.

---

6. STORAGE (`disks.json`)

Defines physical storage mediums.

Key	Value Type	Default	Description
`name`	string	**Mandatory**	Unique name used in `run.json` storage keys.
`type`	enum	**Mandatory**	`directory` , `dm-crypt-versatile` , `swap-disk` , `volume-disk` .
`path`	string	**Mandatory**	Path to block device or image file.
`format`	enum	`ext4`	Filesystem format (`ext3` , `ext4` , `swap`).
`provision`	string	empty	Provisioning source (e.g., `zram`).
`mount_target`	path	empty	Where to mount the disk on the host.
`mount_options`	string	empty	Comma-separated mount flags.
`format_options`	string	empty	Arguments for the `mkfs` command.
`default`	string	`"no"`	If `"yes"` , this disk is used for all volumes without a `disk` key.

---

7. CONTAINER (`<container>/run.json`)

Configures an individual container runtime.

Key	Value Type	Mandatory	Description
`#spec`	string	Yes	Must be `"service-manifest-run@1"` .
`name`	string	Yes	Logical name of the container.
`type`	enum	Yes	Runtime type (currently only `lxc`).
`config`	path string	Yes	Path to the LXC configuration file.
`root-volume`	path string	Yes	Path to the rootfs squashfs artifact.
`volumes`	array	No	Additional artifacts to mount as volumes.
`group`	string	No	Orchestration group name (from `device.json`).
`status_goal`	enum	No	Target state: `MOUNTED` , `STARTED` , `READY` .
`restart_policy`	enum	No	`system` (reboot on crash) or `container` (restart LXC).
`roles`	array	No	Capability roles: `mgmt` (control API access) or `nobody` .
`storage`	object	Yes	Persistence settings for rootfs paths.
`drivers`	object	No	Requirements: `required` , `optional` , or `manual` .
`services`	object	No	Service mesh requirements.
`logs`	array	No	Logger configurations.
`exports`	array	No	(Boolean flag in code) Marks container as an exporter.
`auto_recovery`	object	No	Auto-recovery configuration. If absent, inherited from group.

Auto-Recovery Object

Configures automatic restart behavior when a container crashes. See Auto-Recovery overview for the broader context.

Key	Value Type	Default	Description
`policy`	enum	`no`	Recovery policy: `no`, `always`, `on-failure`, `unless-stopped`. Note: the current implementation does not distinguish exit codes — `on-failure` behaves the same as `always`.
`max_retries`	integer	0	Maximum restart attempts. 0 = unlimited.
`retry_delay`	integer	0	Initial delay in seconds before first restart.
`backoff_factor`	number	1.0	Multiplier applied to `retry_delay` on each subsequent retry.
`reset_window`	integer	0	Seconds of continuous uptime after which the retry counter resets to 0.
`stable_timeout`	integer	0	Seconds the container must survive after reaching its status goal to be considered stable. Used to gate TESTING commit.
`backoff_policy`	string	`reboot`	Action after `max_retries` exhausted in steady state: `reboot`, `never`, or a duration string (`10min`, `1h`, `30s`).

Storage Object

Defines persistence for specific directories. Keys are paths relative to container root. * **`persistence`**: `permanent` (survives updates), `revision` (survives reboots), `boot` (volatile). * **`disk`**: Logical disk name from `device.json`.

Service Requirements

Under the `services` key in `run.json`. * **`required`** / **`optional`**: Arrays of service requirement objects. * `name`: Logical name of the service to find. * `type`: Protocol (`rest`, `dbus`, `unix`, `drm`, `wayland`). * `target`: Path where Pantavisor should inject the socket/resource. * `role`: Masquerade as this role when connecting. * `interface`: Protocol-specific identifier.

---

8. SERVICE MESH (`<container>/services.json`)

Declares services this container provides to others.

Key	Value Type	Description
`#spec`	string	Must be `"service-manifest-xconnect@1"`.
`services`	array	List of service objects (`name`, `type`, `socket`).

---

9. SECURITY (`_sigs/<container>.json`)

JWS-based artifact verification.

Key	Value	Description
`#spec`	`"pvs@2"`	Parser version.
`protected`	base64 string	Encoded headers including `alg`, `typ`, and `pvs` path filters.
`signature`	base64 string	The cryptographic signature of the protected header and payload.
`x5c`	array	(In protected) Certificate chain for verification.
`jwk`	object	(In protected) JSON Web Key for verification.

Pantavisor xconnect

The `pv-xconnect` service mesh facilitates efficient container-to-container and container-to-host interactions. It uses a plugin-driven architecture to inject resources (Unix sockets, D-Bus proxies, DRM nodes) into consumer containers on demand.

For information on how to inspect or manage the service mesh via the Pantavisor Control API, see the Pantavisor Control Socket reference.

ARCHITECTURE

To manage interactions between containers, a dedicated process called `pv-xconnect` handles the mediation logic via on-demand plugins. It runs as a managed daemon spawned by Pantavisor init and is enabled for all init modes (embedded, standalone, and appengine).

Core Responsibilities

- **Discovery & Reconciliation**: Periodically consumes an `xconnect-graph` from Pantavisor's `pv-ctrl` socket and maintains the state of active connects.
- **Plumbing**: Provides namespace-aware helpers to inject virtual resources (sockets/device nodes) inside the consumer's namespace.
- **Security**: Acts as the single point of truth for role-based access control.

SERVICE MANIFESTS

Provider (`services.json`)

A container declares the services it provides in a `services.json` file. This file must use the `#spec` format for identification by Pantavisor's parser.

Example `services.json` :

```
{
  "#spec": "service-manifest-xconnect@1",
  "services": [
    {
      "name": "network-manager",
      "type": "rest",
      "socket": "/run/network-manager/api.sock"
    },
    {
      "name": "system-bus",
      "type": "dbus",
      "socket": "/run/dbus/system_bus_socket"
    }
  ]
}
```

Consumer (`args.json` / `run.json`)

Containers that consume services define their requirements in `args.json` during creation (e.g., with `pvr app add --arg-json args.json`). These are then rendered into the final `run.json` manifest.

Example `run.json` requirement:

```
{
  "#spec": "service-manifest-run@1",
  "name": "my-app",
  "services": {
    "required": [
      {
        "name": "system-bus",
        "type": "dbus",
        "interface": "org.pantavisor.Example",
        "target": "/run/dbus/system_bus_socket"
      }
    ]
  },
  "type": "lxc"
}
```

- **`interface`** : Protocol-specific identifier (e.g., D-Bus interface name).
- **`target`** : The path where `pv-xconnect` should inject the proxied resource inside the consumer container.

MEDIATION PATTERNS

Raw Unix Sockets

Provides direct proxying of Unix Domain Sockets between containers. It supports high-performance features like FD passing (SCM_RIGHTS) and Shared Memory handles.

REST

Identity-injected HTTP over UDS. `pv-xconnect` automatically injects `X-PV-Client` and `X-PV-Role` headers into the first request, allowing the provider to identify the consumer.

D-Bus

Policy-aware proxy for the system bus. It performs **Role-Based Identity Masquerading**: 1. `pv-xconnect` intercepts the D-Bus SASL authentication phase. 2. It takes the **Role** from the link and looks up the corresponding **UID** in the provider container's `/etc/passwd`. 3. It replaces the consumer's identity with the resolved UID.

This allows the provider's standard `dbus-daemon` to enforce fine-grained permissions using standard XML policy files based on the assigned role.

DRM / Graphics

- **Master Role**: Injects `/dev/dri/cardX` for display servers (KMS access).
- **Render Role**: Injects `/dev/dri/renderDX` for accelerated applications.

Wayland

Mediates the Wayland protocol for isolated UI rendering, allowing a containerized compositor to serve multiple isolated clients.

TOOLS

pvcurl

A lightweight shell script wrapping `nc` for HTTP-over-Unix-socket communication. It is preferred in App Engine environments where standard `curl` might not be available.

pvcontrol

A high-level CLI wrapper around `pvcurl` for common Pantavisor control operations.

7.2 Platforms

7.2.1 Network Platforms

Inside our default pantavisor images all devices count with a base container, this container is the one managing the network interfaces. There is two possible containers in our stack: wifi-connect and alpine-conmman.

Configuration

Network platforms can by configured using the Linux Kernel cmdline, which is typically done from the bootloader menu. Using the pvnet_ prefix. The available values are:

- pvnet_ssid: The name of your network
- pvnet_pass: the network password
- pvnet_type: ethernet | wifi
- pvnet_security: type of security
- pvnet_ipv4: off | dhcp | network/netmask/gateway
- pvnet_ipv6: off | auto | network/prefixlength/gateway

All those values are optional, but the combination of them will allow your device to connect to the network, either via the ethernet port or the wifi interface if exists.

In order to configure it the available values for each key will be:

- **pvnet_type**: Mandatory. Other types than ethernet or wifi are not supported.
- **pvnet_security**: The security type of the network. Possible values are "psk" (WPA/WPA2 PSK), "ieee8021x" (WPA EAP), "none" and "wep".
- **pvnet_ipv4**: IPv4 settings for the service. If set to off, IPv4 won't be used. If set to dhcp, dhcp will be used to obtain the network settings. netmask can be specified as length of the mask rather than the mask itself.
- **pvnet_ipv6**: IPv6 settings for the service. If set to off, IPv6 won't be used. If set to auto, settings will be obtained from the network.

7.2.2 Development Platforms

pvr-sdk

PVR-SDK

Welcome to the pvr-sdk container reference! In the following chapters, you will find the reference pages for some of our Pantavisor local control tools: pantabox, pvtx and pvcontrol.

PANTABOX

pantabox -- manage your pantavisor system with a UI

SYNOPSIS

`pantabox`

DESCRIPTION

pantabox is a UI tool to manage your pantavisor system, that will allow to run every kind of actions to mantain your system.

You can read documentation for evey action in particular by using `man` and the command name. The commands names are:

```
pantabox-chpasswd
pantabox-claim
pantabox-edit
pantabox-edit-config
pantabox-edit-sshkeys
pantabox-edit-usermeta
pantabox-golocal
pantabox-goremote
pantabox-goto
pantabox-help
pantabox-info
pantabox-install
pantabox-make-factory
pantabox-reboot
pantabox-redeploy
pantabox-shutdown
pantabox-update
pantabox-view
```

EXAMPLES

```
man pantabox-info
```

COPYRIGHT

pantabox-chpasswd -- change password your system password

SYNOPSIS

`pantabox-chpasswd`

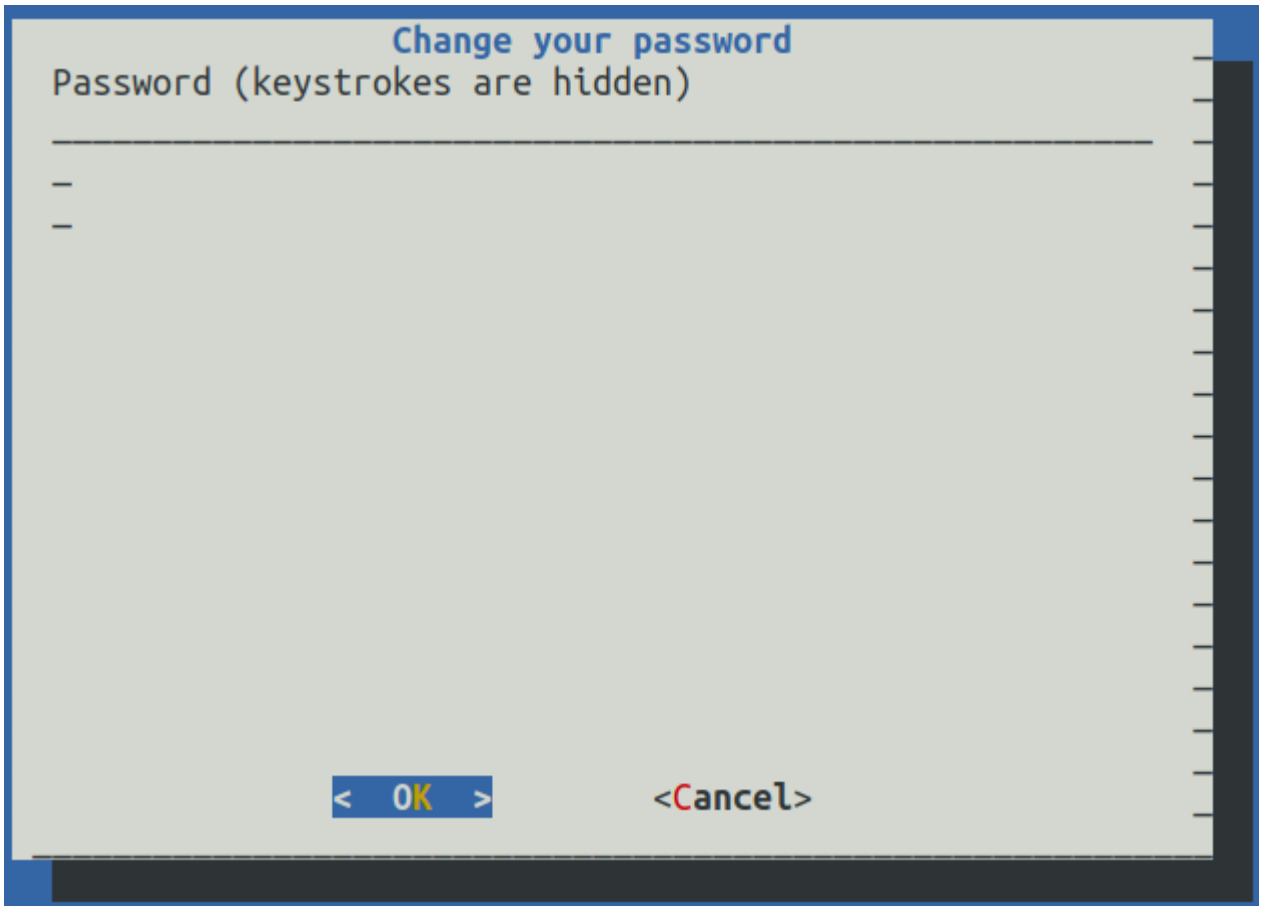
DESCRIPTION

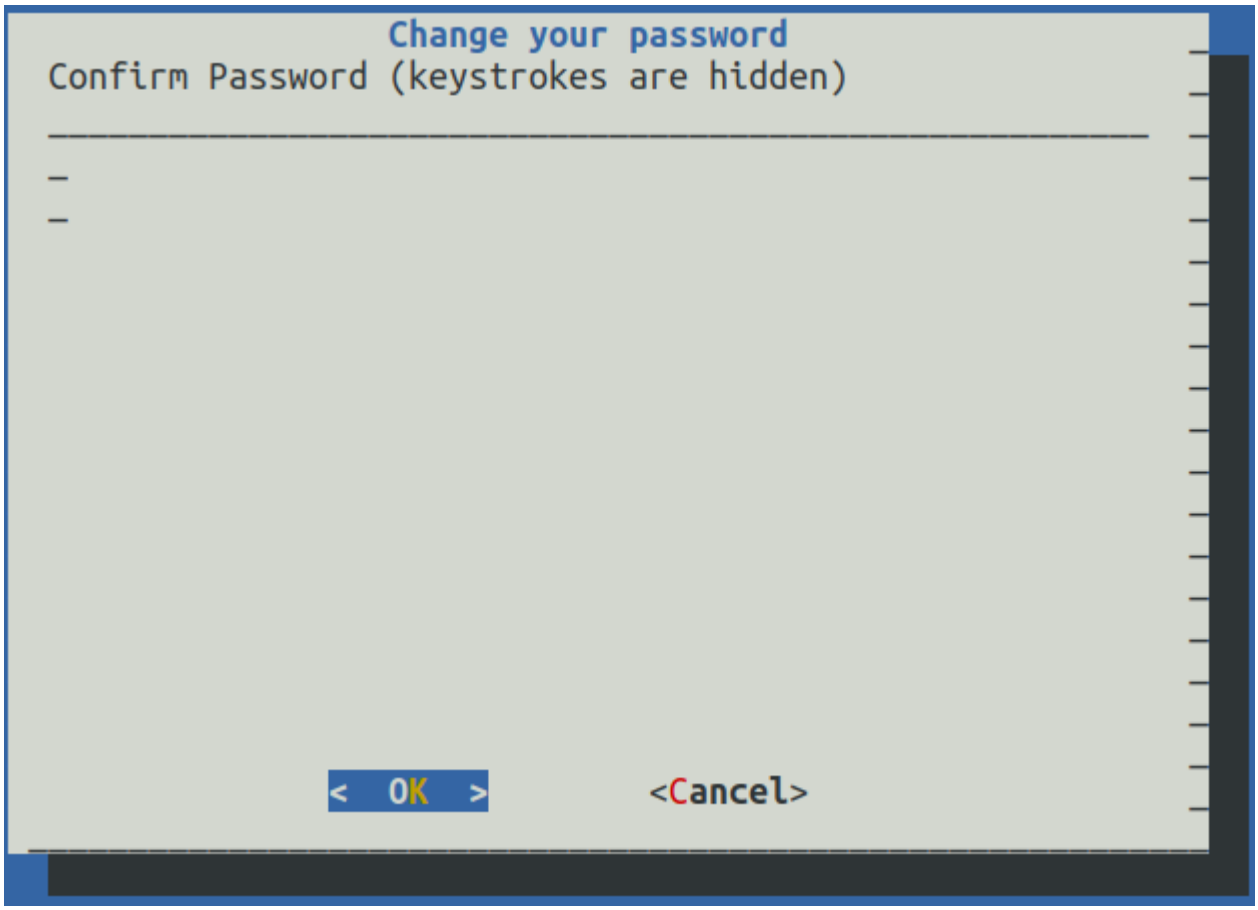
Pantabox uses a root user with a default password to directly login onto the device or to acccess it remotely via SSH at port:22. This command allows you to change the default password.

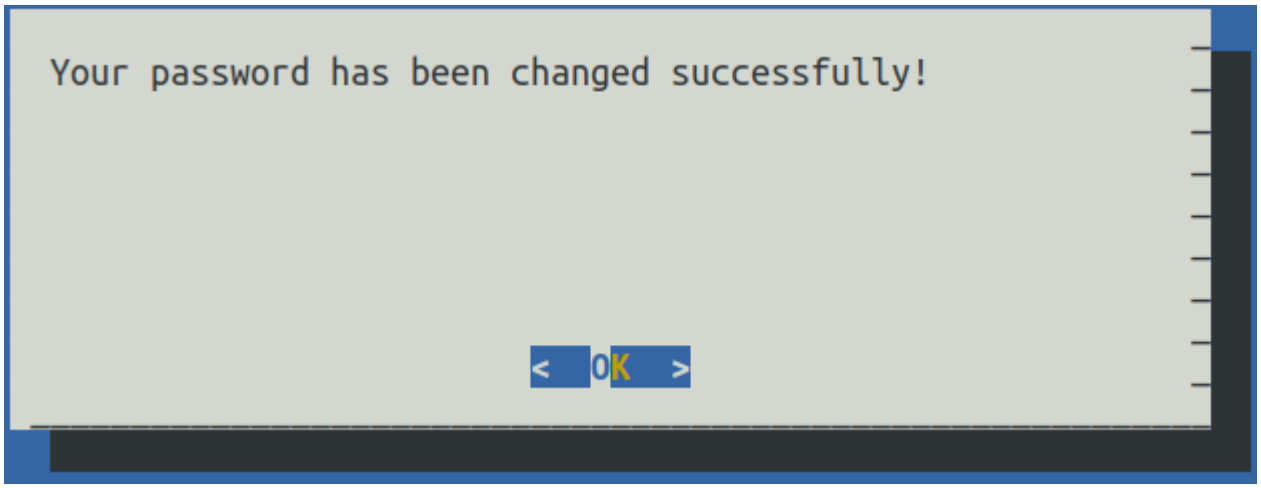
You'll be asked for the password and a confirmation. Your keystrokes are invisible while typing the password. The password cannot be empty and both the password and its confirmation needs to be validated as the same value.

After those validations, an OS call of chpasswd is executed.

EXAMPLES







pantabox-claim -- claim your device in a hub compatible cloud solution

SYNOPSIS

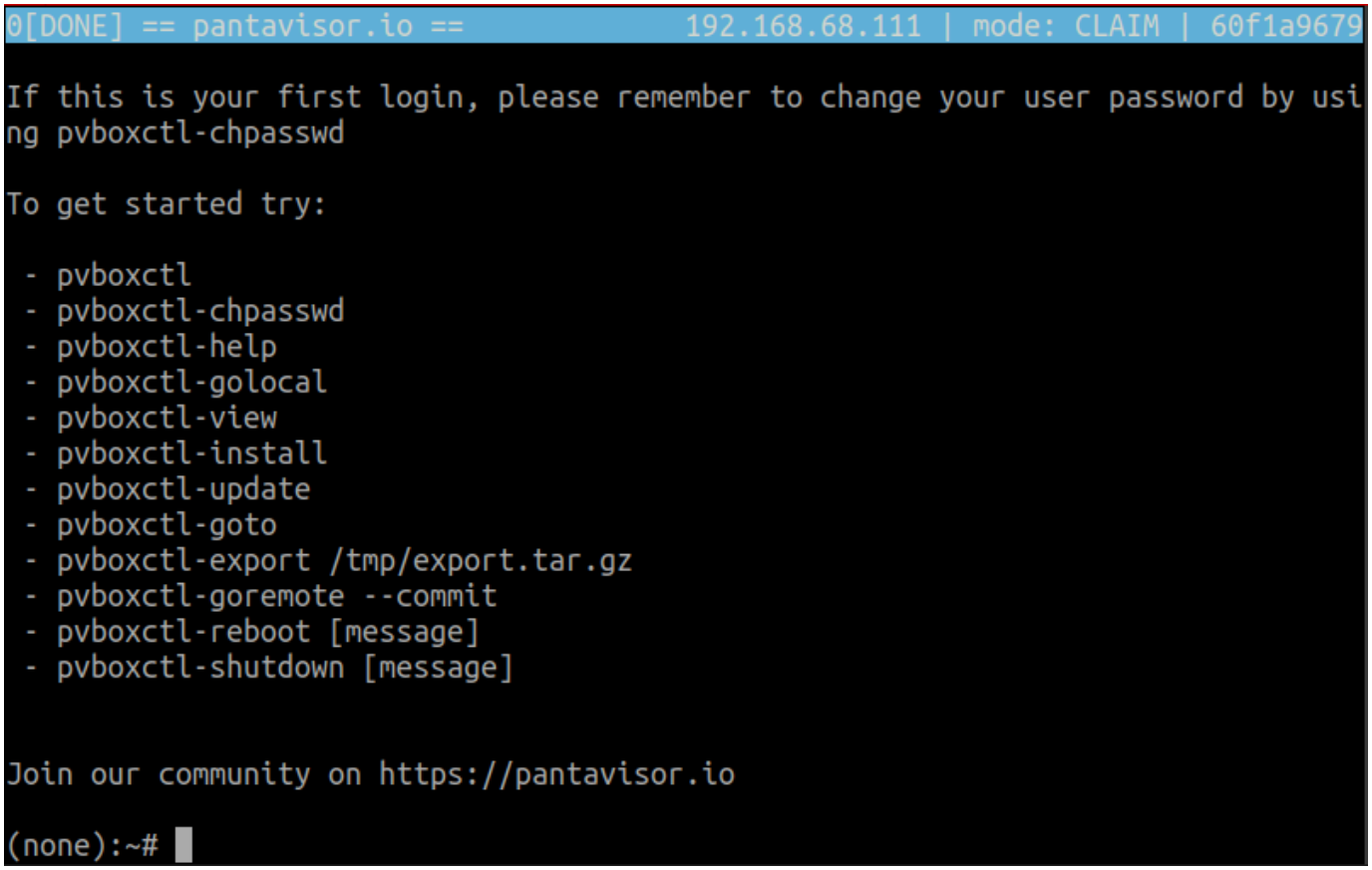
`pantabox-claim`

DESCRIPTION

This command allows you to claim the device to a Pantacor Hub compatible cloud service (the default is https://hub.pantacor.com).

You can only claim it if the device has never been claimed. The device also needs to be connected to Internet and in remote mode.

To check if your device is available to be claimed, look in the top right corner of the console where you should see `MODE: CLAIM` or `MODE: REMOTE`.



Notes:

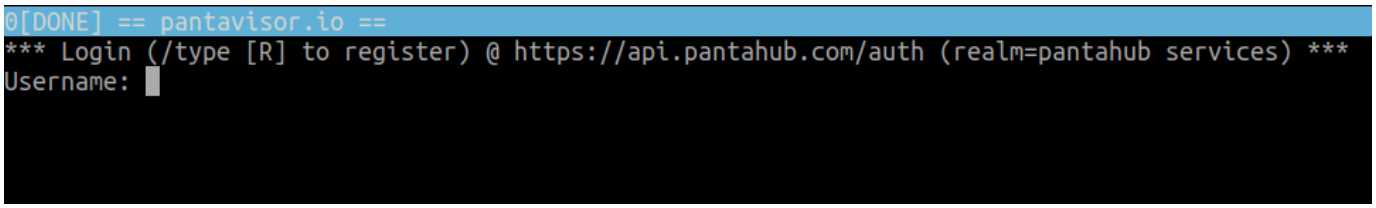
If you started your device in local mode and you have several updates before you claimed your device run, `make-factory` to force your current revision to 0 and causes the device to go remote. Afterwards, you will be able to claim your device.

EXAMPLES

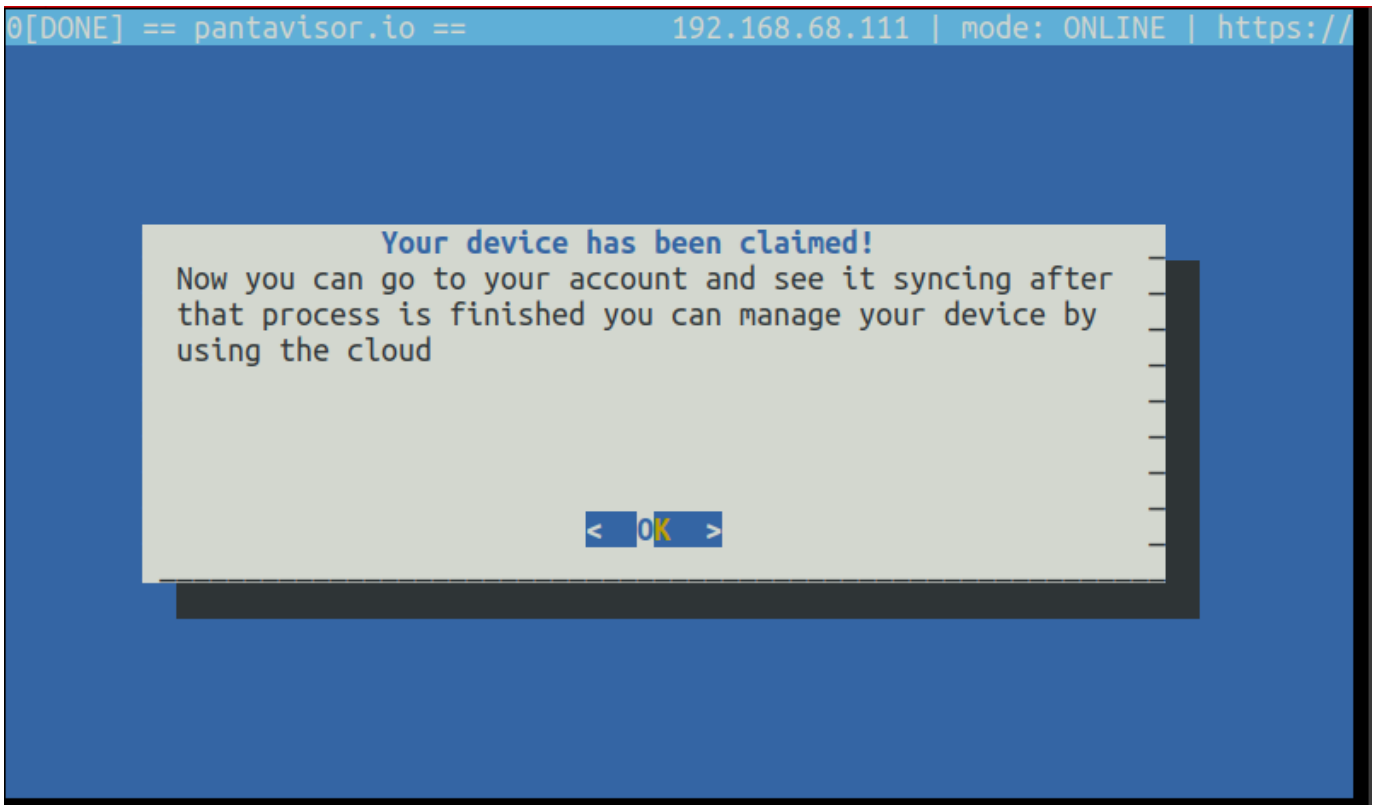
1. Start the claiming process with:

`pantabox-claim`

1. Login to your hub account. If you don't have an account create a new one inside the console.



1. If the claim process was successful, you will see `MODE: ONLINE` in the top right of the console and a success dialog.



pantabox-edit-config -- edit configuration files for a container

SYNOPSIS

`pantabox-edit-config`

DESCRIPTION

The `edit configuration` command allows you to edit the mounted volume for a specific container. In Pantavisor Linux every container has a `_config/container_name` folder. When you add new config files using the container filesystem structure, any existing config files inside the container are overwritten.

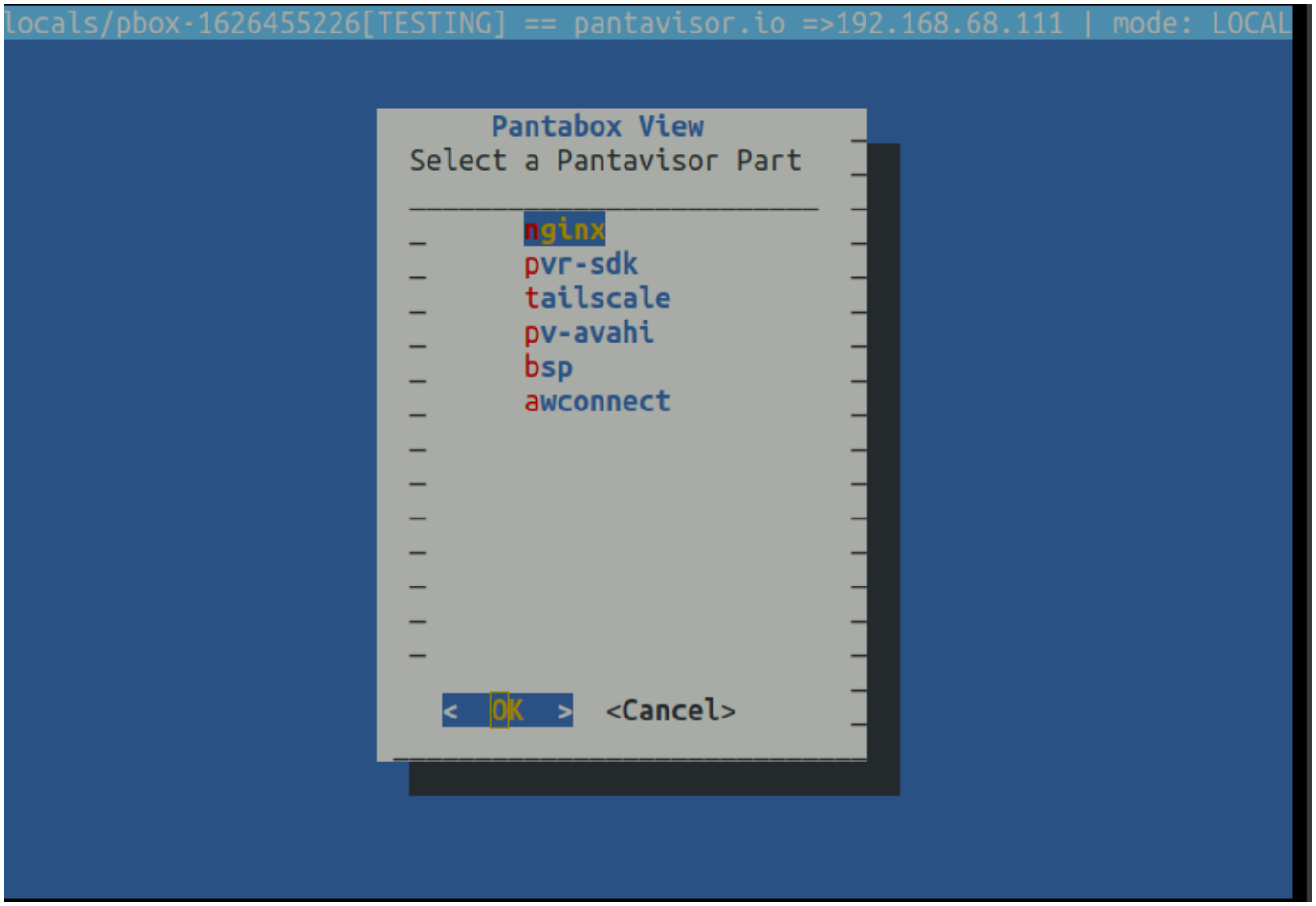
EXAMPLES

In this example, we'll configure an NGINX container. This could be an alpine base container with an NGINX configuration that lives inside `/etc/nginx/nginx.conf`. In this case, you can create a `_config` overlay with a new config file that overwrites any other inside the container.

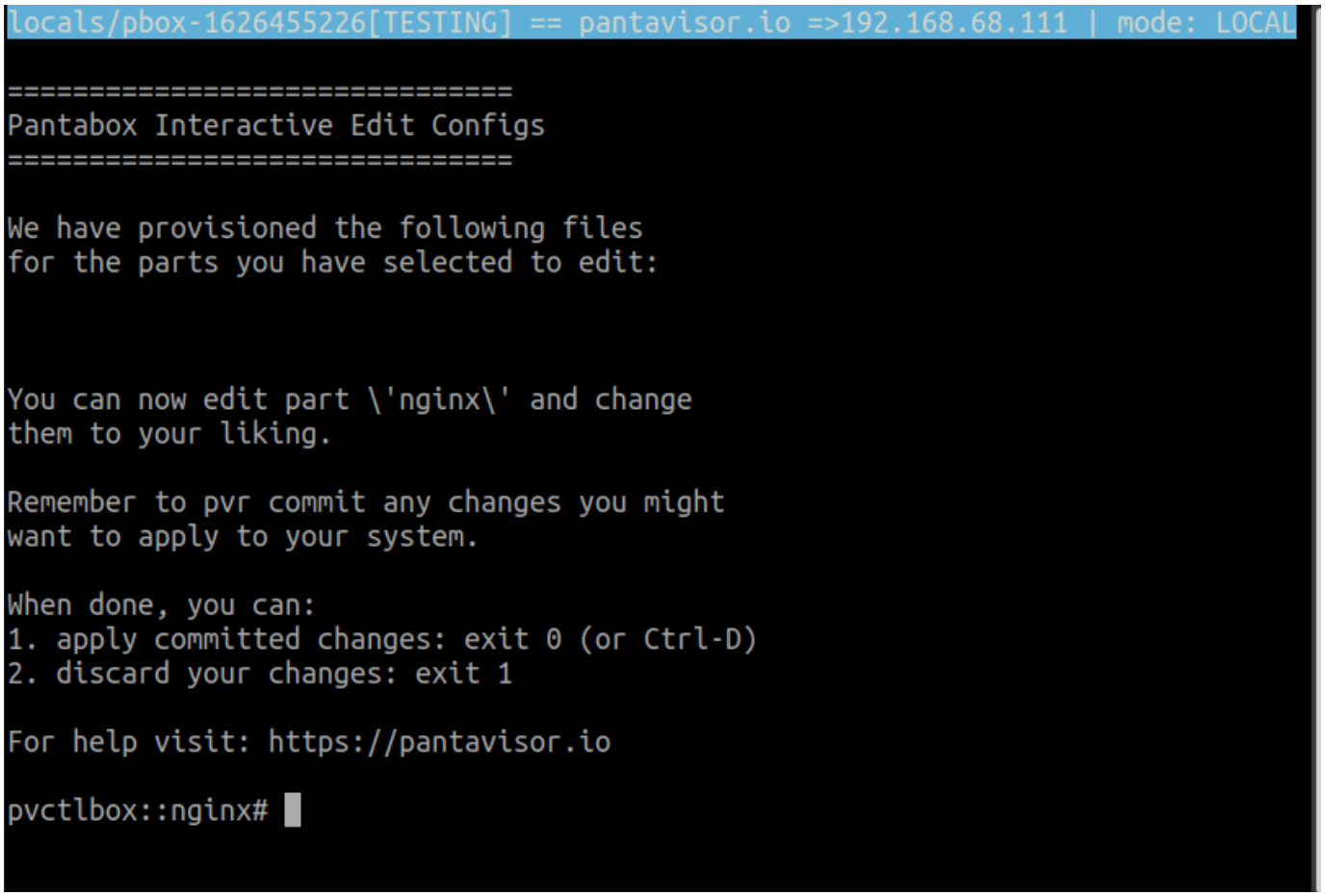
That config file is located at `_config/nginx/etc/nginx/nginx.conf`. The file can change the configuration of NGINX so that every time you make a change, it creates a new revision. It does this without having to change the base NGINX container or without you having to create and maintain another volume.

To see how this works:

1. Run `pantabox-edit-config` and select `nginx`



This launches a shell environment inside the Pantavisor configuration part.



This environment allows you to create files and folders inside `_config/nginx/` and when you're ready to add and commit those changes.

1. In this example, let's create the NGINX config file `_config/nginx/etc/nginx/nginx.conf`

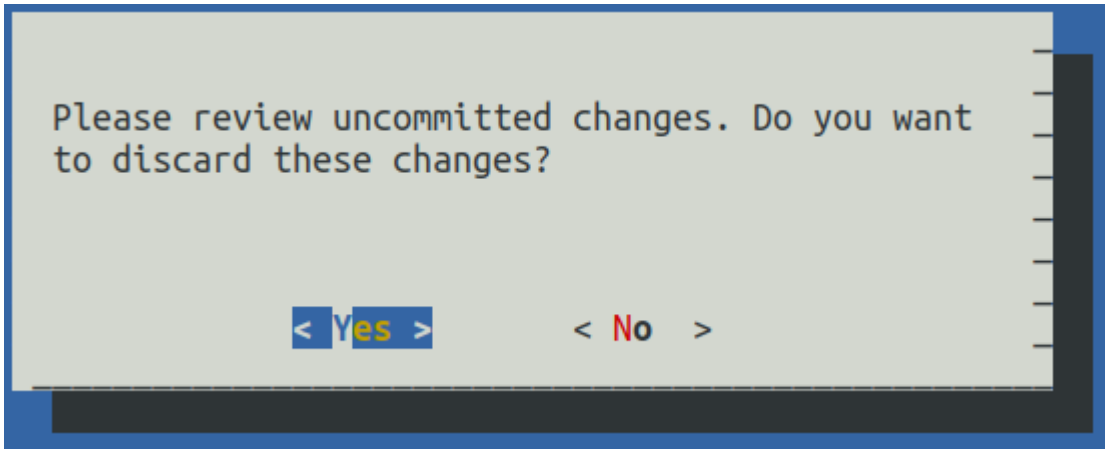
```
locals/pbox-1626455226[UPDATED] == pantavisor.io ==
  GNU nano 5.4                                                  _config/nginx/etc/nginx/nginx.conf
user  nginx;
worker_processes  2;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

1. After creating and editing the `nginx.conf` file you can add and commit those changes:

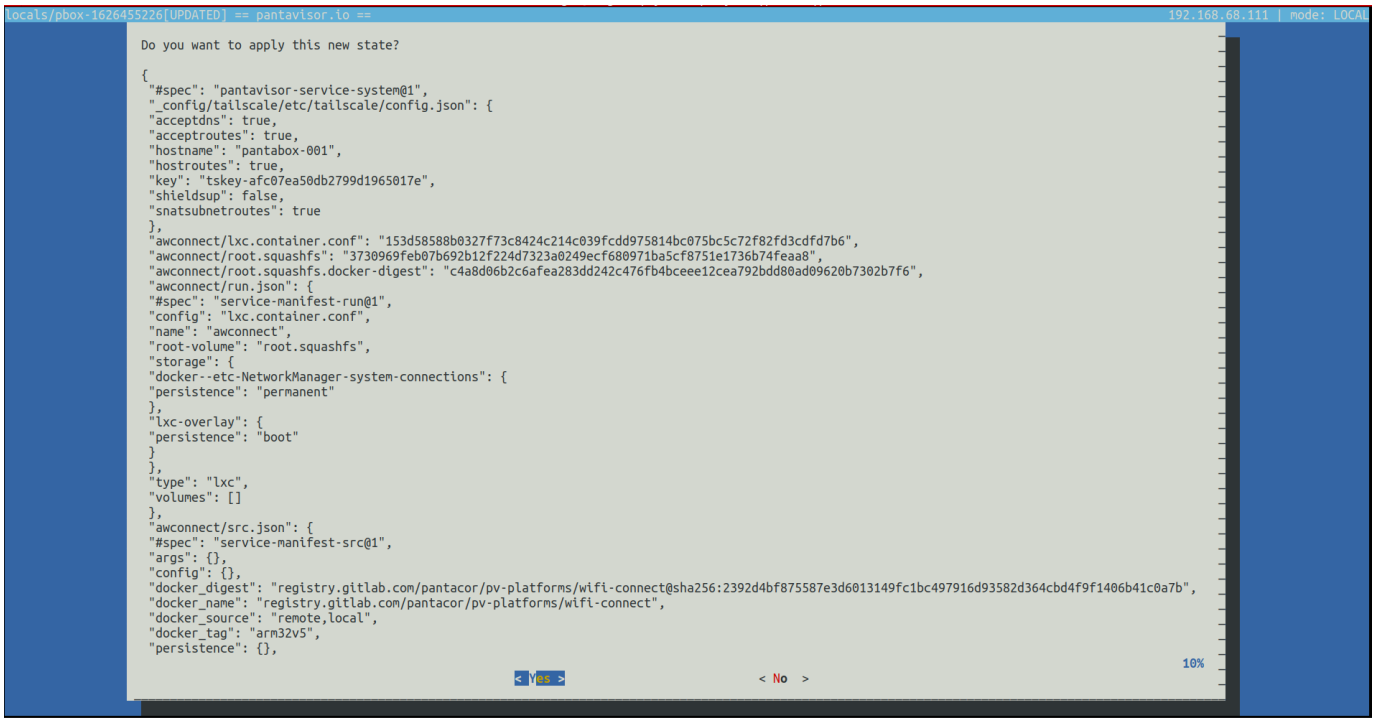
`pvr add .` `pvr commit` `exit 0`

```
locals/pbox-1626455226[UPDATED] == pantavisor.io ==
pvctlbox::nginx# pvr add .
pvctlbox::nginx# pvr commit
Adding _config/nginx/etc/nginx/nginx.conf
pvctlbox::nginx# exit 0
```

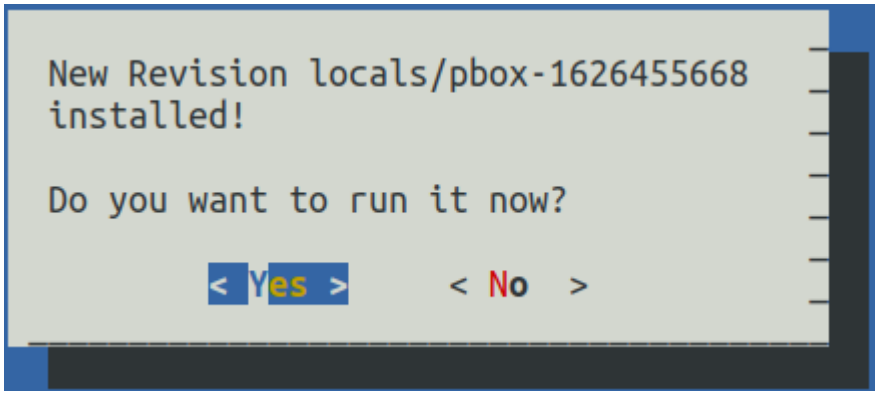
Exit the interactive edit console, and press Ok to continue.



Now you can see the new `state.json` file generated by the changes you made on the config. Confirm the changes to apply them to the device.



After the changes have been applied, you can run the new revision.



COPYRIGHT

pantabox-edit -- edit a pantavisor container configuration

SYNOPSIS

`pantabox-edit`

DESCRIPTION

Edit a Pantavisor container configuration. Every Pantavisor container has the following configuration files for configuration:

```
pvr-sdk/
    lxc.container.conf
    root.squashfs
    root.squashfs.docker-digest
    run.json
    src.json
```

- `src.json` this file defines the source container for Pantavisor. You can configure volumes and LXC containers if you needed.
- `run.json` defines how Pantavisor runs the container. This is generated from the `src.json` when you install an app from a Docker container.
- `root.squashfs` is the result of converting the Docker layers to be used by Pantavisor.
- `root.squashfs.docker-digest` this is the digest of the Docker container. It consists of the full Docker URL and SHA digest.
- `lxc.container.conf` saves the autogenerated configuration for the LXC container. If your Pantavisor configuration was generated directly from a docker application, don't modify the `lxc.container.conf` directly, use the `src.json` to add LXC configurations instead.

COPYRIGHT

pantabox-edit-sshkeys -- edit your ssh keys to access the running containers

SYNOPSIS

`pantabox-edit-sshkeys`

DESCRIPTION

Because Pantavisor Linux runs every part of your system in containers, you need a mechanism for managing SSH connections. `pvr-sdk` is the container that handles SSH connections for all containers running on the device.

SSH keys are converted to user-meta data for the device which then makes them available to all running containers.

When you add or edit your SSH keys, it opens the system editor (by default this is nano). A public SSH key should be added line by line.

EXAMPLES

`pantabox-edit-ssh` opens the system editor, nano.

COPYRIGHT

pantabox-edit-usermeta -- edit pantavisor user-meta data

SYNOPSIS

`pantabox-edit-usermeta`

DESCRIPTION

This command allows you to edit the user-meta data used by Pantavisor Linux. The user-meta data is safe in a `.json` format of key:value

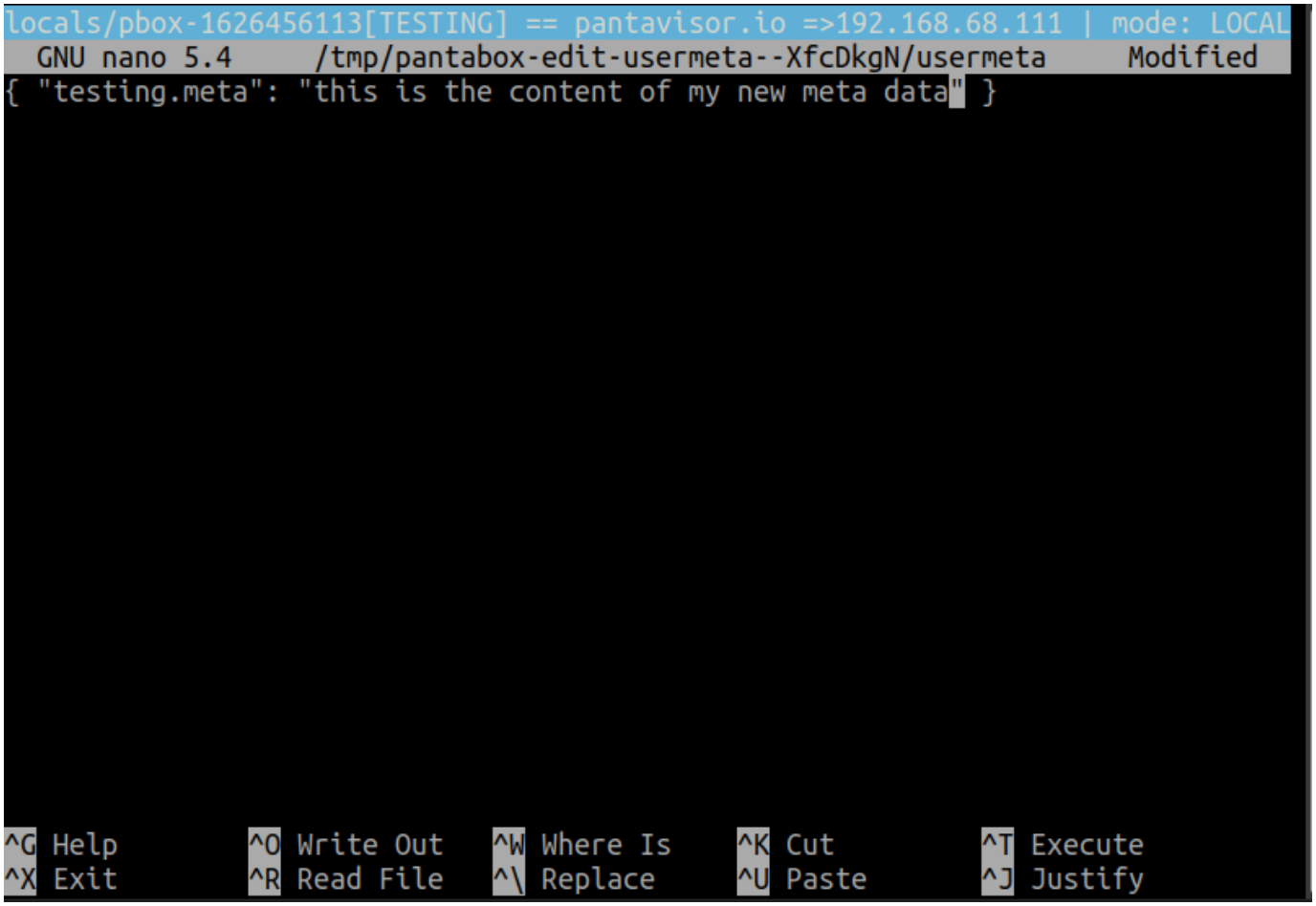
Pantavisor creates files with the key name and value from the `.json` file and keeps them inside the user-meta folder.

Those files reside at: `/pantavisor/user-meta/key`

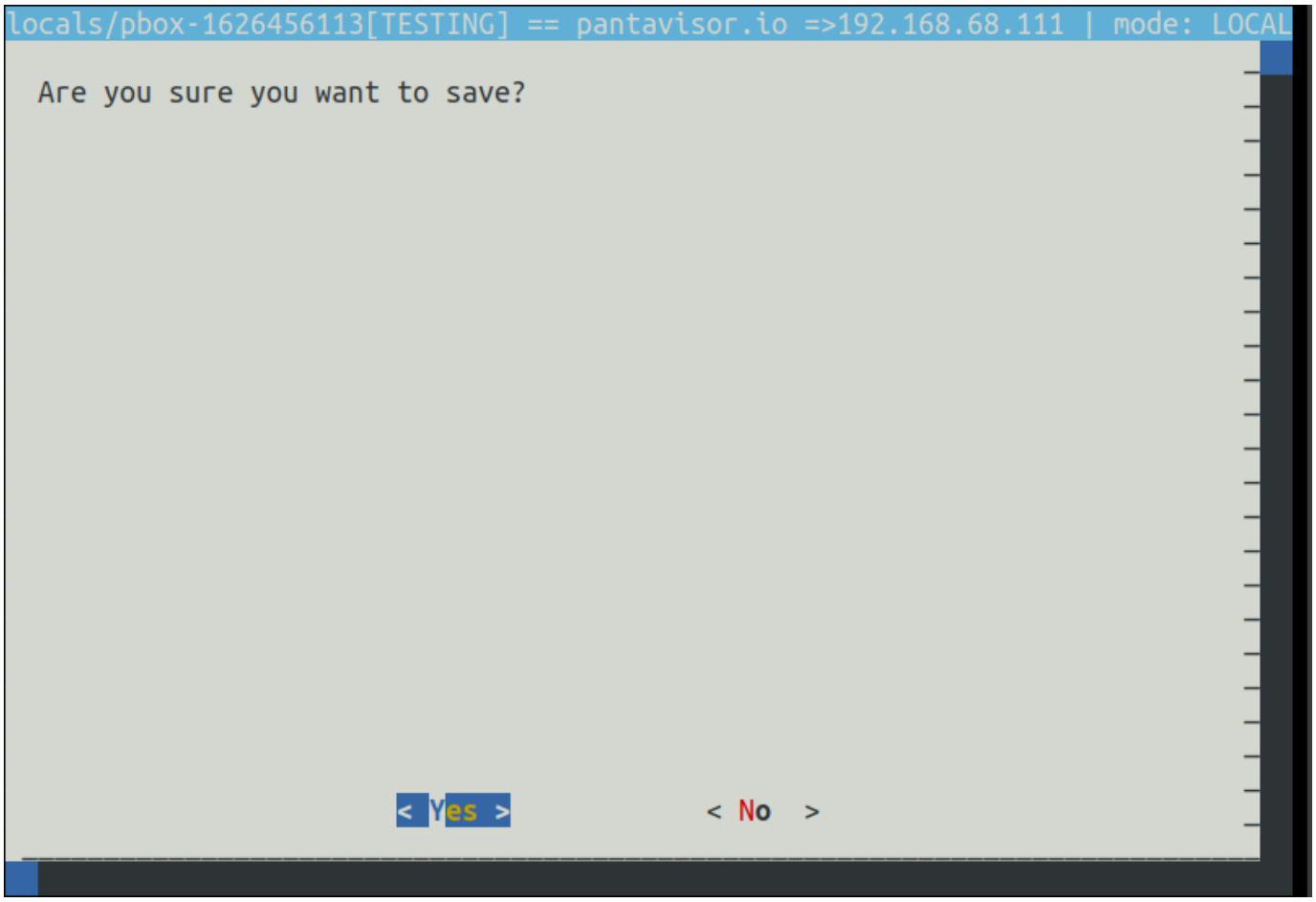
EXAMPLES

To view an example:

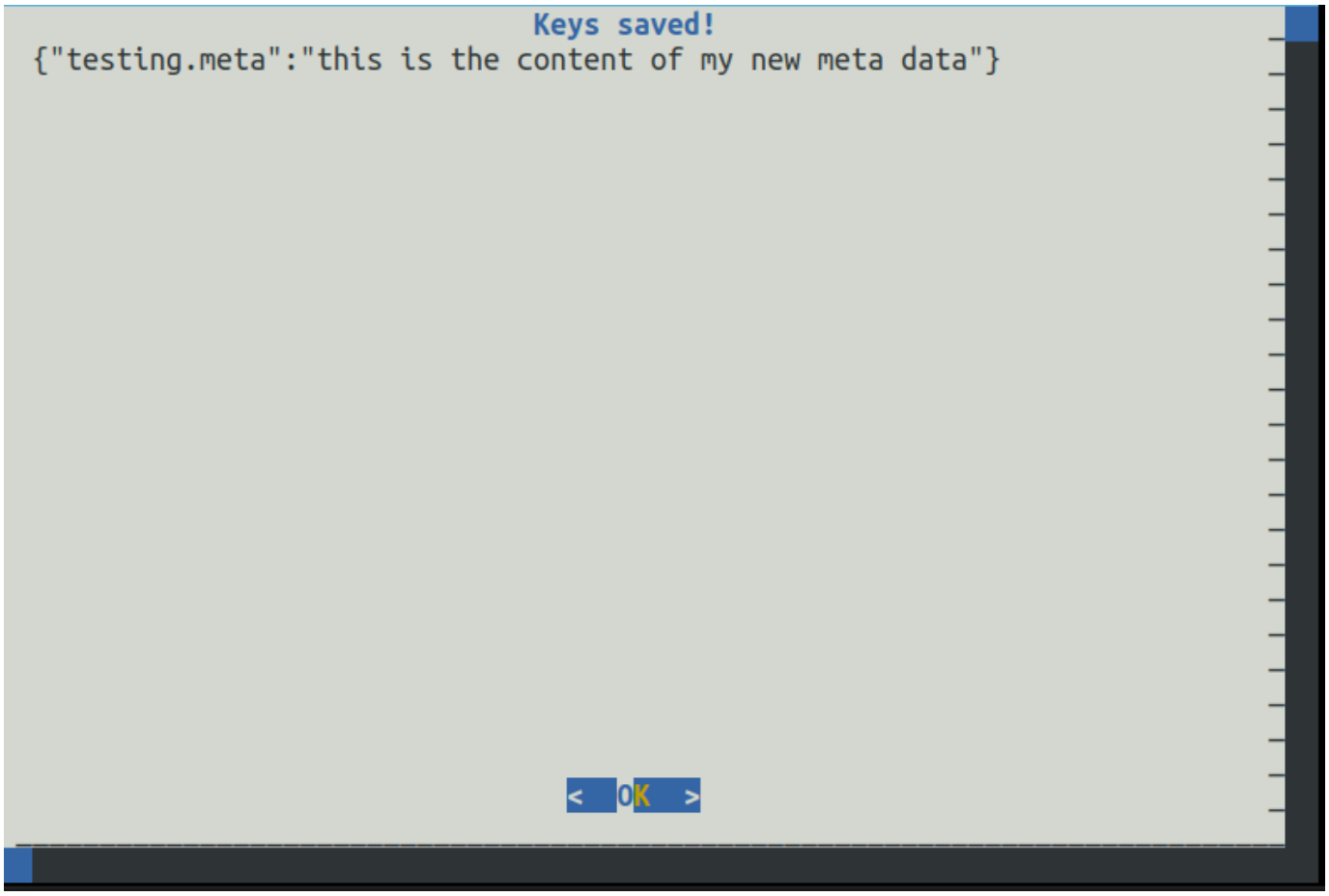
- Run `pantabox-edit-usermeta` to bring up the nano editor.



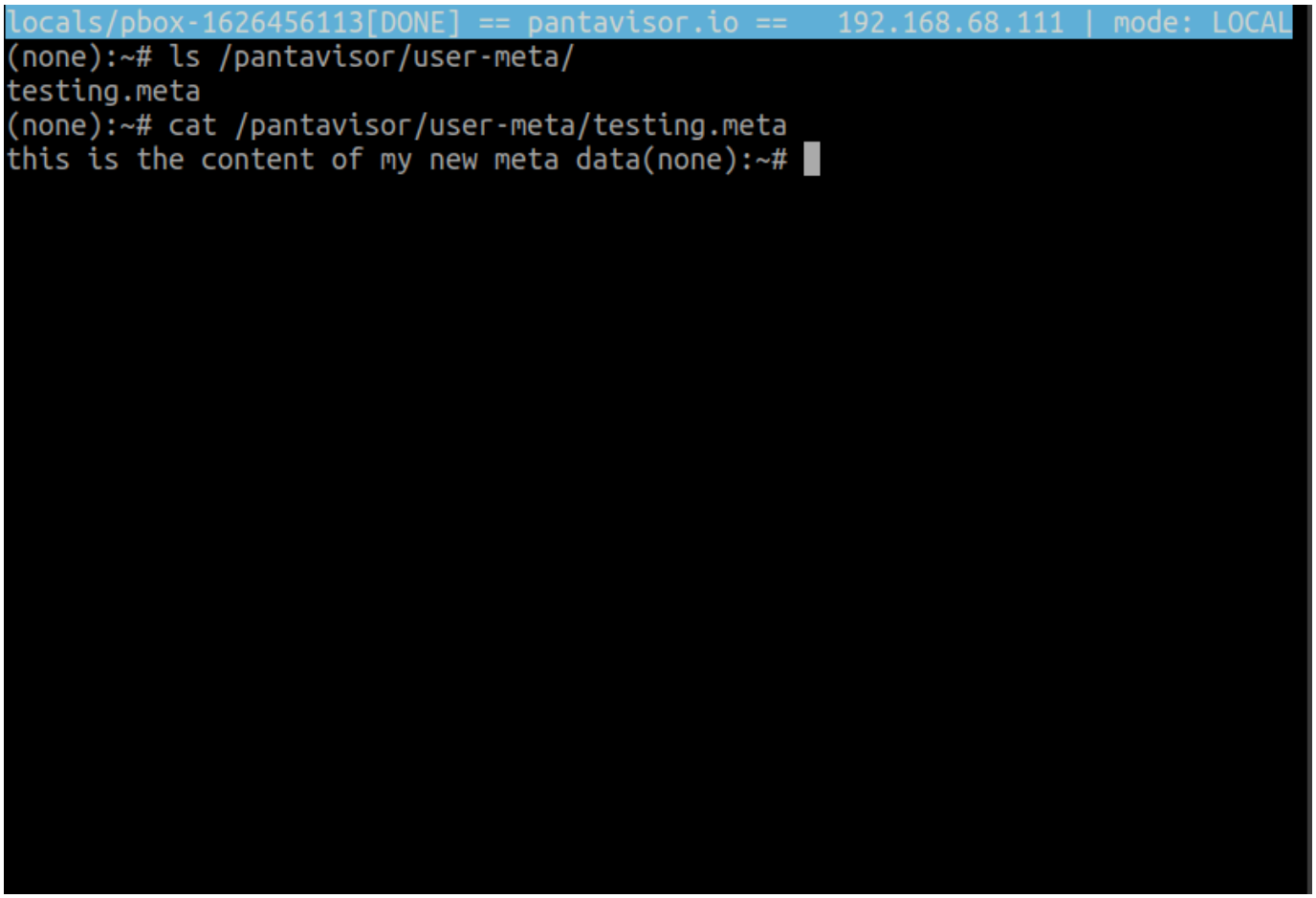
- Edit the `.json` file by adding your user-meta key and values.



- Confirm you changes.



- View them in the filesystem as files in the `/pantavisor/user-meta` folder



pantabox-golocal -- swith your device from cloud base functionality to local

SYNOPSIS

`pantabox-golocal`

DESCRIPTION

Switches your device from consuming user-meta and revisions from the cloud to consuming all the Pantavisor bits from local mode.

And once you're in local mode, you will able to install, update and configure everything for Pantavisor without the a cloud service.

EXAMPLES COPYRIGHT

pantabox-goremote -- switch your device from local functionality to a cloud base one

SYNOPSIS

`pantabox-goremote`

DESCRIPTION

This switches your context from local to remote. By going remote, the device consumes and syncronizes user-meta and revisions from a cloud service.

To perform this action, you need a claimed device. If your device is not claimed, you must first run `make-factory` before you can claim your device.

EXAMPLES COPYRIGHT

pantabox-goto -- run a specific revision on your device

SYNOPSIS

`pantabox-goto` [REVISION_NUMBER]

DESCRIPTION

Tells Pantavisor to switch between device revisions. This operation restarts all of the containers with the selected state of the revision.

EXAMPLES

SOME EXAMPLES HERE

COPYRIGHT

pantabox-info -- get information about your pantavisor system

SYNOPSIS

`pantabox-info`

DESCRIPTION

Shows you information about the system, including the mode that Pantavisor is running, the device-meta data, the network interfaces and the user-meta data.

EXAMPLES

`pantabox-info`

The output is shown inside a dialog, but the content and is similar to the following:

```
https://pantavisor.io

Mode:
local

Network Interfaces:
 lo.ipv4: 127.0.0.1
 eth0.ipv4: 192.168.68.111
 lxcbr0.ipv4: 10.0.3.1
 lo.ipv6: ::1
 eth0.ipv6: fe80::ac03:84d:458a:c434%eth0
 tailscale0.ipv6: fe80::6448:b31d:8176:d81c%tailscale0

Device Info:
 pantahub.claimed: 0
 pantahub.online: 0
 pantavisor.mode: local
 pantavisor.revision: locals/pbox-1626456113
 pantavisor.dtmodel: Raspberry Pi 4 Model B Rev 1.2
 pantavisor.version: 014-rc8-13-ga9eb1b6-210708-1808ba3
 pantavisor.arch: aarch64/64/EL

User Meta:
 testing.meta: this is the content of my new meta data
```

COPYRIGHT

pantabox is Copyright (c) 2021 by Pantacor Ltd.

pantabox-install -- install a new container in your system

SYNOPSIS

```
pantabox-install
```

DESCRIPTION

This allows you to install new containers onto your Pantavisor system. You can install containers from the following three places:

1. By directly downloading a Docker container using the Docker name and tag of the container.
2. From another Pantavisor-enabled device kept in the cloud.
3. From a tarball that's been exported from Pantavisor-enabled device.

EXAMPLES COPYRIGHT

pantabox-make-factory -- convert the running revision into your initial revision

SYNOPSIS

`pantabox-make-factory`

DESCRIPTION

Convert the running revision into the initial state of the device and then switch the device to remote mode. This operation should be run on a unclaimed device or one that has never connected to a hub cloud service.

EXAMPLES COPYRIGHT

pantabox-reboot -- reboot your pantavisor device

SYNOPSIS

```
pantabox-reboot
```

DESCRIPTION

Sends a command to Pantavisor to reboot the whole system.

EXAMPLES COPYRIGHT

pantabox-redeploy -- deploy your current state to a Pantavisor compatible cloud

SYNOPSIS

`pantabox-redeploy`

DESCRIPTION

Deploys the current state to a Pantavisor compatible cloud. If the revision already exists in the cloud, it will be redeployed.

EXAMPLES COPYRIGHT

pantabox-shutdown -- shutdown your system

SYNOPSIS

`pantabox-shutdown` [message]

DESCRIPTION

Sends a signal to Pantavisor to shutdown your system.

EXAMPLES

SOME EXAMPLES HERE

COPYRIGHT

Pantabox is Copyright (c) 2021 by Pantacor Ltd.

pantabox-update -- update any container running in your system

SYNOPSIS

`pantabox-update`

DESCRIPTION

Update any container running in your system. The command downloads the must recent version of the container using the specified tag and container source URL in the `src.json` kept inside the container configuration folder.

EXAMPLES COPYRIGHT

pantabox is Copyright (c) 2021 by Pantacor Ltd.

pantabox-view -- view configuration the details for all running containers in your system

SYNOPSIS

`pantabox-view`

DESCRIPTION

View the configuration of a container, including all of the fields inside the `src.json`, the configuration volumes, the type of LXC overlay, Pantavisor arguments and source configurations.

EXAMPLES

`pantabox-view`

```
Module Overview: nginx

Source (Docker)
- Name: nginx:latest
- Digest: nginx@sha256:1c70a669bbf07f9862f269162d776c35144b116938d1becb4e4676270cff8f75

Storage
- lxc-overlay: boot

Args
PV_RUNLEVEL: app

Source Configs
{}

Config Files
- /etc/nginx/nginx.conf
```

COPYRIGHT

PVTX

PVTX Rest API

PVTX is a tool to manage pantavisor devices using only tar files exported from PVR CLI. Allowing to manage without using any cloud service or docker container sources.

This rest api exposes the same actions availables on the pvtx CLI.

How to use it

We are going to see two examples of how to use pvtx API, one were we replace the whole state of the device, with the package from a pvexport file from hub and another were we add and remove parts from the current running revision.

Create a new transaction and apply a device export from hub

First go to hub.pantacor.com login and search for the device go to the export tab and download the pvexport file of the revision you want to apply using pvtx.

Let's use this `pantahub-ci/rpi64_5_10_y_pvwc_connman_latest` as example:

In order to download the pvexport you need to go to https://hub.pantacor.com/u/pantahub-ci/devices/65cbc800ddd1b00009d70ca5/export

And click in the export all button, that will generate a `rpi64_5_10_y_pvwc_connman_latest.tar.gz` file

The we could start using the pvtx api to apply that from scratch.

```
# start a new transaction
$ curl -X POST __ORIGIN__/cgi-bin/pvtx/begin?empty=true
{"#spec": "pantavisor-service-system@1"}
```

add rpi64_5_10_y_pvwc_connman_latest.tar.gz

```
$ curl -X PUT __ORIGIN__/cgi-bin/pvtx/add --data-binary @rpi64_5_10_y_pvwc_connman_latest.tar.gz
{
  "#spec":"pantavisor-service-system@1"
  ...
  ...
  ...
}
```

commit the state and save the revision name

```
$ curl -X POST __ORIGIN__/cgi-bin/pvtx/commit
locals/pvtx-1721068626-f9ee123b-913
```

run the revision created

```
$ curl -X POST __ORIGIN__/cgi-bin/pvtx/run?rev=locals/pvtx-1721068626-f9ee123b-913
{"status": "ok"}
```

after the run command the device will restart and run the new revision, and we can check the status using:

```
curl -X GET __ORIGIN__/cgi-bin/pvtx/status?rev=locals/pvtx-1721068626-f9ee123b-913
{
  "progress": {
    "status": "DONE",
    "status-msg": "Update finished, revision set as rollback point",
    "progress": 100,
    "data": "0"
  },
  "rev": "pvtx-1721068626-f9ee123b-913"
}
```

Modify the running revision

We start a new transaction using the current running revision as base state.

```
# start a new transaction
$ curl -X POST __ORIGIN__/cgi-bin/pvtx/begin
{
  "#spec":"pantavisor-service-system@1"
  ...
```

```
  ...
  ...
}
```

then we add a container

```
$ curl -X PUT __ORIGIN__/cgi-bin/pvtx/add --data-binary @container.tar.gz
{
  "#spec":"pantavisor-service-system@1"
  ...
  ...
  ...
}
```

or remove a container

```
$ curl -X PUT __ORIGIN__/cgi-bin/pvtx/remove?part=CONTAINER_NAME
{
  "#spec":"pantavisor-service-system@1"
  ...
  ...
  ...
}
```

after that we can commit the state and save the revision name

```
$ curl -X POST __ORIGIN__/cgi-bin/pvtx/commit
locals/pvtx-1721068626-f9ee123b-913
```

after theses steps we need to run the created revision.

```
$ curl -X POST __ORIGIN__/cgi-bin/pvtx/run?rev=locals/pvtx-1721068626-f9ee123b-913
{"status": "ok"}
```

run command will make the device to restart and run the new revision, and we can check the status using:

```
curl -X GET __ORIGIN__/cgi-bin/pvtx/status?rev=locals/pvtx-1721068626-f9ee123b-913
{
  "progress": {
    "status": "DONE",
    "status-msg": "Update finished, revision set as rollback point",
    "progress": 100,
    "data": "0"
  },
  "rev": "pvtx-1721068626-f9ee123b-913"
}
```

Endpoints documentation

The api is available by default on `__ORIGIN__/cgi-bin/pvtx` the access is public it doesn't need password or credentials.

logs

The api has an endpoint to give you the logs by source, if source is not defined the returned logs are going to be the logs from the pantavisor.

```
curl __ORIGIN__/cgi-bin/logs
```

The available query string parameters are:

- rev: the revision number
- source: the source will be the platform name
- tail: (true/false) activate the tail of logs
- follow: (true/false) the follow parameters sents the logs as stream, only works if the tails arguments is true
- tailn: same as tail -n parameter, only applies if the follow and tail parameters are set

show

The show endpoint show the current transaction running on pvtx. If doesn't exist any transaction will return and error.

```
curl __ORIGIN__/cgi-bin/pvtx/show
```

If doesn't have any transaction started will return `HTTP/1.1 400 Bad Request` with this body

```
{
  "error": "No active transaction.",
  "transaction": null
}
```

If there is a transaction will return the full state json of that transaction like this one.

Example response:


```
{
  "#spec": "pantavisor-service-system@1",
  "_config/pvr-sdk/etc/pvr-sdk/config.json": {
    "httpd": {
      "listen": "0.0.0.0",
      "port": "12368"
    }
  },
  "_hostconfig/pvr/docker.json": {
    "platforms": [
      "linux/arm64",
      "linux/arm"
    ]
  },
  "bsp/addon-plymouth.cpio.xz4": "beae6a7bb235916cac52bcfece64c30615cded8c4c640e6941e7ecabe53b4920",
  "bsp/build.json": {
    "altrepogroups": "",
    "branch": "master",
    "commit": "7d5ba78761e4d880c3403d642187bfdc93e49683",
    "gitdescribe": "014-rc9-12-g7d5ba78",
    "pipeline": "355967648",
    "platform": "rpi64",
    "project": "pantacor/pv-manifest",
    "pvrversion": "pvr version 022-15-g26ebb342",
    "target": "arm-rpi64",
    "time": "2021-08-19 15:08:44 +0000"
  },
  "bsp/firmware.squashfs": "c968a674d12258f00f4d9251637065a04abf7f95285308bfca7e4f6ccf9de7c5",
  "bsp/kernel.img": "b59438e4cb0db11689601e7e26e8dc6dad0b5007072edbf10e886a8dd51d2397",
  "bsp/modules.squashfs": "385f04fca555912f8655018f97cd09a2502fdd79afc14cd5fd57682d0a2cf4e0",
  "bsp/pantavisor": "405f907a1d1d086d89808c6fc691bf7d169e6e38b24c0110c7b950c4dda2742d",
  "bsp/run.json": {
    "addons": [
      "addon-plymouth.cpio.xz4"
    ],
    "firmware": "firmware.squashfs",
    "initrd": "pantavisor",
    "initrd_config": "",
    "linux": "kernel.img",
    "modules": "modules.squashfs"
  },
  "bsp/src.json": {
    "#spec": "bsp-manifest-src@1",
    "pvr": "https://pvr.pantahub.com/pantahub-ci/arm_rpi64_bsp_latest#bsp"
  },
  "network-mapping.json": {},
  "pvr-sdk/lxc.container.conf": "a69205914de2e8b95270f94591d5c015796590da5273db35ef6b2ed40631fcca",
  "pvr-sdk/root.squashfs": "896d66166794aa11f14ffe3c8fddc80a0563d85bae086d25dce69836d8eb0468",
  "pvr-sdk/root.squashfs.docker-digest": "35e1d5180e95445a7effdd44e8ef09405e2051d6c50c6b71965469fabc768368",
  "pvr-sdk/run.json": {
    "#spec": "service-manifest-run@1",
    "config": "lxc.container.conf",
    "name": "pvr-sdk",
    "root-volume": "root.squashfs",
    "storage": {
      "docker--etc-dropbear": {
        "persistence": "permanent"
      },
      "docker--etc-volume": {
        "persistence": "permanent"
      },
      "docker--home-pantavisor-.ssh": {
        "persistence": "permanent"
      },
      "docker--var-pvr-sdk": {
        "persistence": "permanent"
      },
      "lxc-overlay": {
        "persistence": "boot"
      }
    },
    "type": "lxc",
    "volumes": []
  },
  "pvr-sdk/src.json": {
    "#spec": "service-manifest-src@1",
    "args": {
      "PV_LXC_EXTRA_CONF": "lxc.mount.entry = /volumes/_pv/addons/plymouth/text-io var/run/plymouth-io-sockets none bind,rw,optional,create=dir 0 0",
      "PV_SECURITY_WITH_STORAGE": "yes"
```

```
    },
    "config": {},
    "docker_digest": "registry.gitlab.com/pantacor/pv-platforms/pvr-sdk@sha256:95b5d63a216773af9c8b2b82e25e2e9d40a049bae9c62cd2308eff5feb9b32dd",
    "docker_name": "registry.gitlab.com/pantacor/pv-platforms/pvr-sdk",
    "docker_source": "remote,local",
    "docker_tag": "arm32v6",
    "persistence": {},
    "template": "builtin-lxc-docker"
  },
  "storage-mapping.json": {},
}
```


begin

The begin endpoint start a new transaction only if there is not a trasaction started. If a transaction is already started this is going to fail.

```
curl -X POST __ORIGIN__/cgi-bin/pvtx/begin
```

If there is not error will return 200 with the full state response (the same seem on show)

Response:

```
{
  # ...FULL TRANSACTION
}
```

If there is an error will get this response

```
{
  "error": "No active transaction.",
  "transaction": { ...FULL TRANSACTION }
}
```

abort

This will abort the current transation running. If there is not transaction running is going to fail.

```
curl -X POST __ORIGIN__/cgi-bin/pvtx/abort
```

If everything is ok you will get a 200 OK response. If not an error similar to this.

```
{
  "error": "No active transaction. Start a transaction first.",
  "transaction": null
}
```

add

This add and update parts using the tar file result of a `pvr export` command.

```
curl -X PUT __ORIGIN__/cgi-bin/pvtx/add --data-binary @exported.tgz
```

This will fail if there is not transaction started or fails to add that exported file.

remove?part=PART_NAME

Remove any part of the current transaction. Using the example here on the show command we can see it has several parts, we could delete one by name, example: `pvr-sdk`.

```
curl -X PUT __ORIGIN__/cgi-bin/pvtx/remove?part=pvr-sdk
```

That will return the now state without the `pvr-sdk` part.

Could give you an error in case it fails.

```
{ "error": "ERROR_MESSAGE_STRING" }
```

commit

The commit endpoint allows to commit all the changes in the transaction and returns a revision ID that could be now run.

```
curl -X POST __ORIGIN__/cgi-bin/pvtx/commit
```

Response:

```
{
  "revision": "locals/pvtx-1636485481-42c76d97"
}
```

If there is not transaction to commit you will get an error.

```
{
  "error": "No active transaction. Start a transaction first.",
  "transaction": null
}
```

run?rev=REVISION_ID

Runs allow you to run any revision on the device.

```
curl -X POST __ORIGIN__/cgi-bin/pvtx/run?rev=locals/pvtx-1636485481-42c76d97
```

This will return the progress of that revision when success

```
{
  "status": "TESTING",
  "status-msg": "Awaiting to see if update is stable",
  "progress": 95
}
```

status?rev=REVISION_ID

Return the progress of a revision. The rev parameter will default to current.

```
curl __ORIGIN__/cgi-bin/pvtx/status
```

Response

```
{
  "rev": "REVISION_ID",
  "progress": {
    "status": "UPDATED",
    "status-msg": "Update finished, revision not set as rollback point",
    "progress": 100
  }
}
```

or

```
curl __ORIGIN__/cgi-bin/pvtx/status?rev=locals/pvtx-1636485481-42c76d97
```

Response:

```
{
  "rev": "locals/pvtx-1636485481-42c76d97",
  "progress": {
    "status": "UPDATED",
    "status-msg": "Update finished, revision not set as rollback point",
    "progress": 100
  }
}
```

get-config

Get all device configuration, device-meta and user-meta

```
curl __ORIGIN__/cgi-bin/pvtx/get-config
```

Response:

```
{
  "device": {
```

```
    "time": {
      "timeval": {
        "tv_sec": 33,
        "tv_usec": 72601
      },
      "timezone": {
        "tz_minuteswest": 0,
        "tz_dsttime": 0
      }
    },
    "sysinfo": {
      "uptime": 34,
      "loads.0": 34688,
      "loads.1": 8448,
      "loads.2": 2816,
      "totalram": 3990536192,
      "freeram": 3530027008,
      "sharedram": 0,
      "bufferram": 17485824,
      "totalswap": 0,
      "freeswap": 0,
      "procs": 86,
      "totalhigh": 0,
      "freehigh": 0,
      "mem_unit": 1
    },
    "storage": {
      "total": 31354646528,
      "free": 26463305728,
      "reserved": 1567732326,
      "real_free": 24895573402
    },
    "pantavisor.version": "018-11-g2ff4a37-221130-1076ae1",
    "pantavisor.uname": {
      "kernel.name": "Linux",
      "kernel.release": "4.19.127-v8+",
      "kernel.version": "#1 SMP PREEMPT Wed Nov 30 08:35:45 UTC 2022",
      "node.name": "(none)",
      "machine": "aarch64"
    },
    "pantavisor.revision": "155",
    "pantavisor.mode": "remote",
    "pantavisor.dtmodel": "Raspberry Pi 4 Model B Rev 1.2",
    "pantavisor.arch": "aarch64/64/EL",
    "pantahub.state": "idle",
    "pantahub.online": "1",
    "pantahub.claimed": "1",
    "pantahub.address": "51.158.191.153:443",
    "interfaces": {
      "lo.ipv4": [
        "127.0.0.1"
      ],
      "eth0.ipv4": [
        "192.168.68.109"
      ],
      "wlan0.ipv4": [
        "10.50.0.1"
      ],
      "lxcbr0.ipv4": [
        "10.0.3.1"
      ],
      "tailscale0.ipv4": [
        "100.109.175.19"
      ],
      "lo.ipv6": [
        "::1"
      ],
      "eth0.ipv6": [
        "fe80::1b19:27ae:2cb7:5ecc%eth0"
      ],
      "wlan0.ipv6": [
        "fe80::d3cc:6b18:8bce:b287%wlan0"
      ],
      "lxcbr0.ipv6": [
        "fe80::216:3eff:fe00:0%lxcbr0"
      ],
      "tailscale0.ipv6": [
        "fd7a:115c:a1e0:ab12:4843:cd96:626d:af13",
        "fe80::3706:f735:423:de8a%tailscale0"
      ],
      "vethN884XO.ipv6": [
        "fe80::fcbd:4ff:fe87:45b1%vethN884XO"
      ]
    }
  },
  "user": {
    "wireguard-vpn.enabled": "false",
    "tailscale.enabled": "true",
    "pvr-sdk.authorized_keys": "...........",
    "openvpn.enabled": "false",
    "fleet.channel": "arm_rpi64_bsp_latest",
    "cloudflared.upstream": "false"
  }
}
```

usermeta/save?key=KEY

Set an user meta value, this will add or update the key with the new value.

```
curl \
  -X POST \
  '__ORIGIN__/cgi-bin/pvtx/usermeta/save?key=newkey' \
  -H 'Content-Type: text/plain' \
  --data-raw "value of the usermeta"
```

Response:

```
{
  "newkey": "value of the usermeta",
  "wireguard-vpn.enabled": "false",
  "tailscale.enabled": "true",
  "openvpn.enabled": "false",
  "fleet.channel": "arm_rpi64_bsp_latest",
  "cloudflared.upstream": "false"
}
```

usermeta/delete?key=KEY

Remove a key from the user-meta

```
curl \
  -X PUT \
  '__ORIGIN__/cgi-bin/pvtx/usermeta/delete?key=newkey' \
  -H 'Content-Type: application/json'
```

Response:

```
{
  "wireguard-vpn.enabled": "false",
  "tailscale.enabled": "true",
  "openvpn.enabled": "false",
  "fleet.channel": "arm_rpi64_bsp_latest",
  "cloudflared.upstream": "false"
}
```

PVCONTROL

pvcontrol -- manage your pantavisor system with a CLI

SYNOPSIS

`pvcontrol [options] [commands] [arguments]`

DESCRIPTION

pvcontrol is a CLI tool to interact with your Pantavisor system that will allow to monitor and manage your device.

OPTIONS

`-h` Show help

`-v` Verbose

`-s` Send queries to socket (default: /pantavisor/pv-ctrl)

`-f` Send output to file instead of stdout

`-m` Commit message (only used for steps install and steps put)

COMMANDS

`ls` List all containers existing in the current revision

`groups` Container groups related operations

`signal` Send signals to Pantavisor

`commands` Send commands to the Pantavisor state machine

`usrmeta` User metadata related operations

`devmeta` Device metadata related operations

`buildinfo` Dump Pantavisor build info

`objects` Object related operations

`steps` Step related operations

`conf` Configuration related operations

To get more information about each command, just type `pvcontrol [command] -h`. For example, `pvcontrol ls -h`.

RETURN VALUE

`0` Success

`1` Syntax error

`48` Not enough disk space available

`60` Object has bad checksum

`70` Step verification has failed

`255` Unknown response

Return value layout is organized by command in the following categories for future return values: pvcontrol script errors (1-15), ls (16-31), commands (32-47), objects (48-63), steps (64-79), usrmeta (80-95), devmeta (96-111), buildinfo (112-127), conditions (128-143), conf (144-159) and unknown (255).

COPYRIGHT

pvcontrol is Copyright (c) 2022 by Pantacor Ltd.

7.3 Pantavisor CI

7.3.1 Pantavisor CI

Pantavisor CI is a set of tools for continuous integration and continuous development on Pantavisor projects. It does so by offering several GitLab CI templates to be included, extended and configured from your own project.

Update CI

Update CI GitLab template allows to keep a number of devices up to date and build flashable images from them in a way that those devices revisions and images can be traceable and reproducible.

The two main functionalities of this template are:

- **Keeping devices up to date**: a list of Pantacor Hub devices can be referred so their BSP and containers are automatically updated using GitLab pipelines. A stable device counterpart can be configured so certain revisions can be promoted from the updatable devices. These updatable and stable devices can then be used as device channels that will feed others in your fleet.
- **Building flashable images**: images can be built and uploaded to the cloud when a device revision is promoted to stable. Different image channels can be set for each device (stable, release candidate, development, etc.) to help you set up-to-date start points for your prototype and production devices.

HOW TO USE UPDATE CI

To start using it, just add our template project to a new GitLab project as a submodule:

```
git submodule add https://gitlab.com/pantacor/ci/device-ci.git
./device-ci/mkdevice-ci.sh
```

This will help you through the process of setting up the contents of .gitlab-ci.yml to be able to run the pipelines with our update-ci template.

To get more information about update-ci, go to our reference.

To see an example of use, go to our pv-initial-devices project.

BSP CI

BSP CI GitLab template helps setting up a project to build Pantavisor BSP binaries for multiple architectures, from source code, in a reproducible and traceable manner.

To get more information about bsp-ci, go to our reference.

To see an example of use, go to our pv-manifest project.

PV Deps CI

PV Devs CI GitLab template is meant to be used from any of the BSP Pantavisor dependency projects and build the BSP for generic architectures when changes are pushed to those projects.

To get more information about pvdeps-ci, go to our reference.

To see an example of use, go to our libthttp project.

7.3.2 Update CI Reference

This template can be used from a new empty GitLab project.

mkdevice-ci.sh

This script will help you set up your update-ci project:

```
mkdocker.sh <operation> [options]

  init       initialize project: create device-ci.conf.json
  install    install projcet: generate .gitlab-ci.yml using the contents of device-ci.conf.json
```

device-ci.conf.json

Device CI configuration file has this format:

```
{
  "yml_template":"update-ci.yml",
  "device_list": [..]
}
```

This table defines all keys for device-ci.conf.json:

Key	Value	Default	Description
yml_template	string	**mandatory**	yml template we are going to use
device_list	list of devices	empty	list of updatable devices

DEVICE

This JSON follows this format:

```
{
  "id":"rpi3_initial",
  "schedule":"rpi3_initial_latest",
  "tag":"rpi3_initial_stable",
  "image_channel:""
  "build": [..]
}
```

The table below explains the necessary keys for each device:

Key	Value	Default	Description
id	string	**mandatory**	unique name to identify each device
schedule	string	**mandatory**	Pantacor Hub device name that will be kept automatically updated when GitLab CI is either manually triggered or scheduled
tag	string	**mandatory**	Pantacor Hub device name that is automatically updated with the contents /recipes when a commit is promoted
image_channel	string	empty	name of the image channel
build	list of builds	empty	list of images per device

BUILD

Build JSON looks like this:

```
{
  "name":"default",
  "options":"PANTAVISOR_DEBUG=yes"
}
```

This table shows the keys for each build:

Key	Value	Default	Description
name	string	**mandatory**	unique name for the image name of this device
options	string	empty	build options

Environment variables

GitLab CI variables used by the template:

Key	Value	Default	Description
PHUSER	string	**mandatory**	Pantacor Hub user that owns the to-be-updated devices
PHPASS	string	**mandatory**	Pantacor Hub password
PH_CIBOT_EMAIL	string	**mandatory**	Email used for git commits
PH_CIBOT_GITLAB_TOKEN	string	**mandatory**	GitLab access token for CI bot
PH_CIBOT_GITLAB_USER	string	**mandatory**	Gitlab user for CI bot. This user needs to have write permissions to the project
AWS_ACCESS_KEY_ID	string	empty	Amazon AWS access ID
AWS_SECRET_ACCESS_KEY	string	empty	Amazon AWS secret key
AWS_BUCKET	string	empty	Amazon AWS bucket name
AWS_PROJECT_PATH	string	empty	Amazon AWS project path
DEPLOY_TRIGGER_PROJECT	string	empty	Project that is going to be triggered after a stable build is passed
DEPLOY_TRIGGER_TOKEN	string	empty	GitLab token with triggering permissions over the deploy project

IMPORTANT: Don't forget to set your variables to Masked so they can not be seen in the GitLab log!

Triggers

- Update device: triggering GitLab pipeline either manually or with a pipeline schedule will result in the device BSP and containers being updated, according to whatever information is in their respective src.json.
- Promote commit: creating a tag over a certain commit in the device project will result in posting each revision in recipes/ to its stable counterpart. It will also build and upload a flashable image to AWS if configured.

Device channel

According to the triggers in the previous section, there are two Pantacor Hub device channels where the revisions will be automatically posted:

- Schedule device: devices that are kept up to date depending on their respective src.json. On success, device revision metadata is self commited to the device-ci project in recipes/.
- Tag device: devices that are updated with the contents of the recipes/ when a commit is promoted. Will only be updated if *tag device* is different than the *schedule device*.

Image channel

Images are only built in a promotion event and uploaded to AWS if configured. Resulting images are registered in `https://pantavisor-ci.s3.amazonaws.com/<project-name>/stable.json` depending on the channel they belong:

- Stable: for tags without `-` character and devices with empty image channel.
- Release candidate: for tags with "-rc*" suffix and devices with empty image channel.
- Additional channels: for tags with "-\<image_channel-name>" suffix and devices with a configured image channel.

7.3.3 BSP CI Reference

This template needs to be imported from a GitLab project with a release.xml and default.xml manifests generated by repo.

.gitlab-ci.yml

To start using the pvdeps-ci template, just import it from your .gitlab-ci.yml:

```
include:
  project: 'pantacor/ci/device-ci'
  ref: '011'
  file: '/yml/bsp-ci.yml'
```

To configure the targets for the build jobs, you just need to configure these parameters for each target:

```
<job-name>:
  extends: .build-bsp
  variables:
    PLATFORM: <platform>
    TARGET: <target>
    PH_TARGET_DEVICE_LATEST: <pantahub-latest-device>
    PH_TARGET_DEVICE_STABLE: <pantahub-stable-device>
```

- job-name: Choose any unique name for your job.
- platform: Build platform.
- target: Build target.
- pantahub-latest-device: Pantacor Hub device that will be updated from the GitLab scheduler or when a new commit is pushed.
- pantahub-stable-device: Pantacor Hub device that will be updated when a new tag is pushed.

For example:

```
build-rpi3:
  extends: .build-bsp
  variables:
    PLATFORM: rpi3
    TARGET: arm-rpi3
    PH_TARGET_DEVICE_LATEST: arm_rpi3_bsp_latest
    PH_TARGET_DEVICE_STABLE: arm_rpi3_bsp_stable
```

Environment variables

GitLab CI variables used by the template:

Key	Value	Default	Description
PHUSER	string	**mandatory**	Pantacor Hub user that owns the to-be-updated devices
PHPASS	string	**mandatory**	Pantacor Hub password

IMPORTANT: Don't forget to set your variables to Masked so they can not be seen in the GitLab log!

Triggers

- latest: triggering GitLab pipeline either manually or with a pipeline schedule will result in a BSP build that will be posted to the configured Pantacor Hub latest device for each one of the jobs.
- stable: creating a tag over a certain commit will result in a BSP build that will be poseted to the configured Pantacor Hub stable device for each job.

7.3.4 PV Deps CI Reference

This template can be used from any of the Pantavisor BSP dependency project in Gitlab.

.gitlab-ci.yml

To start using the pvdeps-ci template, just import it from your .gitlab-ci.yml:

```
include:
  project: 'pantacor/ci/device-ci'
  ref: '011'
  file: '/yml/pvdeps-ci.yml'
```

Environment variables

GitLab CI variables used by the template:

Key	Value	Default	Description
PHUSER	string	**mandatory**	Pantacor Hub user that owns the to-be-updated devices
PHPASS	string	**mandatory**	Pantacor Hub password
PH_TARGET_DEVICE	string	empty	Pantacor Hub device name. Several can be set separated by space character
ARTIFACT	string	empty	Name of the -generic target you want to use in your device. Several can be set separated by space and it must be one for each target device configured

IMPORTANT: Don't forget to set your variables to Masked so they can not be seen in the GitLab log!

7.4 App Engine

7.4.1 Pantavisor App Engine

Welcome to Pantavisor App Engine!

To install it, you just simply have to run the `install` script, setting the output path of your choice:

```
sudo /tmp/appengine/install /opt/pantavisor
```

If everything goes well, the `install` script will inform you about how to run it, like this:

```
sudo /opt/pantavisor/pantavisor -i
```

Have more a more detailed how-to guide along further information in Pantavisor documentation.

7.4.2 Pantavisor Test Framework

The Pantavisor Test Framework contains a set of scripts to permit automated testing on Pantavisor taking advantage of the App Engine init mode.

Requirements

Firstly, to be able to run ARM containers on a x64 host in an apt based system:

```
sudo apt install binfmt-support docker.io git jq qemu qemu-user-static
sudo update-binfmts --enable qemu-arm
```

Now, you need to get the Pantavisor BSP source code.

```
repo init -u https://gitlab.com/pantacor/pv-manifest -m release.xml
repo sync -j10 -c --no-clone-bundle
```

If you prefer to sync by group with the -g option, remember to add the x64-appengine target:

```
repo init -u https://gitlab.com/pantacor/pv-manifest -m release.xml -g runtime,x64-appengine
repo sync -j10 -c --no-clone-bundle
```

How to Run All Tests

Now, you have to build Pantavisor for the x64-appengine target:

```
PANTAVISOR_DEBUG=yes ./build.docker.sh x64-appengine
```

The resulting tarball will be used by the test script. To run all tests with it:

```
./test.docker.sh run
```

This might take a while, but you can always stop it with CTRL-C.

How to Run a Test

Tests can be individually run too.

First, it is important to know which tests are available:

```
./test.docker.sh ls
```

For example, to run the test number 0 from the 'local' group of tests:

```
./test.docker.sh run local:0
```

To run all tests from the local group of tests:

```
./test.docker.sh run local
```

How to Create a New Test

To create a new test inside of the 'local' test group:

```
./test.docker.sh add local
```

This will create the necessary templates for a new test in `tests/local/data/roles`. These templates can then be edited manually to get the desired test.

How to Modify the behavior of an Existing Test

If we go to any newly created test using the previous command, we will find a directory and a JSON file:

```
output  resources  test.json
```

The test.json file contains the necessary metadata information for the execution and evaluation of a test:

```
{
        "#spec": "pv-test@1",
        "description": "ls query to mgmt and non-mgmt containers",
        "setup": {
                "cmdline": "",
                "pantavisor.config": "../../common/configs/pantavisor.config",
                "pvs": "../../common/pvs/pvs.defaultkeys.tar.gz",
                "containers": {
                        "control": "pvr-sdk",
                        "tarballs": [
                        ],
                        "urls": [
                                "https://pvr.pantahub.com/pantahub-ci/pv_tests_assets/3#pvr-sdk/",
                                "https://pvr.pantahub.com/pantahub-ci/pv_tests_assets/3#pvr-sdk_norole/"
                        ]
                },
                "ready-script": "resources/ready",
        },
        "test-script": "resources/test",
        "skip": "false"
}
```

TEST.JSON

`#spec`

Version of the test.json. Only supported value is 'pv-test@1'.

`description`

Brief description of the test objective.

`setup`

Initial Pantavisor setup for the test.

`cmdline`

Substitutes what is read from /proc/cmdline. Can be used for configuraiton purposes.

`pantavisor.config`

Pantavisor configuration file that will be used for the test.

`pvs`

Tarball with the secureboot certs and keys. This makes possible to use tests from different manifests in our own.

`containers`

Initial container setup for the test.

`control`

The name of the container that will be used by the Test Framework utilities as the management controller for things such as initialization and shutdown.

It is important to set a control container for the correct operation of the test (unless what is being tested is a device without management containers), and it is important to notice that the container must be provided either in the list of tarballs or the list of urls.

`tarballs`

List of tarballs to be installed in the first revision for the test. Can be left empty.

`urls`

List of Clone URLs to be installed in the first revision for the test. Can be left empty.

`ready-script`

POSIX-compliant sh script to be executed before evaluating if Pantavisor status is READY. Can be used to force a READY status for any container.

`test-script`

The test itself. Must be a POSIX-compliant sh script. The stdout and stderr of it will form the result of the test.

`skip`

Whether to skip a test or not. Possible values are 'true' or 'false'.

The test will still be executable individually or interactively.

How to Debug a Test

To interactively run a test:

```
./test.docker.sh run local:0 -i
```

A console will be opened with an instance of App Engine already running with the parameters that were set in test.json, except the test script will not be automatically run.

If Pantavisor is not getting to the READY status, you can use the manual option plus the interactive one:

```
./test.docker.sh run local:0 -i -m
```

With this, the console will be opened right before starting Pantavisor. You will be able to start it with a modified configuration, such as enabling stdoud logs for debugging.

How to Modify the Result of a Test

To create or overwrite the results of a test:

```
./test.docker.sh run local:0 -o
```

7.5 pvr

7.5.1 PVR

PVR operates on two types of entities:

Sumodules:

- PVR Repository
- PVR Pantahub
- PVR Devel

Features

- simple repository in json format with object store
- checkout and commit any working directory to repository
- look at working tree diffs with `status` and `diff` commands
- get, push and apply patches as incremental updates
- all repo operations are atomic and can be recovered
- object store can be local or in cloud/CDN

Requirement

- a server backend implementing state CRUD primitives like offered by pantahub "trails"

Install

1. From gitlab:

```
$ go get gitlab.com/pantacor/pvr
$ go build -o ~/bin/pvr gitlab.com/pantacor/pvr
```

Get Started

pvr is about transforming a directory structure that contains files as well as json files into a single managable, diffable and mergeable json file that unambiguously defines directory state.

To leverage the features of json diff/merge etc. all json files found in a directory tree will get inlined while all other objects will get a sha reference entry with the files themselves getting stored in a flat objects directory.

Basics

To start a pvr from scratch you use the pvr init command which sets you up.

```
$ pvr init
```

However, more likely is that you want to consume an existing pvr made by you or someone else and maybe change things against it:

```
$ pvr clone /path/to/pvr/repo example2
 -> mkdir example2
 -> pvr init
 -> pvr get /path/to/pvr/repo
 -> pvr checkout
```

While working on changes to your local checkout, you can use `status` and `diff` to observe your current changes:

```
$ pvr status
A newfile.txt
D deleted.txt
C some.json
C working.txt
```

This means that `newfile.txt` is new, working.txt and some.json changed and `deleted.txt` got removed from your working directory.

You can introspect the changes through the `diff` command:

```
$ pvr diff
{
    "deleted.txt": null,
    "newfile.txt": "dc460da4ad72c482231e28e688e01f2778a88ce31a08826899d54ef7183998b5",
    "some.json": {
        "values": "2"
    },
    "working.txt": "9c7ab50fa91f3d78744043af5f88dce6bacd336f47733ff6a38090da3ff1a5de"
}
```

Being happy with what you see, you can checkpoint your working state using the `commit` command:

```
$ pvr commit
Committing some.json
Committing working.txt
Adding newfile.txt
Removing deleted.txt
```

This will atomically update the json file in your repo after ensuring all the right objects have been synched into the objects store of that pvr repo.

After committing your changes you might want to store your current repository state for reuse or archiving purpose. You can do so using the `push` command:

```
$ pvr push /tmp/myrepo
```

You can always get a birds view on things in your repo by dumping the complete current json:

```
$ pvr json
{...}
```

You can also push your repository to a pvr compliant REST backend. In this case to a device trails (replace device id with your device)

```
$ pvr post https://api.pantahub.com/trails/<DEVICEID>
```

You can later clone that very repo to use it as a starting point or get its content to update another repo.

Internals

The pvr repository has the following structure in v1:

THE STATE JSON

```
$ cat json
{
  "spec": "pantavisor-multi-platform@1",
  "brcm.tar.gz": "8862f6feea4f6d01e28adc674285640874da19d7594dd80ed42ff7fb4dc0eea3",
  "pp/test.txt": "ad6da30bb62fae51c770574a5ca33c5e8e4bbc67fd6c5e7c094c34ad52a28e4d",
  "pp/test1.txt": "ad6da30bb62fae51c770574a5ca33c5e8e4bbc67fd6c5e7c094c34ad52a28e4d",
  "test.json": {
    "I": [
      "thank",
      "you"
    ],
    "My": "Mother",
    "more": "than"
  }
}
```

THE OBJECTS REPOSITORY

Every PVR Repo is backed by an object repository which has for each file a hashed file in it. These will be used on device as hardlink or on a checkout as a source to copy the files referenced from the state json file above.

By default the objects are kept centrally so you they get reused across potentially many projects you might checkout as a developer, but on device or in special cases you can use the --objects-dir parameter to use a different location.

```
$ ls objects/
8862f6feea4f6d01e28adc674285640874da19d7594dd80ed42ff7fb4dc0eea3
ad6da30bb62fae51c770574a5ca33c5e8e4bbc67fd6c5e7c094c34ad52a28e4d
d0365cf6153143a414cccaca9260bc614593feba9fe0379d0ffb7a1178470499
d9206603679fcf0a10bf4e88bf880222b05b828749ea1e2874559016ff0f5230
```

Commands

PVR INIT

```
$ pvr init
```

Observe how the repo json got created in a subdirectory:

```
$ cat .pvr/json
{
    "#spec": "pantavisor-multi-platform@1"
}
```

You would now continue editing this directory as it pleases you and you can refer to any file you put here in your configs just using the path as key (e.g. `systemc.json": {}` would create a systemc.json file on checkout).

PVR ADD [FILE1 FILE2 ...]

`prv add` will put a file that exists in working directory under management of pvr. This means that the file will be honored on future `pvr diff` and `pvr commit` operations.

Example: If you bring in a basic platform to your system you simply copy them into the working dir and put them under pvr management:

```
$ cp /from/somewhere.conf lxc-platform.conf
$ cp /from/somewhere.json lxc-platform.json
$ pvr add lxc-platform.json pxc-platform.conf
```

These files will then be part of the next commit.

pvr add --raw [file1 ...]

By default pvr inlines any .json file in the main system state and does not save it as an object. This implies that the exact formatting and order of elements in that .json might get lost and for large .json files this might artifically grow the state json.

For those cases one might decide to hint at pvr that a .json file shall be added as a raw object instead. For that oe uses the pvr add --raw command flag.

You can use that for new files but also to mark an already committed file for conversion to a raw object on next commit.

PVR DIFF

You can look at your current changes to working directory using the diff command to get RFCXXXX json patch format:

```
$ pvr diff
{
    "lxc-platform.conf": "sha1:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "lxc-platform.json": { ... }
}
```

PVR COMMIT

Committing your pvr will update the .pvr directory so it can be pushed to pantahub.

```
$ pvr commit
Adding file xxx
```

```
Changing File yyy
Commit Done
```

You can then continue editing and see your changes compared to the committed baseline using `pvr diff` again.

PVR PUT

Put local:

```
$ pvr put /some/local/repopath

$ find /some/local/repopath
/some/local/repopath/json
/some/local/repopath/objects/xxxxxxxxxxxxxxxxxxxxx
/some/local/repopath/objects/yyyyyyyyyyyyyyyyyyyyy
```

PVR POST

Post local pvr as a new revision to your device endpoint:

```
$ pvr post https://api.pantahub.com/trails/<YOURDEVICE>
...
```

PVR CLONE

you can clone a remote device state as follows:

```
$ pvr clone https://api.pantahub.com/trails/<YOURDEVICE>
...
```

Alternatively you can get a specific revision:

```
$ pvr clone https://api.pantahub.com/trails/<YOURDEVICE>/steps/<REV>
...
```

PVR CLONE

When developing a device that has a pvr compatible cgi local frontend (such as provided by pvr-sdk by default) installed you can use `pvr clone IP` directly.

In that `pvr clone 1.2.3.4` is equivalent to `pvr clone http://1.2.3.4:12368/cgi-bin/pvr`.

PVR FASTCOPY [#PART1,PART2,PATH/PART3] [#PART1NAME,$PART2NAME,...]

You can copy device state or subelements from one device to another without downloading the bits to your local machine.

This only works for source devices that the user has access to. Right now thats the case for Public devices as well as for devices owned by the user itself.

To fastcopy a complete device experience you would simply use:

```
pvr fastcopy -m "your commit message to remember" \
    https://pvr.pantahub.com/pantahub-ci/rpi4_initial_latest \
    https://pvr.pantahub.com/yournick/yourdevice
```

NOTE: this will delete app apps and bsp/ entries before replacing them. If you dont want that you have to use fragment encoded in the URL to select specific source elements to copy and replace.

To fastcopy a specific folder of your source device and replace the matching folder you coudl simply use:

```
pvr fastcopy -m "your commit message (copying #fragement)" \
    https://pvr.pantahub.com/pantahub-ci/rpi4_initial_latest#fragment \
    https://pvr.pantahub.com/yournick/yourdevice
```

Example (copy just the bsp/):

```
pvr fastcopy -m "your commit message to file for the new revision" \
    https://pvr.pantahub.com/pantahub-ci/rpi4_initial_latest#bsp \
    https://pvr.pantahub.com/yournick/yourdevice
```

Similarly you can copy apps this way:

```
pvr fastcopy -m "copy pvr-sdk from source device to yourdevice" \
    https://pvr.pantahub.com/pantahub-ci/rpi4_initial_latest#pvr-sdk \
    https://pvr.pantahub.com/yournick/yourdevice
```

You can even copy multiple parts like the matching config for your app:

```
pvr fastcopy -m "copy pvr-sdk from source device to yourdevice" \
    https://pvr.pantahub.com/pantahub-ci/rpi4_initial_latest#pvr-sdk,_config/pvr-sdk \
    https://pvr.pantahub.com/yournick/yourdevice
```

... and to make things complete rename them during the copy:

```
pvr fastcopy -m "copy pvr-sdk from source device to yourdevice" \
    https://pvr.pantahub.com/pantahub-ci/rpi4_initial_latest#pvr-sdk,_config/pvr-sdk \
    https://pvr.pantahub.com/yournick/yourdevice#pvr-sdk-new,_config/pvr-sdk-new
```

PVR DEVICE COMMANDS

PVR Devices commands provide convenience for developers and individuals that operate their very own pantavisor enabled solutions.

These commands are designed for developers that want to interface with devices in their own local network, but not does not replace a fleet management solution.

pvr device scan

Scan for pantavisor enabled devices on local network.

Info about how to interface and claim devices that dont have an owner yet are printed on console.

```
example1: $ pvr device scan
Scanning ...
        ID: 5b0aa4363c6f7200095b2566 (owned)
        Host: linux.local.
        IPv4: [192.168.178.97]
        IPv6: [2a02:2028:713:3001:602:a2ff:feb3:d4e8]
        Port: 22
        Pantahub WWW: https://hub.pantacor.com/u/_/devices/5b0aa4363c6f7200095b2566
        PVR Clone: https://api.pantahub.com:443/trails/5b0aa4363c6f7200095b2566
```

pvr claim -c https://api.pantahub.com:443/devices/

pvr claim: Claims a new device

```
example1$ pvr claim -c seemingly-rich-mastodon https://api.pantahub.com:443/devices/5d0a2b4e12734a0008363a9a

Login (/type [R] to register) @ https://api.pantahub.com/auth (realm=pantahub services) ***
Username: youremail@gmail.com
Password: *****
```

pvr device create [DEVICE_NICK]

pvr device create creates a new device from an existing device diretory usings its same state values.

```
example1$ pvr device create mydevice1
{
    "device-meta": {},
    "garbage": false,
    "id": "5d7645019061a500098617bd",
    "nick": "mydevice1",
    "owner": "prn:::accounts:/5bf2ac9e41b2dd0009a96c97",
    "prn": "prn:::devices:/5d7645019061a500098617bd",
    "public": false,
    "secret": "mznzxle72fjf6w2",
    "time-created": "2019-09-09T12:26:41.289253663Z",
    "time-modified": "2019-09-09T12:26:41.289253663Z",
    "user-meta": {}
}
Device Created Successfully
```

pvr device get

pvr device get : Get Device details

```
example1$ pvr device get 5db2f6878b693e0009f1b29f
{
```

```
    "device-meta": {},
    "garbage": false,
    "id": "5db2f6878b693e0009f1b29f",
    "nick": "heavily_strong_pika",
    "owner": "prn:::accounts:/5bf2ac9e41b2dd0009a96c97",
    "owner-nick": "sirinibin",
    "prn": "prn:::devices:/5db2f6878b693e0009f1b29f",
    "public": false,
    "secret": "mznzxle72fjf6w2",
    "time-created": "2019-10-25T13:20:07.842Z",
    "time-modified": "2019-10-25T13:20:07.842Z",
    "user-meta": {}
}
example2$ pvr device get heavily_strong_pika
{
    "device-meta": {},
    "garbage": false,
    "id": "5db2f6878b693e0009f1b29f",
    "nick": "heavily_strong_pika",
    "owner": "prn:::accounts:/5bf2ac9e41b2dd0009a96c97",
    "owner-nick": "sirinibin",
    "prn": "prn:::devices:/5db2f6878b693e0009f1b29f",
    "public": false,
    "secret": "mznzxle72fjf6w2",
    "time-created": "2019-10-25T13:20:07.842Z",
    "time-modified": "2019-10-25T13:20:07.842Z",
    "user-meta": {}
}
```

pvr device set = [KEY2]=[VALUE2]...[KEY-N]=[VALUE-N]

pvr device set : Set or Update device user-meta & device-meta fields (Note:If you are logged in as USER then you can update user-meta field but if you are logged in as DEVICE then you can update device-meta field)

```
example1$ pvr device set 5df243ff0be81900099855e6 a=1 b=2
{
    "a": "1",
    "b": "2"
}
user-meta field Updated Successfully
```

```
example2$ pvr device set 5df243ff0be81900099855e6 a=1 b=2
{
    "a": "1",
    "b": "2"
}
device-meta field Updated Successfully
```

pvr device logs [--template=<short|json|] [/source][@level][#platform]

pvr device logs list the logs with filter options of device,source,level & platform

```
example1$ pvr device logs 5d555d5e80123b31faa3cff2/pantavisor.log@INFO#windows
2020-01-06T12:54:17Z 5e0f4ede:windows(pantavisor.log):INFO       My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:43Z 5e0f4ede:windows(pantavisor.log):INFO       My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:53Z 5e0f4ede:windows(pantavisor.log):INFO       My log line 3 to remember from device:5d555d5e80123b31faa3cff2
```

pvr device logs list the logs with filter options of multiple device,source,level &platform

```
example2$ pvr device logs 5d555d5e80123b31faa3cff2,5d555d5e80123b31faa3cff5/pantavisor.log,pantavisor2.log@INFO,INFO2#windows,linux
2020-01-06T12:54:17Z 5e0f4ede:windows(pantavisor.log):INFO       My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:43Z 5e0f4ede:windows(pantavisor.log):INFO       My log line 1 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:54:53Z 5e0f4ede:windows(pantavisor.log):INFO       My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:55:03Z 5e0f4ede:windows(pantavisor.log):INFO       My log line 2 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:55:17Z 5e0f4ede:linux(pantavisor2.log):INFO2       My log line 3 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:55:43Z 5e0f4ede:linux(pantavisor2.log):INFO2       My log line 3 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:55:53Z 5e0f4ede:linux(pantavisor2.log):INFO2       My log line 4 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:56:03Z 5e0f4ede:linux(pantavisor2.log):INFO2       My log line 4 to remember from device:5d555d5e80123b31faa3cff5
```

```
pvr device logs --from=2020-01-06 --to=2020-01-07 list the logs within a given date range
```

```
example3\$ pvr device logs 5d555d5e80123b31faa3cff2,5d555d5e80123b31faa3cff5/pantavisor.log,pantavisor2.log@INFO,INFO2
2020-01-06T12:54:17Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:43Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:54:53Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:03Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff5
2020-01-07T12:55:17Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:43Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff5
```

```
2020-01-07T12:55:53Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:56:03Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff5
```

```
pvr device logs --from=2020-01-06T12:54:10--to=2020-01-06T12:54:20 list the logs within a given date time range
```

```
example4\$ pvr device logs 5d555d5e80123b31faa3cff2,5d555d5e80123b31faa3cff5/pantavisor.log,pantavisor2.log@INFO,INFO2
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:15Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff5
2020-01-07T12:55:15Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff5
2020-01-07T12:55:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:56:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff5
```

```
pvr device logs --from=2020-01-06T12:54:10+05:30 --to=2020-01-06T12:54:20+05:30 list the logs within a given date time range having timezone:
+05:30(IST) ,Note:Timezone is optional
```

```
example5\$ pvr device logs 5d555d5e80123b31faa3cff2,5d555d5e80123b31faa3cff5/pantavisor.log,pantavisor2.log@INFO,INFO2
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:15Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff5
2020-01-07T12:55:15Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff5
2020-01-07T12:55:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:56:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff5
```

```
pvr device logs --from=P10D --to=P5D list the logs from last 10 days to last 5 days
Note: --from & --too flags support the ISO 8601 Duration strings
```

```
example6\$ pvr device logs 5d555d5e80123b31faa3cff2,5d555d5e80123b31faa3cff5/pantavisor.log,pantavisor2.log@INFO,INFO2
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:54:10Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:15Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff5
2020-01-07T12:55:15Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff2
2020-01-07T12:55:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff5
2020-01-07T12:55:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:56:20Z 5e0f4ede:pantavisor2.log:INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff5
```

pvr device logs list the logs with filter options of multiple platforms

```
example7\$ pvr device logs --platform=linux,windows
5ea97febfb( ) May 1 20:28:28 linux(pantavisor.log ): My log line to remember
5ea97febfb( ) May 1 20:28:29 windows(pantavisor.log ): My log line to remember
5ea97febfb( ) May 1 20:28:30 linux(pantavisor.log ): My log line to remember
```

PVR EXPORT

pvr export : Exports repo into single file (tarball)

```
example1$ pvr export device.tar.gz
$ ls
alpine-hotspot bsp download-layer nginx-app pvr-sdk
app2 device.tar.gz network-mapping.json pv-avahi storage-mapping.json
```

PVR IMPORT

pvr import : import repo tarball (like the one produced by 'pvr export') into pvr in current working dir.It can import files with.gz or .tgz extension as well as plain .tar. Will not do pvr checkout, so working directory stays untouched.

```
example1\$ pvr import device.tar.gz
```

PVR PANTAHUB COMMANDS

Since version 006 PVR also provides convenience commands for interacting with pantahub regardless beyond publishing pvr repositories to pantahub trails.

pvr ps

`WARNING:` This command is DEPRECATED, please use `pvr device ps` instead

`pvr ps` gets a list of devices like below:

```
\$ pvr ps
ID NICK REV STATUS STATE SEEN MESSAGE
5a21cefc tops_urchin 20 NEW xxxx 7 months ago message....
5af32b42 verified_cicada 5 NEW xxxx 5 months ago message....
5af4ca2c classic_crappie 0 DONE xxxx 5 months ago message....
5b07f476 resolved_mule 0 DONE xxxx 4 months ago message....
5b07ff81 right_vervet 0 DONE xxxx 4 months ago message....
5b08464f helped_aphid 3 NEW xxxx about 23 hours ago message....
```

pvr logs [/source][@level][#platform]

`WARNING:` This command is DEPRECATED, please use `pvr device logs` instead

pvr logs list the logs with filter options of device,source & level

```
example1\$ pvr logs 5d555d5e80123b31faa3cff2/pantavisor.log@INFO
2020-01-06T12:54:17Z 5e0f4ede:pantavisor.log:INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:43Z 5e0f4ede:pantavisor.log:INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:53Z 5e0f4ede:pantavisor.log:INFO My log line 3 to remember from device:5d555d5e80123b31faa3cff2
```

pvr logs list the logs with filter options of multiple device,source & level

```
example2\$ pvr logs 5d555d5e80123b31faa3cff2,5d555d5e80123b31faa3cff5/pantavisor.log,pantavisor2.log@INFO,INFO2#linux,windows
2020-01-06T12:54:17Z 5e0f4ede:windows(pantavisor.log):INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:54:43Z 5e0f4ede:windows(pantavisor.log):INFO My log line 1 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:54:53Z 5e0f4ede:windows(antavisor.log):INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:55:03Z 5e0f4ede:linux(pantavisor.log):INFO My log line 2 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:55:17Z 5e0f4ede:linux(pantavisor2.log):INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:55:43Z 5e0f4ede:linux(pantavisor2.log):INFO2 My log line 3 to remember from device:5d555d5e80123b31faa3cff5
2020-01-06T12:55:53Z 5e0f4ede:linux(pantavisor2.log):INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff2
2020-01-06T12:56:03Z 5e0f4ede:linux(pantavisor2.log):INFO2 My log line 4 to remember from device:5d555d5e80123b31faa3cff5
```

pvr logs list the logs with filter options of multiple platforms

```
example3\$ pvr logs --platform=linux,windows
5ea97febfb( ) May 1 20:28:28 linux(pantavisor.log ): My log line to remember
5ea97febfb( ) May 1 20:28:29 windows(pantavisor.log ): My log line to remember
5ea97febfb( ) May 1 20:28:30 linux(pantavisor.log ): My log line to remember
```

pvr logs --template=

Log output can be formatted using `--template` flag.

Valid values are: * `json` : json format (all fields) * `short` : short format (default) * `TEMPLATE` : golang template (see below)

Golang template syntax is documented in official golang docs: https://golang.org/pkg/text/template/

The object passed to the parse is the entire LogsEntry struct. Hence you can refer to any field inside your template:

```
type Entry struct {
    ID          primitive.ObjectID `json:"id,omitempty" bson:"_id,omitempty"`
    Device      string             `json:"dev,omitempty" bson:"dev"`
    Owner       string             `json:"own,omitempty" bson:"own"`
    TimeCreated time.Time          `json:"time-created,omitempty" bson:"time-created"`
    LogTSec     int64              `json:"tsec,omitempty" bson:"tsec"`
    LogTNano    int64              `json:"tnano,omitempty" bson:"tnano"`
    LogRev      string             `json:"rev,omitempty" bson:"rev"`
    LogPlat     string             `json:"plat,omitempty" bson:"plat"`
    LogSource   string             `json:"src,omitempty" bson:"src"`
    LogLevel    string             `json:"lvl,omitempty" bson:"lvl"`
    LogText     string             `json:"msg,omitempty" bson:"msg"`
}
```

Also we offer the following template functions: * sprig funcs: https://github.com/Masterminds/sprig * `sprint FORMAT VALUE` : single value string formatter using strformat syntax (e.g. {{ "test1234" | sprintf "%4s" }} ⇒ test) * `basename PATH` : basename of a given path (e.g. `{{ "/`

`path/one" | basename }} => one`) * `timeformat TIME` : time format for a time field like TimeCreated (e.g. `{{ .TimeCreated | timeformat "Stamp" }} => one`) * Any golang time constant is supported, see https://golang.org/pkg/time/#pkg-constants (e.g. ANSIC, UnixDate, …)

pvr login [END_POINT]

Login to pantahub with your username & password with an optional end point Note:Default endpoint is https://api.pantahub.com/auth/auth_status

```
example1\$ pvr login
**_ Login (/type [R] to register) @ https://api.pantahub.com/auth (realm=pantahub services) _**
Username: sirinibin2006@gmail.com
Password: **\***
Response of GET https://api.pantahub.com/auth/auth_status
{
"exp": 1568206179,
"id": "sirinibin2006@gmail.com",
"nick": "sirinibin",
"orig_iat": 1568205279,
"prn": "prn:::accounts:/5bf2ac9e41b2dd0009a96c97",
"roles": "user",
"scopes": "prn:pantahub.com:apis:/base/all",
"type": "USER"
}
LoggedIn Successfully!
```

```
example2\$ pvr login https://api2.pantahub.com/auth
**_ Login (/type [R] to register) @ https://api2.pantahub.com/auth (realm=pantahub services) _**
Username: sirinibin2006@gmail.com
Password: **\***
Response of GET https://api2.pantahub.com/auth/auth_status
{
"exp": 1568206179,
"id": "sirinibin2006@gmail.com",
"nick": "sirinibin",
"orig_iat": 1568205279,
"prn": "prn:::accounts:/5bf2ac9e41b2dd0009a96c97",
"roles": "user",
"scopes": "prn:pantahub.com:apis:/base/all",
"type": "USER"
}
LoggedIn Successfully!
```

pvr whoami

pvr whoami : List the loggedin details with pantahub instances

```
example1\$ pvr whoami
sirinibin(prn:::accounts:/5bf2ac9e41b2dd0009a96c97) at https://api.pantahub.com/auth
```

PVR GET [REMOTE-OR-LOCAL-REPOSITORY[#PART1,PART2,-UNPART]] [TARGET-REPOSITORY]

Update target-repository from remote or local repository

```
# get latest from remembered repository
example1$ $ pvr get
33aefd8dbf46f05 [OK]
686a026cd606613 [OK]
ff2a85a62cd09f1 [OK]
76d1d085d44fd3f [OK]
b30e6b64d3e1ecb [OK]
66ae7cac57d6d05 [OK]
e8305714eaadc57 [OK]
989647b3d5b7fde [OK]
612408620229de6 [OK]
f1b441cb2721355 [OK]
d49d56059ba219c [OK]
3f4889e5eed2252 [OK]
b56390fbeb5e46f [OK]
70075ca4496451e [OK]
```

or

```
# get latest from remembered repository
example1$ $ pvr get pantahub-ci/rpi3_initial_latest
fe4d959c5541950 [OK]
5a544d45a44cf3e [OK]
834d31840a923e0 [OK]
```

```
763030a2c49b8e8 [OK]
5090f4922c2492e [OK]
018e41b74500fa7 [OK]
237277bb3586033 [OK]
7bdd13f15773ab1 [OK]
62f92986988541a [OK]
13cef79b5098b82 [OK]
dee31a19ce47772 [OK]
67b1ce399971e30 [OK]
12c8d468094e3d0 [OK]
```

You can also retrieve just one part (e.g. 'bsp' or $appname) using the fragment (#) notation:

```
pvr get pantahub-ci/rpi3_initial_latest#bsp
67b1ce399971e30 [OK]
018e41b74500fa7 [OK]
fe4d959c5541950 [OK]
dee31a19ce47772 [OK]
```

You can also retrieve two mor more parts with , separated (e.g. 'bsp' and $appname) using the fragment (#) notation:

```
pvr get pantahub-ci/rpi3_initial_latest#bsp,$appname
...
```

Further can also remove one or multiple parts from repo in the same operation by prefixing the part with a '-', e.g. the following would get the latest bsp but also remove $appname from the pvr repo

```
pvr get pantahub-ci/rpi3_initial_latest#bsp,-$appname
...
```

You can also get from a tarball produced with `pvr export`:

```
pvr get /tmp/myexport.tar.gz#bsp
pulling objects file /tmp/pvr-tmprepo-544439698/objects/dee31a19ce4777282e8f388ee174ec214d60d2682cd2c09537f5656b1106bf3f-> /home/asac/.pvr/objects/
dee31a19ce4777282e8f388ee174ec214d60d2682cd2c09537f5656b1106bf3f.new
pulling objects file /tmp/pvr-tmprepo-544439698/objects/018e41b74500fa7109390f0505180cfb612146e7c1dcac1669b4df226110aa90-> /home/asac/.pvr/objects/
018e41b74500fa7109390f0505180cfb612146e7c1dcac1669b4df226110aa90.new
pulling objects file /tmp/pvr-tmprepo-544439698/objects/fe4d959c5541950fe8fb10fd13eaeee0cfa2a9bf3fcae37c825daeb12a5366d2-> /home/asac/.pvr/objects/
fe4d959c5541950fe8fb10fd13eaeee0cfa2a9bf3fcae37c825daeb12a5366d2.new
pulling objects file /tmp/pvr-tmprepo-544439698/objects/67b1ce399971e304b02aa4ad11049ae78a7c7a44652d89ef44a60a04b2b541b6-> /home/asac/.pvr/objects/
67b1ce399971e304b02aa4ad11049ae78a7c7a44652d89ef44a60a04b2b541b6.new
```

Rememember that `pvr get` will update the pristine state only, but not the working copy.

You would usually introspect retrieved changes first using:

```
pvr status
```

and then replace working copy with pristine state using pvr checkout:

```
pvr checkout
```

PVR GLOBAL-CONFIG

pvr global-config :Get Global Configuration details of the repo.

```
example1\$ pvr global-config
{
"Spec": "1",
"AutoUpgrade": false,
"DistributionTag": "develop"
}
```

PVR MERGE [REMOTE-OR-LOCAL-REPOSITORY[#PART]] [TARGET-REPOSITORY]

pvr merge : Merge content of repository into target-directory.Default target-repository is the local .pvr one. If not is provided the last one is used.

```
example1\$ pvr merge
e8305714eaadc57 [OK]
3f4889e5eed2252 [OK]
76d1d085d44fd3f [OK]
b56390fbeb5e46f [OK]
```

```
d49d56059ba219c [OK]
686a026cd606613 [OK]
70075ca4496451e [OK]
66ae7cac57d6d05 [OK]
989647b3d5b7fde [OK]
33aefd8dbf46f05 [OK]
612408620229de6 [OK]
f1b441cb2721355 [OK]
b30e6b64d3e1ecb [OK]
ff2a85a62cd09f1 [OK]
```

Same syntax for retrieving remote or local or just a part of a repository as for `pvr get` do apply.

PVR PUTOBJECTS

pvr putobjects : put objects from local repository to objects-endpoint

```
example1\$ pvr putobjects https://api.pantahub.com/objects
alpine-hotsp [OK]
bsp/kernel.i [OK]
pvr-sdk/root [OK]
bsp/firmware [OK]
pvr-sdk/root [OK]
alpine-hotsp [OK]
bsp/pantavis [OK]
pv-avahi/roo [OK]
bsp/pantavis [OK]
alpine-hotsp [OK]
pv-avahi/lxc [OK]
bsp/modules. [OK]
pvr-sdk/lxc. [OK]
```

PVR SELF-UPGRADE

pvr self-upgrade : Update pvr command to the latest version

```
example1$ $ sudo pvr self-upgrade
[sudo] password for nintriva:
Starting update PVR using Docker latest tag (sha256:0d6e747e75758535bdee89657902a1499e449db9510d688e0ef16d3171203975)

Downloading layers 8 ...
Done with [3/8] from repository
Done with [2/8] from repository
Done with [7/8] from repository
Done with [8/8] from repository
Done with [5/8] from repository
Done with [6/8] from repository
Done with [1/8] from repository
Done with [4/8] from repository

Extracting layers 8 ...

Pvr installed on /bin/pvr

Docker layers are going to be cache on: /root/.pvr/cache

PVR has been updated!
```

PVR CHECKOUT|RESET

pvr checkout|reset : checkout/reset working directory to match the repo stat.reset/checkout also forgets about added files; pvr status and diff will yield empty

```
example1\$ pvr checkout
```

PVR REGISTER [API_URL] -U \<USERNAME> -P \<PASSWORD> -E \<EMAIL>

pvr register : register new user account with pantahub

```
example1$ $ pvr register https://api.pantahub.com -e jogn123@gmail.com -u john123 -p 123

Your registration process needs to be complete two steps
1.- Confirm you aren't a bot
2.- Confirm your email address

Follow this link to continue and after that come back and continue
```

PVR APP COMMANDS

Since 022 pvr app commands are docker multi platform ready.

By default pvr will look for a priority ordered list of platforms that the device supports running natively in `_hostconfig/docker/config.json`.

The file structure is simple:

```
cat _hostconfig/pvr/docker.json
{
    "platforms": [
        "linux/arm",
        "linux/arm/v6"
    ]
}
```

If no matching architecture is found, pvr continues and falls back to the default that the docker_tag references.

To overwrite the defaults provide in the `_hostconfig/pvr/docker.json` file in the pvr repo, one can pass the --plaform=linux/XXX option to pvr app add:

```
pvr app add --platform=linux/arm/v5 --from=nginx:latest nginx
```

Similar to add, developers can conveniently change platform selection through the same `--platform` argument when using the `pvr app update` operation.

pvr app add --from= --source=[remote|local],[remote|local]

pvr app add creates a new application and generates files by pulling layers from a given docker image in either remote or local docker repo's. By default it will first look in remote repo. when not found it will pull from local docker repo,the priority can be changed using the --source flag(default:remote,local).

```
example1\$ pvr app add nginx-app --from=nginx --source=remote,local
Generating squashfs...
Downloading layers...
Layer 0 downloaded(cache)
Layer 1 downloaded(cache)
Layer 2 downloaded(cache)
Extracting layers...
Extracting layer 0
Extracting layer 1
Extracting layer 2
Stripping qemu files...
Deleted /home/nintriva/work/gitlab.com/pantacor/devices/10/nginx-app/download-layer/rootfs/usr/bin/qemu-arm-static file
Generating squashfs file
Generating squashfs digest
Application added
```

How to use rootfs type

We now can create an app, install an app, and update an app using as source a root filesystem, that filesystem could be as a tar in local for the computer is running the PVR, could be a plain folder with the filesystem inside, or could be an URL where the tar is to be downloaded.

Add app from rootfs

```
pvr app add --from=~/Desktop/pvwebstatus -t rootfs pvwebstatus
```

This will create a new application folder with these files:

```
pvwebstatus/
    lxc.container.conf
    root.squashfs
    root.squashfs.rootfs-digest
    run.json
    src.json
```

The Source file will have a rootfs_url argument and rootfs_digest to track the filesystem digest.

If you need to add some docker_config on creation to the app, you could use all the parameters supported. Example:

```
pvr app add --from=~/Desktop/pvwebstatus --config-json=pvwebstatus.config.json -t rootfs pvwebstatus
```

Where the `pvwebstatus.config.json` is a JSON configuration.

```
{
  "ArgsEscaped": true,
  "Cmd": [
    "/app/pantavisor-web-status"
  ],
  "Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
  ],
  "WorkingDir": "/app"
}
```

Add application that extends from another application

PVR and pantavisor support application that uses another application rootfs system as base and just create a diff squashfs for the files that are different in the added application.

That will allow to have smaller applications that share libraries or tools between several applications.

In this case for this to work the base system should be created as a data application

```
pvr app add --group=data --status-goal=MOUNTED --from=ros:rolling-ros-base rolling-ros-base
```

Then the application that will depend of

```
pvr app add --base rolling-ros-base --from=ANOTHER_CONTAINER extended_ros_app
```

Then after this if you want to sign or export the extended_ros_app we should signed with the base app.

```
pvr sig add --parts=rolling-ros-base,extended_ros_app
```

```
pvr export --parts=rolling-ros-base,extended_ros_app /path/to/exports/extended_ros_app.tgz
```

Make sure the base app and the extended app use the same architecture, to make sure the diff is as small as posible.

Example:

```
pvr app add --group=data --status-goal=MOUNTED --from=nginx:stable-alpine nginx_base

pvr app add --base nginx_base --from=highercomve/hello_world_nginx:latest hello_world_nginx
```

and if you take a look on https://hub.docker.com/_/nginx you will see that has the same architectures as https://hub.docker.com/r/highercomve/hello_world_nginx and the diff is going to be only the index.html and the configuration for the nginx.

How to use pvr type

Another source that could be used as a source for an application is a device description inside your computer or from a PV repository URL.

Add app from pvr

You could use any PVR repository compatible URL, which could be a local .pvr folder or a remote repository URL.

```
pvr app add -t pvr --from https://pvr.pantahub.com/highercomve/one_marketplace_production#tailscale tailscale
```

pvr app info

pvr app info :output info and state of appname

```
example1$ $ pvr app info nginx-app
{
"#spec": "service-manifest-src@1",
"args": {},
"config": {},
"docker_digest": "sha256:231d40e811cd970168fb0c4770f2161aa30b9ba6fe8e68527504df69643aa145",
"docker_name": "nginx",
"docker_source": "remote,local",
"docker_tag": "latest",
"persistence": {},
"template": "builtin-lxc-docker"
}
```

pvr app install

pvr app install rerun the template engine and reproduce any rootfs binary that is not present in the $APP_NAME directory.

For docker app install will take the docker_digest and docker_ovl_digest value from $APP_NAME/src.json and generate the matching rootfs and ovr files.

src.json

`src.json` is an input document used to store information about a Pantavisor container to be consumed by the `pvr app install` and `pvr app update` commands. It contains key-value pairs that define various aspects of the platform configuration.

`#spec`

- Version of the service manifest specification. Only value supported is "service-manifest-src@1".

`args`

- Platform arguments:
- `PV_GROUP` : Group to which the container belongs. Possible values: "data", "root", "platform" "app" or any additional group.
- `PV_RESTART_POLICY` : Restart policy for the container. Possible values: "system" or "container".
- `PV_ROLES` : Roles assigned to the platform. Possible values: "mgmt" or empty.
- `PV_SECURITY_WITH_STORAGE` : Enable/disable security with storage. Possible values: "yes" or "no".
- `PV_STATUS_GOAL` : Desired status goal for the platform. Possible values: "MOUNTED", "STARTED", or "READY".
- `PV_LXC_EXTRA_CONF` : Extra LXC configuration to be appended to lxc.container.conf.
- `PV_LXC_CAP_KEEP` : Specify the capabilities to be kept in the container.
- `PV_LXC_CAP_DROP` : Specify the capabilities to be kept in the container (note: use either keep or drop, but not both).
- `PV_LXC_NAMESPACE_KEEP` : Specify the namespaces to be inherited from Pantavisor.
- `PVR_LXC_CGROUP_DEVICES_WHITE` : Allow list of devices for the container. See the device indexes to be used here.
- `PV_VOLUME_IMPORTS` : Import a volume or a subpath within a volume; syntax: :[@]:[:ro]
- `PV_VOLUME_MOUNTS` : Mount volume owned by the container itself to aditional places; syntax: :
- `PV_IMPORT_CONFIGVOLUMES` : Mount configvolumes to a target path. configvolumes are volumes that can have configs for multiple containers inside. the top level directory of such volume would be the container name. Syntax: : (e.g for a container named 'foo' the directory /foo inside the '' volume would be mounted to ''
- `PV_DISABLE_AUTODEV` : disable lxc.autodev feature. Do this if you know what you are doing only.
- `PV_SECURITY_FULLDEV` : mount host /dev to dev/ inside container. IMPORTANT: usually you dont want this. Only do this if you know what you are doing.
- `PV_SECURITY_WITH_HOST` : mount host / partition to host/ directory inside container. Usually this is not a good idea.
- `PV_SECURITY_WITH_HOSTPROC` : mount /proc of host to host/proc mountpoint. Treat carefully.
- `PV_SECURITY_WITH_STORAGE` : make the pantavisor /storage partition (e.g. pvroot) available inside the container at /storage.
- `PV_RUN_TMPFS_DISABLE` : disable mounting a tmpfs to /run folder. You usually do not want to disable this.
- `PV_RESOLV_CONF_DISABLE` : do not mount the central host /etc/resolv.conf inside the container.

`docker_config`

- Docker container configuration:
- `ArgsEscaped` : Whether command-line arguments are escaped. Possible values: "true" or "false".
- `Cmd` : Command to run when starting the Docker container. Can be any binary path.
- `Env` : Environment variables for the Docker container. Can include any variables.
- `Volumes` : Volumes to be mounted by the Docker container. Can include any paths.
- `WorkingDir` : Working directory inside the Docker container. Can be any path.

`docker_digest`

- Digest (unique identifier) of the Docker image.

`docker_name`

- Name of the Docker image.

`docker_source`

- Source of the Docker image. Possible values: "remote" or "local".

`docker_tag`

- Tag associated with the Docker image.

`persistence`

- Persistence settings for specific directories:
- Each path can have the values: "permanent", "revision", or "boot".
- Optional suffix: "@dm-versatile" to enable the dm feature
- Optional prefix: "ovl" to set the dir as overlay.

`template`

- LXC configuration template used for creating the container.

Example of src.json:

```
{
    "#spec": "service-manifest-src@1",
    "args": {
        "PV_GROUP": "platform",
        "PV_LXC_EXTRA_CONF": "lxc.mount.entry = /volumes/_pv/addons/plymouth/text-io var/run/plymouth-io-sockets none bind,rw,optional,create=dir 0 0",
        "PV_RESTART_POLICY": "system",
        "PV_ROLES": [
            "mgmt"
        ],
        "PV_SECURITY_WITH_STORAGE": "yes",
        "PV_STATUS_GOAL": "STARTED",
        "PVR_LXC_CGROUP_DEVICES_WHITE": [
            "c 1:* rw",
            "c 10:130 mrw"
        ],
    },
    "docker_config": {
        "ArgsEscaped": true,
        "Cmd": [
            "/sbin/init"
        ],
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
            "PVR_DISABLE_SELF_UPGRADE=true",
            "PVR_CONFIG_DIR=/var/pvr-sdk/.pvr"
        ],
        "Volumes": {
            "/etc-volume": {},
            "/etc/dropbear": {},
            "/home/pantavisor/.ssh": {},
            "/var/pvr-sdk": {}
        },
        "WorkingDir": "/workspace"
    },
    "docker_digest": "registry.gitlab.com/pantacor/pv-platforms/pvr-sdk@sha256:08c5339cc11bcde781bfa0e4662fe7449e3a07326323ec94e847d56913acbd49",
    "docker_name": "registry.gitlab.com/pantacor/pv-platforms/pvr-sdk",
    "docker_source": "remote,local",
    "docker_tag": "arm32v6",
    "persistence": {
        "/var/dmcrypt/volume": "permanent@dm-versatile",
        "/var/permanent": "permanent",
        "/var/revision": "revision",
        "/var/boot": "boot"
   },
   "template": "builtin-lxc-docker"
}
```

pvr app ls

pvr app ls :list applications in pvr checkout

```
example1$ $ pvr app ls
alpine-hotspot
app1
app2
nginx-app
pv-avahi
pvr-sdk
```

pvr app rm

pvr app rm : remove app from pvr checkout

```
example1$ $ pvr app rm app1
\$ pvr app ls
alpine-hotspot
app2
nginx-app
pv-avahi
pvr-sdk
```

pvr app update

pvr app update :update an existing application.

```
example1$ $ pvr app update nginx-app
Application updated
```

`app install` will generate a diff overlay squashfs allowing to ship minor updates as a diff layer.

PVR DEPLOY [SOURCE-REPOS]+

With pvr deploy you can deploy one to many source repos to a deployment directory such as the one you find on the pantavisor device for each revision.

This command can be used to modify a rootfs and change/replace/update apps.

Example, will deploy the 'os' container from a local repository, the 'bsp' container from a pvr export to a factory revision (trails/0) in a pantavisor enabled rootfs.

```
pvr deploy trails/0  /path/to/repo/.pvr#os /tmp/export.tgz#bsp
```

This command will create hardlinks of the objects to the objects pool so do not use this on a host where you intend the checkout to be edited.

Update app from rootfs

Here you can update like a normal docker container:

```
pvr app update pvwebstatus
```

But you can update from another tar or an URL

```
pvr app update --from=~/Desktop/pvwebstatus.tar pvwebstatus
```

PVR SIG COMMANDS

PVR sig commands offers support for maintaining pvs signatures inside your tree.

For details see README.pvs.md

Commands currently supported are: * Review all object and file usage cases to ensure tall are validated before they are installed/used.

- pvr sig add - adds a new signature to the _pvs/ hierarchy of the state
- pvr sig update - updates a committed signature from the _pvs/ hierarchy to be validate against committed state
- pvr sig ls - list files covered by signatures in *sigs/ hieararchy; by default sig ls will show signature info while considering _all* signatures in the system state

PVR sig with CA commands

```
commit c2d6f1450422b89d90948499c7cd6dd6949e5df3 (HEAD -> feature/pvs-ca, origin/feature/pvs-ca)
Author: Alexander Sack <asac@pantacor.com>
Date:   Wed Oct 20 17:58:50 2021 +0200

    add support for using x509 cert chains using x5c jws header to determine trust in pvr signatures

    * introduce new --x5c argument pvr app sig command to provide the chain to include in pvr sig add and update commands
    * introduce --cacerts argument to pvr sig commands to allow to post a trust CACERTS file to use to validate in pvr app ls;
      using special value _system will use the system cert store to validate ca chain
    * pubkey validation now allows to have multiple trusted pubkeys in the file referenced by --pubkey
    * document this feature in README.md

    Example 1: "add signature with trust ca chain"

    Below statement injects the myKey.crt as the trust chain into the jws.
    If you have intermediates your .crt file would need to include those
    also in reverse order.

    ```
    pvr sig --x5c ../ca/myKey.crt --key ../ca/myKey.key add --part nginx
    ```

    Example 2: "update signatures with trustchain"

    Below will refresh the nginx.json signature and attach myKey.crt as
    the trust ca cert chain to validate against root certificates

    ```
    pvr sig --x5c ../ca/myKey.crt --key ../ca/myKey.key update _sigs/nginx.json
    ```

    Example 3: "validate signatures with cert pool in file"

    Below you can see how to validate signature with ca cert pool in file myCA.pem.

    ```
    pvr sig --cacerts ../ca/myCA.pem ls --part _sigs/nginx.json
    ```

    Example 4: use system ca cert pool to validate signature

    For this you have to put your myCA.pem into one of the system folders for
    trusted certificates. e.g. /etc/ssl/certs

    ```
    pvr sig ls --part _sigs/nginx.json
    ```
```

PVR sig options

To allow easier integration in higher level tools, like external signing tools `pvr sig` offers a few options to make their live easier.

The idea of making an external signing tool is to keep the logic that interprets the PVS headers and create the protected header and the payload of the jose token inside pvr itself, but allow for easy export of the complete jose token including payload which then external tools can inspect and sign.

pvr sig add/update with payload

The first avenue is to produce a signature during signing that includes the full payload. This can be achieved through the `--with-payload` option for `pvr sig add` and `pvr sig update`. Example:

```
$ pvr clone pantahub-ci/rock64_initial_stable example
$ cd example
$ pvr sig --with-payload add --part awconnect
```

This will produce a jose signature file in _sigs/awconnect.json that includes the full payload and hence can be send to a service that resigns it in infrastructure.

To make this more conveniebnt, the --output=- option allows to print that signature to stdout:

```
$ pvr sig --with-payload add --part awconnect
```

pvr sig ls with full jose token

The second approach that might be viable is to take a package that was already signed by a developer or CI system, inspect it and resign it with a production key.

For that the `pvr sig ls` command offers some convenience to include the jose serialization found in the result summary with the `--with-sig` option.

Combined with the `--with-payload` option this gives the user the ability to again a produce a complete jose token that an external system can resign without further business logic. Example:

```
$ pvr sig --with-payload ls --with-sig _sigs/awconnect.json
```

PVR DM COMMANDS (DEVICE-MAPPER) (BETA)

pvr device mapper support for container volumes allows for an easy way to postprocess the squashfs volumes produced by pvr app add etc. in a way that the pantavisor device mapper addon can mount volumes using device-mapper.

For now dm-verity type device mapper entries are supported by pantavisor client and hence pvr supports that mode first and foremost for now.

PVR dm-convert (BETA)

This command allows you to convert any standard squashfs volume into a device-mapper mounted volume.

For dm-verity this will create a manifest file in /_dm/.json, e.g. os/_dm/root.squashfs.json.

To convert a volume simply use the following command:

```
pvr dm-convert os root.squashfs
```

This will convert the root.squashfs volume of the container 'os' into a device mapper enabled one.

It will create the os/_dm/root.squashfs.json manifest:

```
$ cat os/_dm/root.squashfs.json | jq .
{
  "type": "dm-verity"
  "data_device": "root.squashfs",
  "hash_device": "root.squashfs.hash",
  "root_hash": "88298f349288e685ac2474134ef22bf8f77465cde250c9f698780b5b6d942b96",
}
```

... and also the hash_device in `os/root.squashfs.hash`.

Also it will convert the volume reference in run.json to "dm:<volume", e.g. `dm:rootfs.squashfs`. This syntax will ensure that pantavisor will not try to mount the squash himself, but rather delegate that to a device-mapper volume handler.

You can then `pvr commit` this and post it to a pantavisor device-mapper enabled device

PVR dm-apply (BETA)

This command iterates through the whole committed pristine json of the repo and updates the hash and manifests for all dm-verity manifests.

This is good if you updated a container and want to recalculate the hash file.

7.6 Pantacor Hub API

7.6.1 Third Party Application Service

This service handle the creation, update, read and delete of third party applications. This applications can use pantahub oauth to authenticate their users and ask permission to different scopes inside pantahub base and can create their own scopes.

For most of the endpoints you will need a TOKEN to identify the owner of the application

Login

```
TOKEN=`http localhost:12365/auth/login username=user1 password=user1 | json token`
```

Retrive Pantahub avaliable scopes (Public endpoint)

```
curl --request GET \
  --url http://localhost:12365/apps/scopes \
  --header 'content-type: application/json'
```

Response:

```
[
  {
    "id": "all",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Complete Access"
  },
  {
    "id": "user.readonly",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read only user"
  },
  {
    "id": "user.write",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Write only user"
  },
  {
    "id": "devices",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read/Write devices"
  },
  {
    "id": "devices.readonly",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read only devices"
  },
  {
    "id": "devices.write",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Write only devices"
  },
  {
    "id": "devices.change",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Update devices"
  },
  {
    "id": "objects",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read/Write only objects"
  },
  {
    "id": "objects.readonly",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read only objects"
  },
  {
    "id": "objects.write",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Write only objects"
  },
  {
    "id": "objects.change",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Update objects"
  },
  {
    "id": "trails",
    "service": "prn:pantahub.com:apis:/base",
```

```
    "description": "Read/Write only trails"
  },
  {
    "id": "trails.readonly",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read only trails"
  },
  {
    "id": "trails.write",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Write only trails"
  },
  {
    "id": "trails.change",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Update trails"
  },
  {
    "id": "metrics",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read/Write only metrics"
  },
  {
    "id": "metrics.readonly",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Read only metrics"
  },
  {
    "id": "metrics.write",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Write only metrics"
  },
  {
    "id": "metrics.change",
    "service": "prn:pantahub.com:apis:/base",
    "description": "Update metrics"
  }
]
```

Create App

In order to create an app you need to send a json with this 3 obligatory properties:

- type: One application can have two types (public|condidential) (more about that in here)[https://tools.ietf.org/html/rfc6749#section-2.1].
- redirect_uris: This is an array of string with the URLs where can redirect the oauth service to sent the token or code.
- scopes: this is an array of scopes, this set a approved list of scopes that can be asked to the user to give permission.

```
curl --request POST \
  --url http://localhost:12365/apps/ \
  --header 'authorization: Bearer $TOKEN' \
  --header 'content-type: application/json' \
  --data '{
    "type": "public",
    "redirect_uris": ["http://localhost/return_url"],
    "scopes": [
        {
            "id": "all",
            "service": "prn:pantahub.com:apis:/base"
        }
    ]
}'
```

Response:

```
{
  "id": "5e0a658db0acd7109320fbe0",
  "type": "public",
  "nick": "secretly_better_grouper",
  "prn": "prn:pantahub.com:apis:/5e0a658db0acd7109320fbe0",
  "owner": "prn:::accounts:/5dfaac1b883859b4de940ca9",
  "owner-nick": "highercomve",
  "secret": "ct6bzdrzhaya7ezc75wy2ocuw6qz1v",
  "redirect_uris": [
    "http://localhost/return_url"
  ],
  "scopes": [
    {
      "id": "all",
      "service": "prn:pantahub.com:apis:/base",
      "description": "Complete Access"
    }
  ],
  "time-created": "2019-12-30T21:01:01.253338883Z",
  "time-modified": "2019-12-30T21:01:01.253338883Z"
}
```

Get all apps of a user

```
curl --request GET \
  --url http://localhost:12365/apps/ \
  --header 'authorization: Bearer $TOKEN' \
  --header 'content-type: application/json'
```

Get app by ID

```
curl --request GET \
  --url http://localhost:12365/apps/5e0a658db0acd7109320fbe0 \
  --header 'authorization: Bearer $TOKEN' \
  --header 'content-type: application/json'
```

Update app

```
curl --request PUT \
  --url http://localhost:12365/apps/5e0a658db0acd7109320fbe0 \
  --header 'authorization: Bearer $TOKEN' \
  --header 'content-type: application/json' \
  --data '{
    "type": "public",
    "redirect_uris": [
        "http://posibleappurl.com/oauth2/cb",
        "https://posibleappurl.com/oauth2/cb"
    ],
    "scopes": [
        {
            "id": "all",
            "service": "prn:pantahub.com:apis:/base",
            "description": "Complete Access"
        },
        {
            "id": "programs.all",
            "description": "Read/write programs from the thirdparty application"
        }
    ]
}'
```

Delete APP

```
curl --request DELETE \
  --url http://localhost:12365/apps/5e0a658db0acd7109320fbe0 \
  --header 'authorization: Bearer $TOKEN' \
  --header 'content-type: application/json'
```

7.6.2 Auth

Using the Auth API is simple. For now we recommend using httpie command line tool ('http' below).

Start Service

Start your server:

```
./pantahub-serv
```

Register a user

The registration process is a two steps flow when you are using the CLI or the API, in that case in order to register:

1.- simply POST your details to the accounts endpoint. If nick and email are not taken it will response with a token and redirect-uri to continue the second step

2.- Open the browser with the redirect-uri resolve the captcha challenge and finish the process. If there is any error the system will send a registration confirm email out.

```
curl --request POST \
  --url http://localhost:12365/auth/accounts \
  --header 'content-type: application/json' \
  --data '{
    "email": "sergio.marin@pantacor.com",
    "nick": "sergiomarin",
    "password": "1234567890"
}'

{
    token: "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkExMjhHQ00iLCJ0eXAiOiJKV1QifQ.ExjbJW5nNxOKftqAZ0RAqMfE5Q20bzm2fcICJM-
r0fqRqn3mbBadVuZrk5NqYJrCH2YfBU4Jt1yhDBjlZwM1HM4QU7Xau6IeX8h4n8kyYDd819Ra9uisYfnq2lT2mAnS-08RaFDmjh3B2o--
JZDydIEAa7hXDlo7sXuIPXayj3oQVLhhpLkaTII5XYy1U47_7hPHGTf5MbXbK79yYzPS1EqtWDjLpxIiViY0oMZUgZLP_IZ81rAD_0zdTrKeh_mBUefN0d2fpcWtyH3zSGX_o6-N5-
MASFc9LgVfQDBymv_9XkG_MAggv3pmfFQXeRunJZdIXJujAgFh9rEg67voOA.Eb8SnMx_c0ImvlVb.O8yTX9gqbx4a5oTGgHEMEmOBWw_aibkQs6fjgazs6OZH9u1lRXil3MSzJ4D0NlAtS-jH_Rz_hQ6gt-m-
SuM_JYH-2T_-KJ67n1VugApaATIADWX-7r_d-oYkM50qcvC7U3tmiWLQ3WHveSi-1JpPPG6Ukuuw9G7itbLMpoi2AzHpMtrZPyjIv5A6JdCIFPlLax8N2krHA2lRqv24596Vv4zy_w1Y9nb3RDLX4F81F33t0VA-
dE-i-Bay2RH2KH9dfm9CmlvxWpk8NwzKCgoHBHuzm7dhEVAlfNU.ucIQP10FLHcgsSNflIrO9Q",
    "redirect-uri": "http://localhost:3000/signup#account=eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkExMjhHQ00iLCJ0eXAiOiJKV1QifQ.ExjbJW5nNxOKftqAZ0RAqMfE5Q20bzm2fcICJM-
r0fqRqn3mbBadVuZrk5NqYJrCH2YfBU4Jt1yhDBjlZwM1HM4QU7Xau6IeX8h4n8kyYDd819Ra9uisYfnq2lT2mAnS-08RaFDmjh3B2o--
JZDydIEAa7hXDlo7sXuIPXayj3oQVLhhpLkaTII5XYy1U47_7hPHGTf5MbXbK79yYzPS1EqtWDjLpxIiViY0oMZUgZLP_IZ81rAD_0zdTrKeh_mBUefN0d2fpcWtyH3zSGX_o6-N5-
MASFc9LgVfQDBymv_9XkG_MAggv3pmfFQXeRunJZdIXJujAgFh9rEg67voOA.Eb8SnMx_c0ImvlVb.O8yTX9gqbx4a5oTGgHEMEmOBWw_aibkQs6fjgazs6OZH9u1lRXil3MSzJ4D0NlAtS-jH_Rz_hQ6gt-m-
SuM_JYH-2T_-KJ67n1VugApaATIADWX-7r_d-oYkM50qcvC7U3tmiWLQ3WHveSi-1JpPPG6Ukuuw9G7itbLMpoi2AzHpMtrZPyjIv5A6JdCIFPlLax8N2krHA2lRqv24596Vv4zy_w1Y9nb3RDLX4F81F33t0VA-
dE-i-Bay2RH2KH9dfm9CmlvxWpk8NwzKCgoHBHuzm7dhEVAlfNU.ucIQP10FLHcgsSNflIrO9Q"
}
```

In this case you will receive the confirmation link on the console of your pantahub server instance:

```
    To verify your account, please click on the link: <a href="http://localhost:12365/auth/verify?id=58dcb86bc094f66125a698dd&challenge=yuieui5a0ost1l2">http://
localhost:12365/auth/verify?id=58dcb86bc094f66125a698dd&challenge=yuieui5a0ost1l2</a><br><br>Best Regards,<br><br>A. Sack and R. Mendoza (Pantacor Founders)
```

Simply open this url and you will be able to log in from here now.

Authenticate

```
http POST localhost:12365/auth/login username=user1 password=user1

HTTP/1.1 200 OK
Content-Length: 256
Content-Type: application/json; charset=utf-8
Date: Fri, 19 Aug 2016 12:11:03 GMT
X-Auth-Accesstoken:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhYnJuIjoiYWJybjphYmNpZS54eXo6YXV0aDovdXNlcjEiLCJleHAiOjE0NzE2MTIyNjMsImlkIjoidXNlcjEiLCJvcmlnX2lhdCI6MTQ3MTYwODY2Mywicm9
sZXMiOiJ1c2VyIiwidHlwZSI6IlVTRVIifQ.Fdwmbphn_OA7nBe9jWvWbfCbuiKcBtD0rQqEoZFBIRk
X-Powered-By: go-json-rest

{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhYnJuIjoiYWJybjphYmNpZS54eXo6YXV0aDovdXNlcjEiLCJleHAiOjE0NzE2MTIyNjMsImlkIjoidXNlcjEiLCJvcmlnX2lhdCI6MTQ3MTYwODY2Mywicm
9sZXMiOiJ1c2VyIiwidHlwZSI6IlVTRVIifQ.Fdwmbphn_OA7nBe9jWvWbfCbuiKcBtD0rQqEoZFBIRk"
}
```

Note down the token either from json body or header for further API access.

You can use the json tool to do so automatically without hazzle:

```
TOKEN=`http POST localhost:12365/auth/login username=user1 password=user1 | json token`
```

Account Classes

- USER - human/botty users with id, email and secret
- DEVICE - devices with id and secret
- SERVICE - API services with id, location and secret

Available users

- user1:user1
- user2:user2
- device1:device1
- device2:device2
- service1:service1
- service2: service2
- service3: service3

Refreh Token

To get a refreshed token, use the GET method with the Bearer token on the api/auth/login endpoint:

```
http GET localhost:12365/auth/login  Authorization:"Bearer $TOKEN"
HTTP/1.1 200 OK
Content-Length: 256
Content-Type: application/json; charset=utf-8
Date: Wed, 28 Sep 2016 11:22:57 GMT
X-Auth-Accesstoken:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhYnJuIjoiYWJybjphYmNpZS54eXo6YXV0aDovdXNlcjEiLCJleHAiOjE0NzUwNjUzNzcsImlkIjoidXNlcjEiLCJvcmlnX2lhdCI6MTQ3NTA2MTI5Mywicm9
sZXMiOiJ1c2VyIiwidHlwZSI6IlVTRVIifQ.R2Em_nvxzYq--EBAEXW3WKTo558PN_VwmAc4TVJ_-ek
X-Powered-By: go-json-rest

{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhYnJuIjoiYWJybjphYmNpZS54eXo6YXV0aDovdXNlcjEiLCJleHAiOjE0NzUwNjUzNzcsImlkIjoidXNlcjEiLCJvcmlnX2lhdCI6MTQ3NTA2MTI5Mywicm
9sZXMiOiJ1c2VyIiwidHlwZSI6IlVTRVIifQ.R2Em_nvxzYq--EBAEXW3WKTo558PN_VwmAc4TVJ_-ek"
}
```

Device registry authentication

Auth API now suppors authenticating against real device registry entries. just pass the full prn in as username and the secret that you have put into the device registry to do this.

Example

```
$ http POST localhost:12365/auth/login username="prn:::devices:/57ebaaddc094f6188d000002" password="yourdevicesecret"
HTTP/1.1 200 OK
Content-Length: 376
Content-Type: application/json; charset=utf-8
Date: Wed, 28 Sep 2016 16:04:07 GMT
X-Auth-Accesstoken:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhYnJuIjoiYWJybjo6OmRldmljZXM6LzU3ZWJhYWRkYzA5NGY2MTg4ZDAwMDAwMiIsImV4cCI6MTQ3NTA4MjI0NywiaWQiOiJhYnJuOjo6ZGV2aWNlczovNTd
lYmFhZGRjMDk0ZjYxODhkMDAwMDAyIiwib3JpZ19pYXQiOjE0NzUwNzg2NDcsIm93bmVyIjoiYWJybjphYmNpZS54eXo6YXV0aDovdXNlcjEiLCJyb2xlcyI6ImRldmljZSIsInR5cGUiOiJERVZJQ0UifQ.
7_lwQB2mk-ZvuLrNzbk1Wg_UxGe5QQp9Nr9YbhEPq8w
X-Powered-By: go-json-rest

{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhYnJuIjoiYWJybjo6OmRldmljZXM6LzU3ZWJhYWRkYzA5NGY2MTg4ZDAwMDAwMiIsImV4cCI6MTQ3NTA4MjI0NywiaWQiOiJhYnJuOjo6ZGV2aWNlczovNT
dlYmFhZGRjMDk0ZjYxODhkMDAwMDAyIiwib3JpZ19pYXQiOjE0NzUwNzg2NDcsIm93bmVyIjoiYWJybjphYmNpZS54eXo6YXV0aDovdXNlcjEiLCJyb2xlcyI6ImRldmljZSIsInR5cGUiOiJERVZJQ0UifQ.
7_lwQB2mk-ZvuLrNzbk1Wg_UxGe5QQp9Nr9YbhEPq8w"
}
```

7.6.3 Service authorization with access tokens (aka oauth2'ish authorization flow)

Services are just like normal user accounts to be authenticated through the /auth/login endpoint.

Services/Clients can impersonate a user through a token exchange inspired by oauth2.

For that the service has to request from the user to issue a code with certain access scopes.

The user then uses the /auth/code endpoint to issue such accesscode and hands it over to the service/client who in turn swaps out that code for a long-lived access-token.

For example the following steps will show how the authorization flow could look like:

Step 1 - user authenticates to pantahub

```
UTOK=`http http://localhost:12365/auth/login username=user1 password=user1 | jq -r .token`
```

Step 2 - user issues authorization code for service1L

```
CODE=`http http://localhost:12365/auth/code Authorization:" Bearer $UTOK" service="prn:pantahub.com:auth:/service1" scopes="*" | jq -r .code`
```

Step 3 - service authenticates itself with pantahub

```
STOK=`http http://localhost:12365/auth/login username=service1 password=service1 | jq -r .token`
```

Step 4 - service requests swaps code for token

```
OTOK=`http http://localhost:12365/auth/token Authorization:" Bearer $STOK" access-code="$CODE" | jq -r .token`
```

Step 5. service uses access token to access pantahub on behalf of user

```
http http://localhost:12365/auth/auth_status Authorization:" Bearer $OTOK"
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: application/json; charset=utf-8
Date: Wed, 20 Feb 2019 23:39:45 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000000

{
    "aud": "prn:pantahub.com:auth:/service1",
    "id": "prn:pantahub.com:auth:/user1",
    "nick": "user1",
    "prn": "prn:pantahub.com:auth:/user1",
    "roles": "admin",
    "scopes": "*",
    "token_id": "5c6de5279c8c94c4dc06f067",
    "type": "USER"
}
```

7.6.4 sudo: Admins an log in as any user

If your user has the "admin" role you get the ability to support other users.

Get Info about all accounts

As "admin" user you can query the /accounts endpoint to retrieve account info of any user in the system:

```
http http://localhost:12365/auth/accounts?asadmin=yes Authorization:" Bearer $TOK"

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 18 Mar 2019 09:49:08 GMT
Transfer-Encoding: chunked
X-Powered-By: go-json-rest
X-Runtime: 0.006500

[
    {
        "email": "asac@pantacor.com",
        "nick": "asac",
        "prn": "prn:::accounts:/58dc21d76e2bc30224f160b0"
        "time-created": "2017-03-29T23:06:31.345+02:00",
```

```
        "time-modified": "2017-03-29T23:08:56.688+02:00",
        "type": "USER"
    },
    {
...
```

login as another user

To login as another user you can use the /auth/login endpoint by specifying the special username: "$youradminuser⟹$loginasuser", e.g.

```
http POST  localhost:12365/auth/login username='user1==>user2' password=user1

HTTP/1.1 200 OK
Content-Length: 469
Content-Type: application/json; charset=utf-8
Date: Mon, 18 Mar 2019 09:53:08 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000183

{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjYWxsLWFzIjp7ImlkIjoicHJuOnBhbnRhaHViLmNvbTphdXRoOi91c2VyMiIsIm5pY2siOiJ1c2VyMiIsInBybiI6InBybjpwYW50YWh1Yi5jb206YXV0aDovdXNlcjIiLCJyb2xlcyI6ImFkbWluIiwidHlwZSI6IlVTRVIifSwiZXhwIjoxNTUyOTA2Mzg4LCJpZCI6InVzZXIxPT1cdTAwM2V1c2VyMiIsIm5pY2siOiJ1c2VyMSIsIm9yaWdfaWF0IjoxNTUyOTAyNzg4LCJwcm4iOiJwcm46cGFudGFodWIuY29tOmF1dGg6L3VzZXIxIiwicm9sZXMiOiJhZG1pbiIsInR5cGUiOiJVU0VSIn0.yhUwT4ExaY0KyO2_uRDlHb9kOp04lEvoL8MZ2ui3_Sk"
}
```

7.6.5 Client authorization with oauth'ish approach

To mimic oauth2 implicit flow we offer /authorize endpoint that allows user to issue directly accesstoken for a preregistered client with redirect URI.

To use simply call /authorize endpoint as user:

```
OTOK=`http POST localhost:12365/auth/authorize service=prn:pantahub.com:auth:/client1 scopes='*' redirect_uri=http://localhost:808 Authorization:" Bearer $TOK"
| jq -r .token`
```

And client can then use that token to call api on behalf of user given the selected scopes:

```
http POST localhost:12365/auth/authorize
calhost:12365/devices/auth_status Authorization:" Bearer $OTOK"
HTTP/1.1 200 OK
Content-Length: 279
Content-Type: application/json; charset=utf-8
Date: Wed, 06 Mar 2019 13:38:42 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000193

{
    "aud": "prn:pantahub.com:auth:/client1",
    "exp": "2019-03-06T15:21:28.929136745+01:00",
    "id": "prn:::accounts:/59ef9e241e7e6b000d3d2bc7",
    "nick": "asacasa",
    "prn": "prn:::accounts:/59ef9e241e7e6b000d3d2bc7",
    "roles": "admin",
    "scopes": "*",
    "token_id": "5c7fc958c094f6720b6a9bc8",
    "type": "USER"
}
```

Implicit access tokens need renewal after given expiry timeout (currently 1h)

7.6.6 Device

PANTAHUB Device Registry

NOTE: the auth service does not take the secrets here into account yet!

Start Service

Start your server:

```
./pantahub-serv
```

Login

```
TOKEN=`http localhost:12365/auth/login username=user1 password=user1 | json token`
```

... will store access token in TOKEN for requests below

Upload File

REGISTER A DEVICE (AS USER)

```
http POST localhost:12365/devices/  Authorization:"Bearer $TOKEN" \
    secret="yourdevicesecret"

HTTP/1.1 200 OK
Content-Length: 331
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 21:13:08 GMT
X-Powered-By: go-json-rest

{
    "challenge": "likely-creative-troll",
    "device-meta": {},
    "id": "596690e4632d7234e270360f",
    "nick": "proper_sponge",
    "owner": "prn:pantahub.com:auth:/user1",
    "prn": "prn:::devices:/596690e4632d7234e270360f",
    "secret": "yourdevicesecret",
    "time-created": "2017-07-12T23:13:08.199223323+02:00",
    "time-modified": "0001-01-01T00:00:00Z",
    "user-meta": {}
}
```

REGISTER A DEVICE FOR CLAIMING (AS DEVICE)

To allow distribution of images and to allow transfer of ownership, devices that have no owner can have a challenge field that allows an authenticated user to claim it through its PUT endpoint.

In example of generic image a device would first register itself with its own secret to the hub and would present the user with a challenge that he can use to claim it.

Example:

1. device registers itself

```
http POST localhost:12365/devices/ secret="mysec1"
HTTP/1.1 200 OK
Content-Length: 286
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 21:14:34 GMT
X-Powered-By: go-json-rest

{
    "challenge": "duly-helped-bat",
    "device-meta": {},
    "id": "59669139632d7234e2703610",
    "nick": "immortal_worm",
    "owner": "",
    "prn": "prn:::devices:/59669139632d7234e2703610",
    "secret": "mysec1",
    "time-created": "2017-07-12T23:14:33.97805159+02:00",
    "time-modified": "0001-01-01T00:00:00Z",
```

```
    "user-meta": {}
}
```

Note how the output has a challenge entry, but no owner yet...

1. extract the challenge from the json and display to user

```
challenge=duly-helped-bat
```

1. as a logged in user with TOKEN you claim the device through a simple PUT

```
http PUT localhost:12365/devices/59669139632d7234e2703610?challenge=$challenge Authorization:"Bearer $TOKEN"
HTTP/1.1 200 OK
Content-Length: 309
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 21:15:49 GMT
X-Powered-By: go-json-rest

{
    "challenge": "",
    "device-meta": {},
    "id": "59669139632d7234e2703610",
    "nick": "immortal_worm",
    "owner": "prn:pantahub.com:auth:/user1",
    "prn": "prn:::devices:/59669139632d7234e2703610",
    "secret": "mysec1",
    "time-created": "2017-07-12T23:14:33.978+02:00",
    "time-modified": "2017-07-12T23:15:49.078077992+02:00",
    "user-meta": {}
}
```

As you can see the challenge field is now reset and the owner is assigned.

GET YOUR DEVICES

... only yours!

```
http localhost:12365/devices/  Authorization:"Bearer $TOKEN"
HTTP/1.1 200 OK
Content-Length: 345
Content-Type: application/json; charset=utf-8
Date: Tue, 04 Oct 2016 20:43:25 GMT
X-Powered-By: go-json-rest

[
    {
        "prn": "prn:::devices:/57f41438b376a825cf000001",
        "id": "57f41438b376a825cf000001",
        "nick": "desired_stud",
        "owner": "prn:pantahub.com:auth:/user1",
        "secret": "yourdevicesecret"
    },
    {
        "prn": "prn:::devices:/57f4146bb376a825cf000002",
        "id": "57f4146bb376a825cf000002",
        "nick": "composed_pheasant",
        "owner": "prn:pantahub.com:auth:/user1",
        "secret": "anotherdevice"
    }
]

## select your device here from above list
DEVICEID=57f41438b376a825cf000001
```

CHANGE DEVICE SECRET

```
http PUT localhost:12365/devices/$DEVICEID  Authorization:"Bearer $TOKEN" \
    secret="mynewsecret1"

HTTP/1.1 200 OK
Content-Length: 324
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 21:16:58 GMT
X-Powered-By: go-json-rest

{
    "challenge": "",
    "device-meta": {},
    "id": "59669139632d7234e2703610",
    "nick": "immortal_worm",
    "owner": "prn:pantahub.com:auth:/user1",
    "prn": "prn:::devices:/59669139632d7234e2703610",
    "secret": "mynewsecret1",
    "time-created": "2017-07-12T23:14:33.978+02:00",
    "time-modified": "2017-07-12T23:16:58.707242972+02:00",
```

```
    "user-meta": {}
}
```

Fill in User Metadata

To fill in user metadata you need to be logged in as user like for all operations above:

```
http PUT localhost:12365/devices/$DEVICEID/user-meta Authorization:" Bearer $TOKEN" \
    some=user meta=datafields
HTTP/1.1 200 OK
Content-Length: 35
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 21:21:18 GMT
X-Powered-By: go-json-rest

{
    "meta": "datafields",
    "some": "user"
}
```

Afterwards your device will have this metadata filled in:

```
http GET localhost:12365/devices/$DEVICEID Authorization:" Bearer $TOKEN"
HTTP/1.1 200 OK
Content-Length: 351
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 21:21:47 GMT
X-Powered-By: go-json-rest

{
    "challenge": "",
    "device-meta": {},
    "id": "59669139632d7234e2703610",
    "nick": "immortal_worm",
    "owner": "prn:pantahub.com:auth:/user1",
    "prn": "prn:::devices:/59669139632d7234e2703610",
    "secret": "mynewsecret1",
    "time-created": "2017-07-12T23:14:33.978+02:00",
    "time-modified": "2017-07-12T23:16:58.707+02:00",
    "user-meta": {
        "meta": "datafields",
        "some": "user"
    }
}
```

Fill in Device Metadata

To fill in user metadata you need to be logged in as the device:

```
DTOKEN=`http localhost:12365/auth/login username=prn:::devices:/$DEVICEID password=mynewsecret1 | json token`
```

After that you can update device metadata in same way as the user meta data above using the device credentials:

```
http PUT localhost:12365/devices/$DEVICEID/device-meta Authorization:" Bearer $DTOKEN" some=device meta=datafields work=too
HTTP/1.1 200 OK
Content-Length: 50
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 21:26:49 GMT
X-Powered-By: go-json-rest

{
    "meta": "datafields",
    "some": "device",
    "work": "too"
}
```

Share your device with the world; mark it as public

To mark device as public you can either update the public field using a full PUT on the device you you can use the convenience endpoint `/devices/:id/public`:

```
http PUT localhost:12365/devices/$DEVICEID/public Authorization:" Bearer $TOKEN"
HTTP/1.1 200 OK
Content-Length: 329
Content-Type: application/json; charset=utf-8
Date: Wed, 18 Oct 2017 20:09:55 GMT
```

```
X-Powered-By: go-json-rest

{
...
    "public": true,
...
}
```

Unshare your devices; unmark them public

If you change your mind simply use the DELETE method on the public endpoint:

```
http PUT localhost:12365/devices/$DEVICEID/public Authorization:" Bearer $TOKEN"
HTTP/1.1 200 OK
Content-Length: 330
Content-Type: application/json; charset=utf-8
Date: Wed, 18 Oct 2017 20:09:52 GMT
X-Powered-By: go-json-rest

{
...
    "public": false,
...
}
```

Change device nick

To change device nick, use the PATCH method on the device resource, e.g.

```
http PATCH localhost:12365/devices/$DEVICEID/ nick=mynewnick Authorization:" Bearer $TOKEN"

HTTP/1.1 200 OK
Content-Length: 273
Content-Type: application/json; charset=utf-8
Date: Wed, 23 May 2018 20:43:10 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000977

{
    "device-meta": {},
    "id": "5aadaf259c8c9433706048ab",
    "nick": "mynewnick",
    "owner": "prn:pantahub.com:auth:/user1",
    "prn": "prn:::devices:/XXXXXXXXXXXXXXXXXXXXXXXXXX",
    "public": false,
    "time-created": "2018-03-18T01:13:25.11+01:00",
    "time-modified": "0001-01-01T00:00:00Z",
    "user-meta": {}
}
```

If nick is already taken, clients will get a 409 (Conflict) http status code:

```
http PATCH http://localhost:12365/devices/$DEVICE_ID Authorization:" Bearer $TOK" nick=already-taken
HTTP/1.1 409 Conflict
Content-Length: 45
Content-Type: application/json; charset=utf-8
Date: Wed, 23 May 2018 20:40:07 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.001955

{
    "Error": "Device unique constraint violated"
}
```

Auto Assign Devices to Owners

Pantahub base offers a built in basic factory story in the sense that we offer the ability to auto assing devices to a specific owner.

For that right now we use a simple token based approach:

1. Owner uses `/devices/tokens/` end point to create a new token; optionally he can also provide a set of default user-meta information that the auto assign feature will put in place for every device joinig using such token.
2. Token is a one-time-visible secret that will only be displayed on reply of the token registration, but not afterwards. If user looses a token he can generate a new one. Old token can stay active if user does not believe the token has been compromised
3. User configures device at factory to use the produced token as its pantahub registration credential. Pantavisor will then use the token when registering itself for first time. It uses `Pantahub-Devices-Auto-Token-V1` to pass the token to pantahub when registering itself. With this pantahub will auto assign the device to the owner of the given token and will put UserMeta in place.

Example:

```
http --print=bBhH POST localhost:12365/devices/tokens Authorization:" Bearer $TOK" default-user-meta:='{"mykey": "yourvalue"}'
POST /devices/tokens HTTP/1.1
Accept: application/json, */*
Accept-Encoding: gzip, deflate
Authorization: Bearer eyJxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx...
Connection: keep-alive
Content-Length: 45
Content-Type: application/json
Host: localhost:12365
User-Agent: HTTPie/0.9.9

{
    "default-user-meta": {
        "mykey": "yourvalue"
    }
}

HTTP/1.1 200 OK
Content-Length: 370
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Dec 2018 15:46:39 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000635

{
    "default-user-meta": {
        "mykey": "yourvalue"
    },
    "disabled": false,
    "id": "5c0e8a5fc094f62eafcc96b4",
    "nick": "informally_trusted_dingo",
    "owner": "prn:pantahub.com:auth:/user1",
    "prn": "prn:::devices-tokens:/5c0e8a5fc094f62eafcc96b4",
    "time-created": "2018-12-10T16:46:39.740026537+01:00",
    "time-modified": "2018-12-10T16:46:39.740026537+01:00",
    "token": "xxxxxxxxxxxx"
}
```

**Remember the* `token` you won't be able to retrieve it another time.

AUTO ASSIGN DEVICES ON REGISTRATION

Now on device side you have to send the Pantahub-Devices-Auto-Token-V1: http header when registering yourself. This will make pantahub to automatically associated you with the owner of the token. Example:

```
http --print=bBhH POST localhost:12365/devices/ Pantahub-Devices-Auto-Token-V1:Fn5xxxxxxxxxxxxxxxxxxxxxxxxxx
POST /devices/ HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 0
Host: localhost:12365
Pantahub-Devices-Auto-Token-V1: Fn5xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
User-Agent: HTTPie/0.9.9



HTTP/1.1 200 OK
Content-Length: 353
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Dec 2018 16:03:35 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000788

{
    "device-meta": {},
    "id": "5c0e8e57c094f62eafcc96b8",
    "nick": "privately_correct_quail",
    "owner": "prn:pantahub.com:auth:/user1",
    "prn": "prn:::devices:/5c0e8e57c094f62eafcc96b8",
    "public": false,
```

```
    "secret": "0jf4yyyyyyyyyyy",
    "time-created": "2018-12-10T17:03:35.489788421+01:00",
    "time-modified": "2018-12-10T17:03:35.489788421+01:00",
    "user-meta": {
        "mykey": "yourvalue"
    }
}
```

LIST TOKENS REGISTERED

If you want to see which tokens are already registered by your user, you can do so through a simple get at collection level:

```
http --print=bBhH GET localhost:12365/devices/tokens Authorization:" Bearer $TOK"
GET /devices/tokens HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Authorization: Bearer xxxxxxxxxxxxxxXxxxxxxxxxxxxxxxxxxxxxxxxxxXXXXXxxxxxXXXXXXXXXXXXXXxxxxXXXXxxxxxxxxQ
Connection: keep-alive
Host: localhost:12365
User-Agent: HTTPie/0.9.9



HTTP/1.1 200 OK
Content-Length: 1256
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Dec 2018 15:48:14 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000664

[
    {
        "default-user-meta": {
            "mykey": "yourvalue"
        },
        "disabled": false,
        "id": "5c0e8a5fc094f62eafcc96b4",
        "nick": "informally_trusted_dingo",
        "owner": "prn:pantahub.com:auth:/user1",
        "prn": "prn:::devices-tokens:/5c0e8a5fc094f62eafcc96b4",
        "time-created": "2018-12-10T16:46:39.74+01:00",
        "time-modified": "2018-12-10T16:46:39.74+01:00"
    },
    {
        "default-user-meta": {
            "mykey": "yourvalue"
        },
        "disabled": true,
        "id": "5c0e6b93c094f67d861e9444",
        "nick": "entirely_fit_badger",
        "owner": "prn:pantahub.com:auth:/user1",
        "prn": "prn:::devices-tokens:/5c0e6b93c094f67d861e9444",
        "time-created": "2018-12-10T14:35:15.984+01:00",
        "time-modified": "2018-12-10T14:35:15.984+01:00"
    }, <>
    {
        "default-user-meta": {
            "mykey": "yourvalue"
        },
        "disabled": true,
        "id": "5c0e5affc094f6050521ae9c",
        "nick": "correctly_literate_camel",
        "owner": "prn:pantahub.com:auth:/user1",
        "prn": "prn:::devices-tokens:/5c0e5affc094f6050521ae9c",
        "time-created": "2018-12-10T13:24:31.615+01:00",
        "time-modified": "2018-12-10T13:24:31.615+01:00"
    },
    {
        "default-user-meta": {
            "mykey": "yourvalue"
        },
        "disabled": true,
        "id": "5c0e5ad6c094f603c3d11f7c",
        "nick": "miserably_singular_insect",
        "owner": "prn:pantahub.com:auth:/user1",
        "prn": "prn:::devices-tokens:/5c0e5ad6c094f603c3d11f7c",
        "time-created": "2018-12-10T13:23:50.359+01:00",
        "time-modified": "2018-12-10T13:23:50.359+01:00"
    }
]
```

DISABLE TOKENS

In case you want to ensure no further devices will be auto assinged you can disable your token in pantahub. Existing devices will continue to be authenticated though.

```
http --print=bBhH  DELETE localhost:12365/devices/tokens/5c0e8a5fc094f62eafcc96b4 Authorization:" Bearer $TOK"
DELETE /devices/tokens/5c0e8a5fc094f62eafcc96b4 HTTP/1.1
Accept: */*
```

```
Accept-Encoding: gzip, deflate
Authorization: Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXx
Connection: keep-alive
Content-Length: 0
Host: localhost:12365
User-Agent: HTTPie/0.9.9



HTTP/1.1 200 OK
Content-Length: 15
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Dec 2018 15:52:19 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000664

{
    "status": "OK"
}
...
```

GET ANY USER'S PUBLIC DEVICES

You can get the list of any user's public devices by using the owner /owner-nick params

```
http --print=bBhH GET localhost:12365/devices/?owner-nick=asac Authorization:" Bearer $TOK"
GET /devices?owner-nick=asac HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Authorization: Bearer xxxxxxxxxxxxxxXxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxXXXXXxxxxxXXXXXXXXXXXXXXxxxxXXXXxxxxxxxxxQ
Connection: keep-alive
Host: localhost:12365
User-Agent: HTTPie/0.9.9



HTTP/1.1 200 OK
Content-Length: 1256
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Dec 2018 15:48:14 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000664

[
    {
        "id": "5e1875e2fb13950bc38d0ebd",
        "prn": "prn:::devices:/5e1875e2fb13950bc38d0ebd",
        "nick": "truly_wanted_krill",
        "owner": "prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd",
        "secret": "123",
        "time-created": "2020-01-10T13:02:26.232Z",
        "time-modified": "2020-03-10T11:52:16.405Z",
        "public": true,
        "user-meta": {},
        "device-meta": {},
        "garbage": false
    },
    {
        "id": "5e5e8a1afb1395136d8b2ae2",
        "prn": "prn:::devices:/5e5e8a1afb1395136d8b2ae2",
        "nick": "trivially_noted_flea",
        "owner": "prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd",
        "secret": "123",
        "time-created": "2020-03-03T16:47:22.86Z",
        "time-modified": "2020-03-03T17:01:11.332Z",
        "public": true,
        "user-meta": {},
        "device-meta": {},
        "garbage": false
    },
    {
        "id": "5e668963fb139512df214ec4",
        "prn": "prn:::devices:/5e668963fb139512df214ec4",
        "nick": "vigorously_master_werewolf",
        "owner": "prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd",
        "secret": "123",
        "time-created": "2020-03-09T18:22:27.585Z",
        "time-modified": "2020-03-09T18:22:51.234Z",
        "public": true,
        "user-meta": {},
        "device-meta": {},
        "garbage": false
    }
]
```

```
http --print=bBhH GET localhost:12365/devices/?owner=prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd Authorization:" Bearer $TOK"
GET /devices?owner-nick=prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Authorization: Bearer xxxxxxxxxxxxxxXxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxXXXXXxxxxxXXXXXXXXXXXXXXxxxxXXXXxxxxxxxxxQ
Connection: keep-alive
```

```
Host: localhost:12365
User-Agent: HTTPie/0.9.9



HTTP/1.1 200 OK
Content-Length: 1256
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Dec 2018 15:48:14 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.000664

[
    {
        "id": "5e1875e2fb13950bc38d0ebd",
        "prn": "prn:::devices:/5e1875e2fb13950bc38d0ebd",
        "nick": "truly_wanted_krill",
        "owner": "prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd",
        "secret": "123",
        "time-created": "2020-01-10T13:02:26.232Z",
        "time-modified": "2020-03-10T11:52:16.405Z",
        "public": true,
        "user-meta": {},
        "device-meta": {},
        "garbage": false
    },
    {
        "id": "5e5e8a1afb1395136d8b2ae2",
        "prn": "prn:::devices:/5e5e8a1afb1395136d8b2ae2",
        "nick": "trivially_noted_flea",
        "owner": "prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd",
        "secret": "123",
        "time-created": "2020-03-03T16:47:22.86Z",
        "time-modified": "2020-03-03T17:01:11.332Z",
        "public": true,
        "user-meta": {},
        "device-meta": {},
        "garbage": false
    },
    {
        "id": "5e668963fb139512df214ec4",
        "prn": "prn:::devices:/5e668963fb139512df214ec4",
        "nick": "vigorously_master_werewolf",
        "owner": "prn:pantahub.com:auth:/5e1875e2fb13950bc38d0ebd",
        "secret": "123",
        "time-created": "2020-03-09T18:22:27.585Z",
        "time-modified": "2020-03-09T18:22:51.234Z",
        "public": true,
        "user-meta": {},
        "device-meta": {},
        "garbage": false
    }
]
```

7.6.7 Trails

Trails allow to users to change device configurations asynchronously using a step wise approach.

Trails API hence has two views with different priliges on it:

1. Device accounts
2. User accounts

Creating a trail.

Trails can be created as users or device on first boot.

To create a trail log in with a device acccount (see auth api documentation).

For this REAME we assume that you saved a device token in the `DTOKEN` environemtn variable and a user token in the `UTOKEN` env.

To create a trail we use the POST method on the trails api top level element using the DTOKEN:

```
http POST localhost:12365/trails/ Authorization:"Bearer $DTOKEN" \
    kernel:='{ "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000001"}' \
    app:='{"object": "prn:pantahub.com:objects:/57b6fa88c094f67942000002"}'

HTTP/1.1 200 OK
Content-Length: 419
Content-Type: application/json; charset=utf-8
Date: Sat, 27 Aug 2016 22:04:31 GMT
X-Powered-By: go-json-rest

{
    "device": "prn:pantahub.com:auth:/device1",
    "factory-state": {
        "app": {
            "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000002"
        },
        "kernel": {
            "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000001"
        }
    },
    "id": "57c20e6fc094f6729b000001",
    "last-seen": "0001-01-01T00:00:00Z",
    "last-step": "2016-08-28T00:04:31.386929333+02:00",
    "last-walk": "2016-08-28T00:04:31.386929333+02:00",
    "owner": "prn:pantahub.com:auth:/user1",
    "tail": 0,
    "tip": 0
}
```

Getting Steps (User Account)

Taking a peek at the steps as a user will by default return all steps that are not completed.

As a device by defaults only those steps get returned that are in state NEW.

All other states can be queried with query parameter q=|ALL

Lets see as a user:

```
http GET localhost:12365/trails/57c20e6fc094f6729b000001/steps Authorization:"Bearer $UTOKEN"
HTTP/1.1 200 OK
Content-Length: 424
Content-Type: application/json; charset=utf-8
Date: Sat, 27 Aug 2016 22:05:07 GMT
X-Powered-By: go-json-rest

[
    {
        "commit-msg": "Factory State (rev 0)",
        "committer": "",
        "device": "prn:pantahub.com:auth:/device1",
        "id": "57c20e6fc094f6729b000001-0",
        "owner": "prn:pantahub.com:auth:/user1",
        "progress": {
            "log": "",
            "progress": 0,
```

```
            "status": "DONE",
            "status-msg": ""
        },
        "rev": 0,
        "state": {
            "app": {
                "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000002"
            },
            "kernel": {
                "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000001"
            }
        },
        "trail-id": "57c20e6fc094f6729b000001"
    }
]
```

Here we see the intiial factory step being added to the system.

Adding steps

Steps need to be added with appropriate incremental rev to ensure that no concurrently added steps can cause incontinuity in the sequence of steps.

Lets simulate an app update as our rev 1:

```
http POST  localhost:12365/trails/57c20e6fc094f6729b000001/steps Authorization:"Bearer $UTOKEN" \
    rev:=1 \
    commit-msg="Update App to new Release" \
    state:='{
              "kernel": {"object": "prn:pantahub.com:objects:/57b6fa88c094f67942000001"},
              "app":    {"object": "prn:pantahub.com:objects:/57b6fa88c094f67942000003"}
            }'

HTTP/1.1 200 OK
Content-Length: 411
Content-Type: application/json; charset=utf-8
Date: Sat, 27 Aug 2016 22:22:14 GMT
X-Powered-By: go-json-rest

{
    "commit-msg": "Update App to new Release",
    "committer": "",
    "device": "prn:pantahub.com:auth:/device1",
    "id": "57c20e6fc094f6729b000001-1",
    "owner": "prn:pantahub.com:auth:/user1",
    "progress": {
        "log": "",
        "progress": 0,
        "status": "NEW",
        "status-msg": ""
    },
    "rev": 5,
    "state": {
        "app": {
            "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000003"
        },
        "kernel": {
            "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000001"
        }
    },
    "trail-id": "57c20e6fc094f6729b000001"
}
```

Accessing Individual Steps

To access individual steps relative to the trail you use "rev" in the path:

```
http GET localhost:12365/trails/57c20e6fc094f6729b000001/steps/1 Authorization:"Bearer $UTOKEN"
HTTP/1.1 200 OK
Content-Length: 422
Content-Type: application/json; charset=utf-8
Date: Sat, 27 Aug 2016 22:45:59 GMT
X-Powered-By: go-json-rest

{
    "commit-msg": "Update App to new Release",
    "committer": "",
    "device": "prn:pantahub.com:auth:/device1",
    "id": "57c20e6fc094f6729b000001-0",
    "owner": "prn:pantahub.com:auth:/user1",
    "progress": {
        "log": "",
        "progress": 0,
        "status": "NEW",
```

```
        "status-msg": ""
    },
    "rev": 0,
    "state": {
        "app": {
            "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000003"
        },
        "kernel": {
            "object": "prn:pantahub.com:objects:/57b6fa88c094f67942000001"
        }
    },
    "trail-id": "57c20e6fc094f6729b000001"
}
```

Device Progress Postings

To post progress, devices will PUT to the pseudo "progress" node under each step. In this case our device wants to confirms that it has seen the newly requested step and that it is acting on it.

```
http PUT localhost:12365/trails/57c20e6fc094f6729b000001/steps/progress Authorization:"Bearer $DTOKEN" \
    log = "" \
    progress = 0 \
    status = "QUEUE" \
    status-msg  = ""

HTTP/1.1 200 OK
Content-Length: 57
Content-Type: application/json; charset=utf-8
Date: Sat, 27 Aug 2016 22:50:29 GMT
X-Powered-By: go-json-rest

{
    "log": "",
    "progress": 0,
    "status": "QUEUE",
    "status-msg": ""
}
```

Steps Meta info

Steps have a general purpose 'meta' field holding a map. This is useful for apps to store info and state about steps that they need for their internal processing.

As owner you can PUT meta on any step:

```
http PUT localhost:12365/trails/5c2cc99990cd51000906c218/steps/10/meta Authorization:" Bearer $TOK" app-meta1=value1 app-meta2:='{"object": "also possible"}'
HTTP/1.1 200 OK
Content-Length: 61
Content-Type: application/json; charset=utf-8
Date: Wed, 17 Apr 2019 09:09:51 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.003029

{
    "app-meta1": "value1",
    "app-meta2": {
        "object": "also possible"
    }
}
```

As owner you can GET meta on any step:

```
http localhost:12365/trails/5c2cc99990cd51000906c218/steps/10/meta Authorization:" Bearer $TOK"
HTTP/1.1 200 OK
Content-Length: 61
Content-Type: application/json; charset=utf-8
Date: Wed, 17 Apr 2019 09:10:25 GMT
X-Powered-By: go-json-rest
X-Runtime: 0.002949

{
    "app-meta1": "value1",
    "app-meta2": {
        "object": "also possible"
    }
}
```

7.6.8 Objects

PANTAHUB Objects API for CDN ready storage of blob objects (aka files).

Start Service

DEVELOPMENT

If you do not set PANTAHUB_S3PATH ennvironment to production we will use local fake s3. If no path is set it will use the ./local-s3 directory as file storage on local disk

PRODUCTION

In order to enable production S3 usage you have to set the PANTAHUB_S3PATH environment like:

```
export PANTAHUB_S3PATH=production
```

Before starting, set your AWS credentials in your environment:

```
AWS_ACCESS_KEY_ID=XXXX
AWS_SECRET_ACCESS_KEY=YYYYYYYY

export AWS_SECRET_ACCESS_KEY AWS_ACCESS_KEY_ID
```

Now start your server:

```
./pantahub-base
```

Login

```
TOKEN=`http localhost:12365/auth/login username=user1 password=user1 | json token`
```

... will store access token in TOKEN for requests below

Upload File

REGISTER OBJECT

```
# adjust the below to be correct:
upload_file=myfile.jpg
upload_size=12365
upload_shasum256=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

http POST localhost:12365/objects/  Authorization:"Bearer $TOKEN" \
    objectname=$upload_file \
    size=$upload_size \
    sha256sum=$upload_shasum256

HTTP/1.1 200 OK
Content-Length: 152
Content-Type: application/json; charset=utf-8
Date: Fri, 19 Aug 2016 12:24:58 GMT
X-Powered-By: go-json-rest

{
    "id": "57b6fa9ac094f67942000002",
    "mime-type": "",
    "objectname": "myfile.jpg",
    "owner": "prn:pantahub.com:auth:/user1",
    "sha256sum": "xxxxxxxxxxx",
    "size": "12356"
}
```

GET PUT URL

Use the id from above to get info about where to upload:

```
http GET localhost:12365/objects/57b6fa9ac094f67942000002  Authorization:"Bearer $TOKEN"

HTTP/1.1 200 OK
Content-Length: 941
```

```
Content-Type: application/json; charset=utf-8
Date: Fri, 19 Aug 2016 12:26:41 GMT
X-Powered-By: go-json-rest

{
    "expire-time": "900",
    "id": "57b6fa9ac094f67942000002",
    "mime-type": "",
    "now": "1471609601",
    "objectname": "myfile.jpg",
    "owner": "prn:pantahub.com:auth:/user1",
    "sha256sum": "xxxxxxxxxxx",
    "signed-geturl": "https://systemcloud-001.s3.amazonaws.com/57b6fa9ac094f67942000002?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=AKIAJCANUJOIDFTXDLJA%2F20160819%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20160819T122641Z&X-Amz-Expires=900&X-Amz-SignedHeaders=host&X-Amz-
Signature=19801ba6781f9b10d7d108cc429e55942497b9bc5b46aafba7325709a82c0029",
    "signed-puturl": "https://systemcloud-001.s3.amazonaws.com/57b6fa9ac094f67942000002?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=AKIAJCANUJOIDFTXDLJA%2F20160819%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20160819T122641Z&X-Amz-Expires=900&X-Amz-SignedHeaders=host&X-Amz-
Signature=64ade69ed129f2aedcee9b84cd2e318b4863e2b6518301fbae9e53703c794e73",
    "size": "12356"
}

SIGNED_GETURL="https://systemcloud-001.s3.amazonaws.com/57b6fa9ac094f67942000002?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=AKIAJCANUJOIDFTXDLJA%2F20160819%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20160819T122641Z&X-Amz-Expires=900&X-Amz-SignedHeaders=host&X-Amz-
Signature=19801ba6781f9b10d7d108cc429e55942497b9bc5b46aafba7325709a82c0029"
SIGNED_PUTURL="https://systemcloud-001.s3.amazonaws.com/57b6fa9ac094f67942000002?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=AKIAJCANUJOIDFTXDLJA%2F20160819%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20160819T122641Z&X-Amz-Expires=900&X-Amz-SignedHeaders=host&X-Amz-
Signature=64ade69ed129f2aedcee9b84cd2e318b4863e2b6518301fbae9e53703c794e73"
```

PUT FILE TO S3

```
cat $upload_file | http PUT $SIGNED_PUTURL
```

GET FILE FROM S3

```
http GET $SIGNED_GETURL
```

7.6.9 Logs

PANTAHUB Logs Service

Start Service

Logs is part of pantahub-base. Start your base server:

```
./pantahub-base
```

Login

Login as a user:

```
TOKEN=`http localhost:12365/auth/login username=user1 password=user1 | json token`
```

... will store access token in TOKEN for USER requests below

Login as a device:

```
DTOKEN=`http localhost:12365/auth/login username=device1 password=device1| json token`
```

... will store access token in TOKEN for DEVICE requests below

Post Log Entries (DEVICE)

Devices post log entries by POSTING elements to the logs endpoint.

Mandatory fields are: * src - what log is this from? * msg - what is the message?

Recommended fields are: * tsec - time in seconds since 1970 * tnano - nanoseconds in seconds * lvl - severity/log/debug level

Implicit fields are: * dev - device id; will be extracted from login context for Devices * time-created - time when this entry first became known to the logs endpoint

You can post either a single entry or a json error for batch submission:

WRITE SINGLE ENTRY (DEVICE)

Option 1 (post single entry)

```
http POST localhost:12365/logs/ Authorization:" Bearer $DTOKEN" \
                                src="pantavisor.log" \
                                msg="My log line to remember" \
                               lvl="INFO" \
                               tsec="1496532292" \
                               tnano="802110514"
```

WRITE BATCH OF ENTRIES (DEVICE)

Option 2 (post a batch)

```
http POST localhost:12365/logs/ Authorization:" Bearer $DTOKEN" <<EOF
[
 { "src": "pantavisor.log",
   "msg": "message 1 text",
   "lvl": "INFO",
   "tsec": 1496532292,
   "tnano": 802110514
 },
 { "src": "pantavisor.log",
   "msg": "message 2 text",
   "lvl": "INFO",
   "tsec": 1496532322,
   "tnano": 802110545
 }
]
EOF
```

Browse the Logs (USER)

As user you can navigate through your logs using the `GET /logs/` endpoint.

Various parameters are available to restrict and sort your search.

PAGING

You can page using the start= and page= parameters:

- `start` - start offset
- `page` - page size; maximum entries to return in one call

LIMIT SEARCH BY TIME

You can limit search using the "after=" and "before=" query parameter to the `/logs` endpoint.

- `after` - RFC3399 formatted time to limit search to log entries with `time-created` larger than this time.
- `before` - RFC3399 formatted time to limit search to log entries with `time-created` smaller than this time.

At this point behavioru if both parameters are found in query is undefined.

STREAMING

To realize streaming typically you would query for logs and then use the date of last item retrieved as after= parameter until you retrieve a new item.

Also see below for Cursor feature which gives you a good way to step through long lists sorted by keys that are not unique.

CURSORS

If you want to scroll through later lists you can use cursor feature. With cursor you can use the `/logs/cursor` to page throught the search results whose initial invocation of the `/logs` endpoint got passed `cursor=true` a query parameter.

Until the cursor gets exhausted the result page of `/logs` and `/logs/cursor` endpoints will return a `next-cursor` field that contains the cursor you will have to pass to `/logs/cursor` to retrieve the next page.

Note that cursors do get invalidated if they get exhausted (meaning: you retrieved the last entries). In case the cursor is found to be exhausted by `/logs` or `/logs/cursor` endpoint, `next-cursor` will be empty string ("").

EXAMPLES

Example: Get log

```
http GET localhost:12365/logs/ Authorization:" Bearer $TOKEN" \
            start=0 \
            page=50

HTTP/1.1 200 OK
Content-Length: 1999
Content-Type: application/json; charset=utf-8
Date: Sat, 03 Jun 2017 23:44:46 GMT
X-Powered-By: go-json-rest

{
    "count": 8,
    "entries": [
        {
            "dev": "prn:pantahub.com:auth:/device1",
            "id": "59309891632d7256597b03d2",
            "lvl": "INFO",
            "msg": "MyMessage 4 single",
            "own": "prn:pantahub.com:auth:/user1",
            "src": "pantavisor.log",
            "time-created": "2017-06-02T00:43:29.136+02:00",
            "tnano": 121312212,
            "tsec": 123213
        },

    [...]

        {
            "dev": "prn:pantahub.com:auth:/device1",
            "id": "5930943d632d724db7c123e4",
            "lvl": "INFO",
            "msg": "MyMessage",
            "own": "prn:pantahub.com:auth:/user1",
```

```
            "src": "pantavisor.log",
            "time-created": "2017-06-02T00:25:01.885+02:00",
            "tnano": 12312212,
            "tsec": 123113
        }
    ],
    "page": 50,
    "start": 0
}
```

EXAMPLE: LIMIT SEARCH WITH `after` :

```
http GET "localhost:12365/logs/?after=2017-06-02T00:25:01.885%2B02:00 "Authorization:" Bearer $TOKEN"

HTTP/1.1 200 OK
Content-Length: 1999
Content-Type: application/json; charset=utf-8
Date: Sat, 03 Jun 2017 23:44:46 GMT
X-Powered-By: go-json-rest

{
    "count": 1,
    "entries": [
        {
            "dev": "prn:pantahub.com:auth:/device1",
            "id": "59309891632d7256597b03d2",
            "lvl": "INFO",
            "msg": "MyMessage 4 single",
            "own": "prn:pantahub.com:auth:/user1",
            "src": "pantavisor.log",
            "time-created": "2017-06-02T00:43:29.136+02:00",
            "tnano": 121312212,
            "tsec": 123213
        }

    ],
    "page": 50,
    "start": 0
}
```

Example: sorting

You can sort the logs by time-created.

```
http GET 'localhost:12365/logs/?src=pantavisor.log&sort=-time-created' \
               Authorization:" Bearer $TOKEN"

{
    "count": 8,
    "entries": [
        {
            "dev": "prn:pantahub.com:auth:/device1",
            "id": "5930943d632d724db7c123e4",
            "lvl": "INFO",
            "msg": "MyMessage",
            "own": "prn:pantahub.com:auth:/user1",
            "src": "pantavisor.log",
            "time-created": "2017-06-02T00:25:01.885+02:00",
            "tnano": 12312212,
            "tsec": 123113
        },

    [...]

        {
            "dev": "prn:pantahub.com:auth:/device1",
            "id": "59309891632d7256597b03d2",
            "lvl": "INFO",
            "msg": "MyMessage 4 single",
            "own": "prn:pantahub.com:auth:/user1",
            "src": "pantavisor.log",
            "time-created": "2017-06-02T00:43:29.136+02:00",
            "tnano": 121312212,
            "tsec": 123213
        }
    ],
    "page": 50,
    "start": 0
}
```

All fields available for sorting are: * tsec,tnano,device,src,lvl,time-created

Example: logs with cursor

```
http GET localhost:12365/logs/?cursor=true Authorization:" Bearer $TOK"
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json; charset=utf-8
Date: Mon, 15 Oct 2018 09:10:50 GMT
Server: nginx/1.13.5
Strict-Transport-Security: max-age=15724800; includeSubDomains;
Transfer-Encoding: chunked
X-Powered-By: go-json-rest
X-Runtime: 0.719266

{
    "count": 50,
    "entries": [
        {
            "dev": "prn:::devices:/5b27dbfbadf5440009c5020b",
            "id": "5bbd2bb56629c6000954cb8c",
            "lvl": "DEBUG",
            "msg": "going to state = STATE_WAIT(617)",
            "own": "prn:::accounts:/59ef9e241e7e6b000d3d2bc7",
            "src": "controller",
            "time-created": "2018-10-09T22:29:09.430488437Z",
            "tnano": 62937,
            "tsec": 1539124149
        },

[...]

    ],
    "next-cursor": "eyJhbGciOiJIUzI1N.....XU5xwIrtI4M",
    "page": 50,
    "start": 0
}
```

And use the value of `next-cursor` for follow up calls to the cursor endpoint:

```
http  POST localhost:12365/logs/cursor next-cursor="$next" Authorization:" Bearer $TOK"
...
```

This will also include a fresh `next-cursor`, please use that for doing the next call.

7.6.10 WARNING (alpha API and code)

This is alpha API and code; do not use it from third party software that you are not willing to rewrite or throw away very soon and often...

You have been warned :).

7.6.11 Configure admin users in pantahub

By default we assign the demouser "admin" permissions to grant subscriptions for pantahub.

You can change that through adding a comma separated list of prns for the following env configurations. The following are the defaults we use:

```
    PANTAHUB_ADMINS="prn:pantahub.com:auth:/admin"
    PANTAHUB_SUBSCRIPTION_ADMINS=""
```

Remember that you have to set password explictely if you want to enable the admin demo user.

To do define the admin password as env also with:

```
    PANTAHUB_DEMOACCOUNTS_PASSWORD_admin=YOURPASSWORDHERE
```

7.6.12 Set Subscription (as Admin user)

First log in as admin user, e.g.

```
TOK=`http POST https://api.pantahub.com/auth/login username=admin password=YOURPASSWORDHERE | jq  -r .token`
```

Then you can create or update the subscription plan for any given prn using the following rest call:

```
http PUT https://api.pantahub.com/subscriptions/admin/subscription \
    subject=prn:::accounts:/XXXXXXXXXXXXXXXXXX \
    plan=prn::subscriptions:VIP \
    Authorization:" Bearer $TOK"
```

Right now the following plans are available to choose from:

```
    prn::subscriptions:FREE
    prn::subscriptions:VIP
    prn::subscriptions:CUSTOM
```

You can overwrite default properties of plan through a json map that you can pass in as 'attrs' argument:

```
http PUT https://api.pantahub.com/subscriptions/admin/subscription \
    subject=prn:::accounts:/XXXXXXXXXXXXXXXXXX \
    plan=prn::subscriptions:VIP \
    attrs:='{"BANDWIDTH": "100GiB"}' \
    Authorization:" Bearer $TOK"
```

The effects should be visible for users right away in the UI and on dash endpoint.

7.6.13 See your subscription details (as user)

First login as normal user, e.g. with "user1" demoaccount:

```
TOK1=`http POST https://api.pantahub.com/auth/login username=user1 password=YOURUSER1PASSWORT | jq -r .token`
```

Next you can get your subscription status through simple GET against the subscriptions main endpoint:

```
http GET https://api2.pantahub.com/subscriptions/ Authorization:" Bearer $TOK1"
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json; charset=utf-8
Date: Mon, 05 Mar 2018 20:54:40 GMT
Server: nginx/1.13.5
```

```
Strict-Transport-Security: max-age=15724800; includeSubDomains;
Transfer-Encoding: chunked
X-Powered-By: go-json-rest

{
    "Page": -1,
    "Size": 1,
    "Start": 0,
    "Subs": [
        {
            "attr": {
                "BANDWIDTH": "100GiB",
                "DEVICES": "100",
                "OBJECTS": "20GiB"
            },
            "history": [
                {
                    "attr": {
                        "BANDWIDTH": "100GiB",
                        "DEVICES": "100",
                        "OBJECTS": "20GiB"
                    },
                    "id": "5a9dab7a9764eb000731cc60",
                    "issuer": "prn:pantahub.com:auth:/admin",
                    "last-modified": "2018-03-05T20:42:09.2Z",
                    "prn": "prn::subscriptions:/5a9dab7a9764eb000731cc60",
                    "service": "prn::subscriptions:",
                    "subject": "prn:::accounts:/59ef9e241e7e6b000d3d2bc7",
                    "time-created": "2018-03-05T20:41:30.58Z",
                    "type": "prn::subscriptions:VIP"
                }
            ],
            "id": "5a9dab7a9764eb000731cc60",
            "issuer": "prn:pantahub.com:auth:/admin",
            "last-modified": "2018-03-05T20:42:22.047Z",
            "prn": "prn::subscriptions:/5a9dab7a9764eb000731cc60",
            "service": "prn::subscriptions:",
            "subject": "prn:::accounts:/59ef9e241e7e6b000d3d2bc7",
            "time-created": "2018-03-05T20:41:30.58Z",
            "type": "prn::subscriptions:VIP"
        }
    ]
}
```

7.6.14 See your quotas on the dash endpoint

Simply log in as normal user like in section above and then query dash api:

```
$ http GET https://api2.pantahub.com/dash/ Authorization:" Bearer $TOK1"
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json; charset=utf-8
Date: Mon, 05 Mar 2018 20:55:53 GMT
Server: nginx/1.13.5
Strict-Transport-Security: max-age=15724800; includeSubDomains;
Transfer-Encoding: chunked
X-Powered-By: go-json-rest

{
    "nick": "user1",
    "prn": "prn:::accounts:/user1",
    "subscription": {
        "billing": {
            "AmountDue": 0,
            "Currency": "USD",
            "Type": "Monthly",
            "VatRegion": "World"
        },
        "plan-id": "VIP",
        "quota-stats": {
            "BANDWIDTH": {
                "Actual": 0,
                "Max": 100,
                "Name": "BANDWIDTH",
                "Unit": "GiB"
            },
            "DEVICES": {
                "Actual": 1,
                "Max": 100,
                "Name": "DEVICES",
                "Unit": "Piece"
            },
            "OBJECTS": {
                "Actual": 0.06,
                "Max": 20,
                "Name": "OBJECTS",
                "Unit": "GiB"
            }
        }
```

```
    },
    "top-devices": [
    ]
}
```

Have fun!

7.6.15 Device

PANTAHUB Dash

Start Service

Start your server:

```
./pantahub-serv
```

Login

```
TOKEN=`http localhost:12365/auth/login username=user1 password=user1 | json token`
```

... will store access token in TOKEN for requests below

Get Dash Content

```
http localhost:12365/dash/ Authorization:" Bearer $TOKEN"
HTTP/1.1 200 OK
Content-Length: 1239
Content-Type: application/json; charset=utf-8
Date: Wed, 12 Jul 2017 15:35:08 GMT
X-Powered-By: go-json-rest

{
    "nick": "user1",
    "prn": "prn:pantahub.com:auth:/user1",
    "subscription": {
        "billing": {
            "AmountDue": 0,
            "Currency": "USD",
            "Type": "Monthly",
            "VatRegion": "World"
        },
        "plan-id": "AlphaTester",
        "quota-stats": {
            "BANDWIDTH": {
                "Actual": 0,
                "Max": 5,
                "Name": "BANDWIDTH",
                "Unit": "GiB"
            },
            "BILLINGPERIOD": {
                "Actual": 0,
                "Max": 30,
                "Name": "BILLINGPERIOD",
                "Unit": "Days"
            },
            "DEVICES": {
                "Actual": 12,
                "Max": 25,
                "Name": "DEVICES",
                "Unit": "Piece"
            },
            "OBJECTS": {
                "Actual": 0,
                "Max": 5,
                "Name": "OBJECTS",
                "Unit": "GiB"
            }
        }
    },
    "top-devices": [
        {
            "message": "Device changed at 2017-06-30 00:19:26.79 +0200 CEST",
            "nick": "honest_collie",
            "prn": "prn:::devices:/5947ca58c4a28b000e8204f4",
            "type": "INFO"
        },
        {
            "message": "Device changed at 2017-06-21 13:42:39.484 +0200 CEST",
            "nick": "polished_stallion",
            "prn": "prn:::devices:/5947c794c4a28b000e82048b",
            "type": "INFO"
        },
        {
            "message": "Device changed at 2017-06-19 19:36:56.974 +0200 CEST",
            "nick": "bursting_manatee",
            "prn": "prn:::devices:/593821ddeb775d2154a4d7c2",
```

```
            "type": "INFO"
        },
        {
            "message": "Device changed at 2017-06-19 19:36:41.096 +0200 CEST",
            "nick": "refined_walleye",
            "prn": "prn:::devices:/593d7f67eb775d2154a4f2e3",
            "type": "INFO"
        },
        {
            "message": "Device changed at 2017-06-19 19:36:19.71 +0200 CEST",
            "nick": "nice_pangolin",
            "prn": "prn:::devices:/591b6cc27a6e8e197041e83a",
            "type": "INFO"
        }
    ]
}
```

7.6.16 Healthz API

This is a simple rest endpoint API directed towards devops that need a way to check for readiness and liveliness of the system during operation to manage the service lifecycle smartly.

This API endpoint is NOT meant to be used by devices or users.

Only principal that has access to this endpoint is the "saadmin" endpoint which must be authenticated through Basic Auth with the password configured by devops using the PANTAHUB_SA_ADMIN_SECRET.

If that secret is not set, this endpoint cannot be used.

Example 1

We start pantahub with PANTAHUB_SA_ADMIN_SECRET set to 'test1234':

```
$ PANTAHUB_SA_ADMIN_SECRET=test1234 pantahub-base
...
```

And then use the following json/rest call to get info about healthz:

```
http --auth saadmin:123123 localhost:12365/healthz/
HTTP/1.1 200 OK
Content-Length: 80
Content-Type: application/json; charset=utf-8
Date: Thu, 29 Jun 2017 21:50:28 GMT
X-Powered-By: go-json-rest

{
    "code": 0,
    "duration": 2706206,
    "start=-time": "2017-06-29T23:50:28.09254266+02:00"
}
```

Any status code greater or equal than 400 should be interpreted as failed.

8. Other Tutorials & Showcases

8.1 LimeSDR Mini USD on RPi

This tutorial requires a LimeSDR mini USB on a Pantavisor-enabled device. You can easily set up Pantavisor on Raspberry Pi 3 B+ with a pre-compiled image. Download the initial device image and then follow the rest of the instructions from here.

With the device set up and connected to the network, continue on with the next section.

8.1.1 Cloning and resetting the device

Claim the Pantavisor-enabled device, before connecting the LimeSDR micro USB to the board. Then log into Pantacor Hub to view the device on the dashboard.

Clone the device URL to your computer. Cloning creates a device representation on your host computer that allows you to modify, commit, and post a new revision to Pantacor Hub. After you've posted the new revision to the hub, the device downloads it and resets.

8.1.2 Adding Pantavisor apps to test your LimeSDR device

The table below displays a few Pantavisor applications that you can use to test your LimeSDR. Add these to the device with the app add command.

Since these apps take control of the LimeSDR micro, you can only run one at a time on your board. To delete a Pantavisor app and make room for another one, remove the app folder from the device after you've checked it out of the repository. When complete, don't forget to `add`, `commit` and `post` your changes to the device repository. See pvr commands for more information

Note that these apps are developed for the ARM architecture:

Name	Source	pvr Command
limeScan	link	pvr app add --from registry.gitlab.com/myriadrf/pantahub/limescan-device:ARM32V6 limescan
osmocom	link	pvr app add --from registry.gitlab.com/myriadrf/pantahub/osmo-gsm-net:ARMHF-master osmocom
limeSNA	link	pvr app add --from registry.gitlab.com/myriadrf/pantahub/limesna:ARM32V6-docker limesna
SDRangel	link	pvr app add --from registry.gitlab.com/myriadrf/pantahub/sdrangel:ARM32V7-ubuntu sdrangel

8.2 Mozilla IoT Gateway as an App on Pantavisor Devices

8.2.1 Mozilla Iot Gateway as a Pantavisor App

Mozilla IoT Gateway allows you to manage WebThings directly on your device.

To get started, download a Pantavisor enabled base image for your preferred device and add Mozilla WebThings Gateway to it.

First clone your device from Pantacor Hub:

```
pvr clone YOURNICK/YOURDEVICE
cd YOURDEVICE
```

Then add the WebThings gateway from docker:

```
pvr app add \
    --volume=lxc-overlay:revision \
    --volume=/home/node/.mozilla-iot:permanent \
    --from=mozillaiot/gateway:arm gateway
```

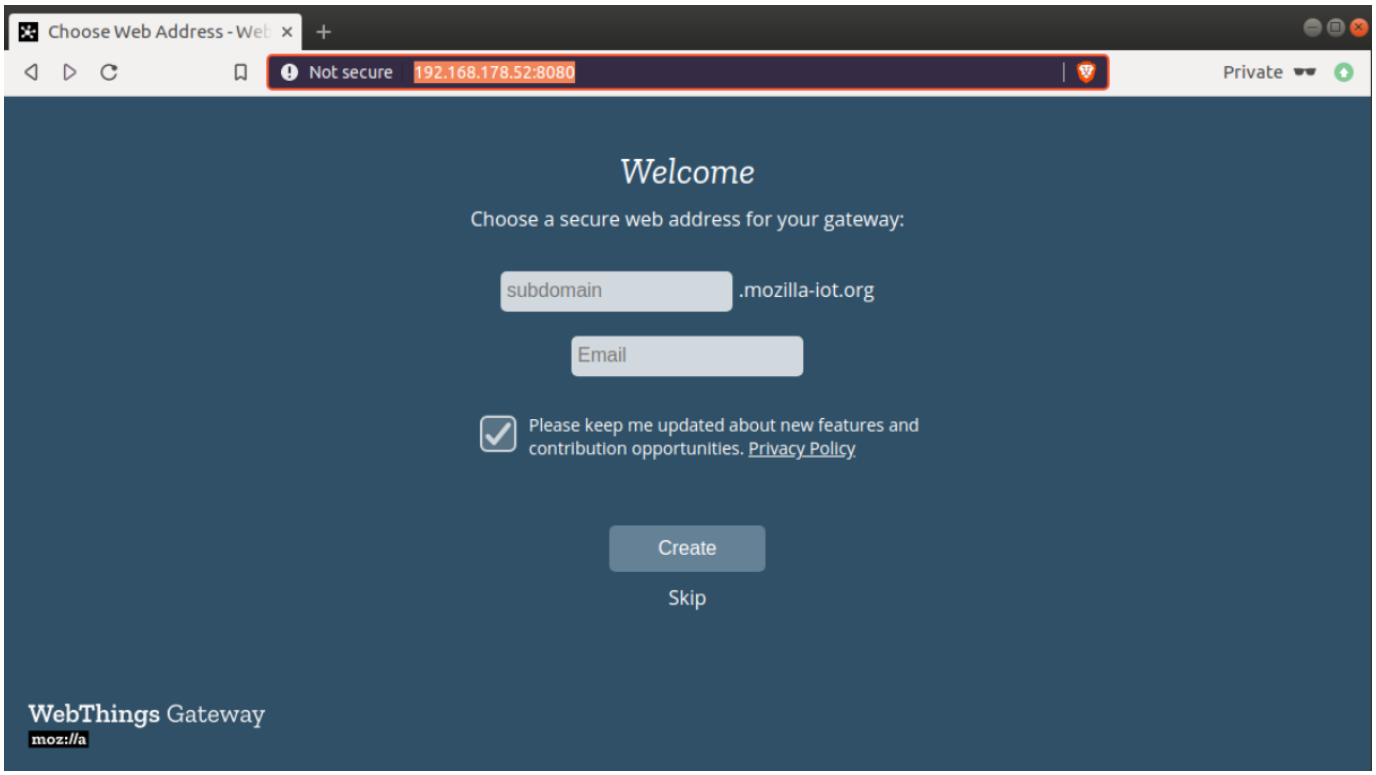
Note: We make the mozilla-iot datafolder volume go to permanent storage that will persist across reboots and revisions. The general overlay used to make the whole container filesystem read write however we only persist for revisions, that means on upgrade you won't have to worry about potential dirt that got left behind.

Now commit and post it to your device:

```
pvr add .
pvr commit
pvr post -m "add mozilla iot gateway" YOURNICK/YOURDEVICE
```

This will trigger a reboot of your device and afterwards you will be able to use WebThings gateway at the IP address of your gateway.

8.2.2 Screenshots



8.3 Yocto Digital Signage on OpenWrt Router

As shown at ELC 2019 in San Diego, using Linux container technology to mix and match best user space stacks that come with otherwise mutually hard to combine userspace isn't all that difficult.

In this case we will show how to make a router powered by OpenWRT also deliver an on-screen device experience.

This type of combined router+onscreen experience is becoming more relevant as digital signage powered by standard computers as well as the need for more pervasive networking becomes more and more ubiquous.

8.3.1 The Hardware

In order to make a product that combines both router as well as on screen signage in one device one has to pick an ARM SoC based router that comes with the GPU integrated - such as Mali.

Interestingly enough there are not many routers our there that also have graphics output interface, but we believe this will change in the future.

For this tutorial we picked the awesome BananaPi R2 (BPI-R2) platform.

8.3.2 The software

Putting together a software stack that can provide both, a product grade router vertical as well as a product grade on screen experience feels in first sight more trivial than finding hardware, but when looking closer it turns out there is no single distribution that is good at both.

If one wants to still setttle on using a single distribution for the whole experience one would have to either add a product grade wifi experience to a distro good on screens or add a great screen experience to a distro optimized for routers.

Both choices would mean quite substantial work and hence we believe there needs to be a third way:

1. We take an awesome router system stack and separately pick an awesome on screen display stack
2. We use containers to deploy them next to each other, so we can avoid big itegration pain

And thats what we did ...

8.3.3 The solution

To make this tutorial we cobbined the following components into one: 1. bananapi r2 hardware platform 2. Pantavisor container lifecycle management 4. Close-to-upstream Kernel for Bananpi R2 with openwrt patches applied and graphics modules enabled 5. OpenWRT 18.06 rootfs deployed as a container 6. Yocto Digital Signage based on yocto-warrior deployed as a container

8.3.4 Yocto on OpenWrt using prebuilt binaries

- Download our prebuilt BPI-R2 image (includes basic OpenWRT): https://pvzero.s3.amazonaws.com/bpi-r2-pv-2048MiB.img.xz
- Install, boot and claim your device on Pantacor Hub like described in our Getting Started guide
- add the yocto based digital signage stack:

```
pvr clone <yournick/yourdevice>
cd yourdevice
pvr app add \
    --arg "LXC_TTY_MAX=0" \
    --arg="LXC_MOUNT_AUTO_PROC=proc:rw" \
    --arg="LXC_MOUNT_AUTO_SYS=sys:rw" \
    --from=asac/signage signage
pvr add
pvr commit
```

- Post your changes back to your device: `pvr post -n "add signage demo"`

This will trigger the upgrade process and your Bananpi will reboot into the new experience - which would bring up a basic digital signage experience on your HDMI connected screen.

8.4 Installing App on Pantavisor Device for GPIO management via Pantacor Hub

In this tutorial, we are going to flash and claim a Pantavisor device to our Pantacor Hub account. Then, we are going to install a simple container in it with GPIO setting capabilities that can read cloud-controlled Pantacor Hub input metadata. All these elements will be combined to demonstrate how you can interact with your device peripherals (in this case, GPIO) from our Pantacor Hub user interface.

8.4.1 What you will need

To flash and claim your Pantavisor device on Pantacor Hub:

- Raspberry Pi 3 B+
- Power cable for RPi3
- Internet-facing Ethernet cable
- Your computer

Also, you will need this hardware to test the GPIOs (see figure):

- One or more LEDs
- 330 ohm resistor for each LED (anything over about 50 ohm would work)
- One female to male breadboard jumper wires per LED plus an additional one for GND
- Breadboard

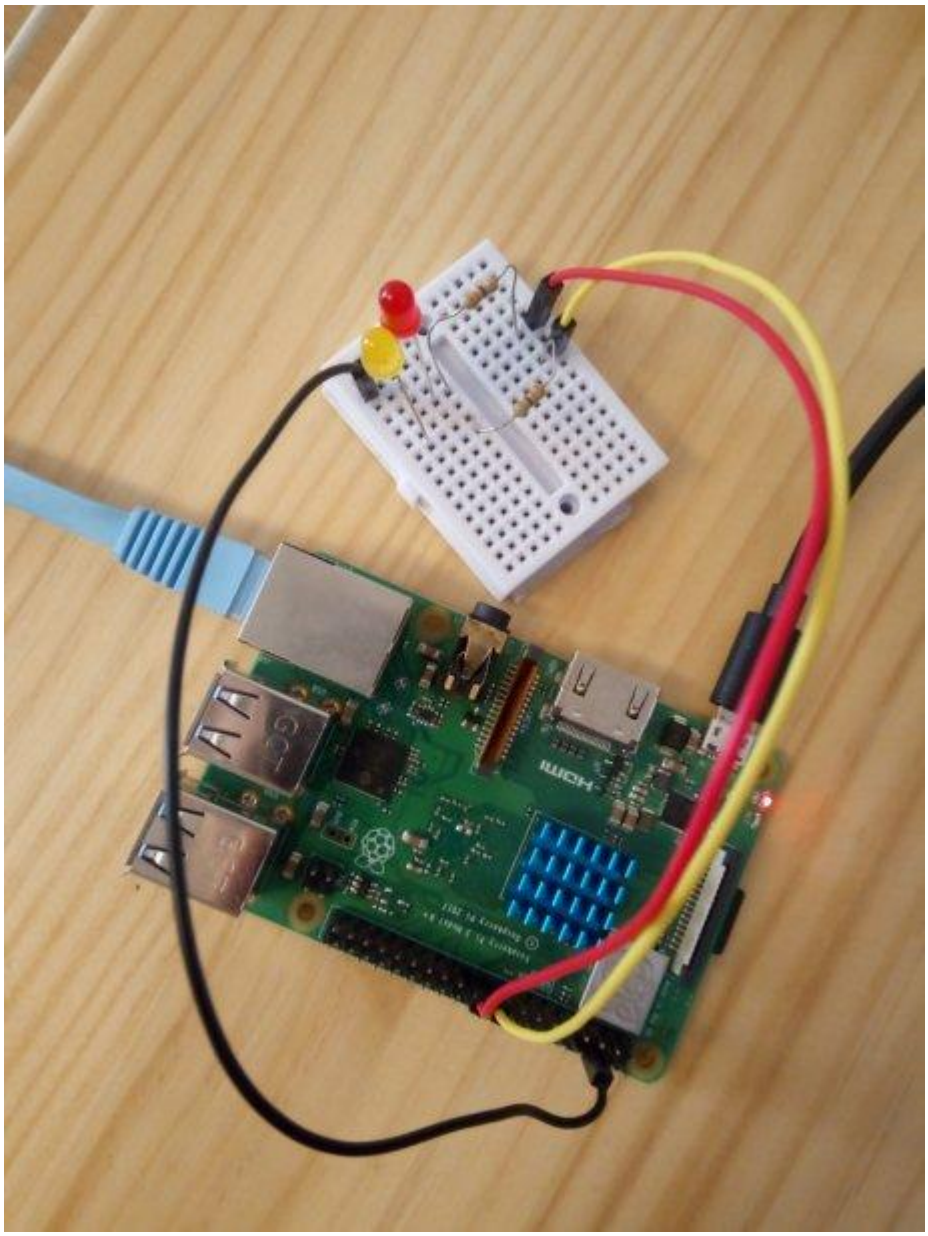
8.4.2 Instructions

1. Set up hardware
2. Flash and claim
3. Install GPIO App
4. Test it
5. Go further

Set up hardware

You have to firstly set up your LED circuit. In raspberrypi.org, you have access to the GPIO reference to identify where are the GPIO pins and GND on the board. Furthermore, you can see how to connect a LED to the board here.

Take a look at our set up in this image:



For this example, we are going to use GPIO 9 (red LED) and GPIO 10 (yellow LED).

Flash and claim

Now, you must go through our prerequisites and get started tutorials.

Once you have your RPi3 running with Pantavisor and you can access its cloud representation in Pantacor Hub, you can continue to the next step.

Install GPIO app

Go ahead and clone your device checkout to your host computer.

What we are going to do now, is to install an app that let you set and unset your device's GPIO. We have prepared a shell script inside a Docker container, which can be deployed to an ARMv6 device with Pantavisor (like your RPi3). We are going to do this using the `pvr` tool from your host computer.

First, *cd* to your cloned device checkout and execute the following commands:

```
cd ../<clone-checkout> # your cloned device repesentation
fakeroot pvr app add --from registry.gitlab.com/pantacor/pv-platforms/rpi-gpio:ARM32V6 gpio
pvr add .
pvr commit
pvr post
```

This will automatically commit the new app and post it to the device, which will then process the update. You can follow this process on Pantacor Hub, checking out the console of your device. Once the device is in *done* state, you will be ready to test it.

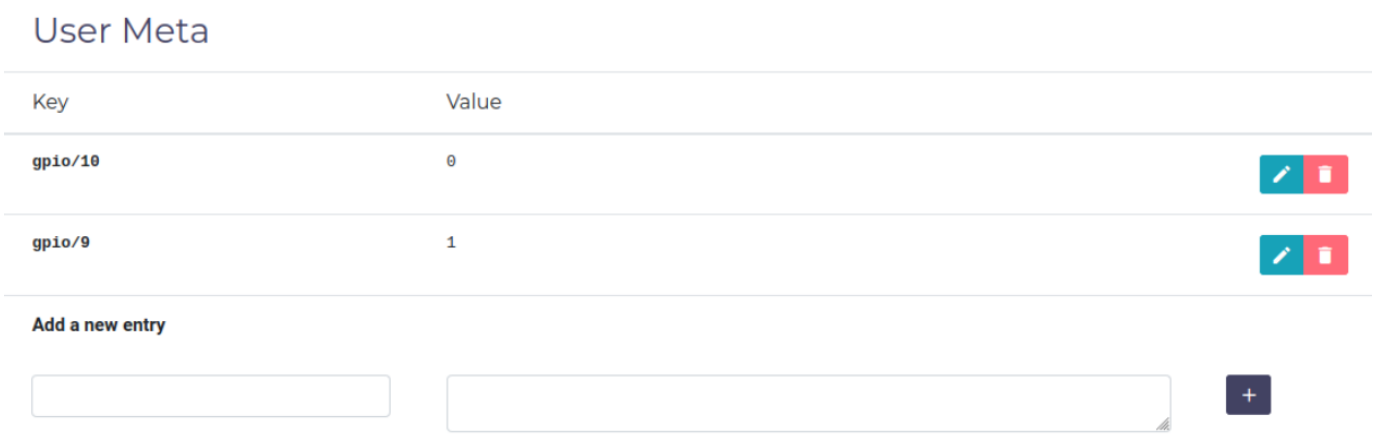
> **Note**
>
> Check the source code of the container you just installed here. Have in mind that we are natively running this container on Pantavisor, and that we only use Docker for the app definition.

Test it

In order to test it, you can go to your device metadata panel in Pantacor Hub. This is the pair used by the app we just installed.

```
key: gpio/<gpio-number>
value: 0 or 1
```

For our example, we are going to set GPIOs 9 and 10 to turn on the red and yellow LEDs:



Go further

You can make changes to this app by cloning the *rpi-gpio* repository and building its dockerfile locally. As the app is a pure Bash one, any x86 laptop can build and deploy the container using `pvr app add`. For example, to locally build and update the app, we could do:

```
cd ..
git clone https://gitlab.com/pantacor/pv-platforms/rpi-gpio.git
cd rpi-gpio
docker build -t demos:gpio -f Dockerfile.ARM32V6 .
cd ../<clone-checkout> # your cloned device
rm -r gpio
fakeroot pvr app add --from demos:gpio gpio
pvr add .
pvr commit
pvr post
```

You can iterate the above by making changes to the Docker container. I.e: change the gpio.sh script so the LED blinks when set in the device metadata. After editing the shell script:

```
cd ../rpi-gpio
docker build -t demos:gpio -f Dockerfile.ARM32V6 .
cd ../<clone-checkout>
fakeroot pvr app update gpio
pvr commit
pvr post
```

8.5 Installing simple Hello World App on Pantavisor Device

In this tutorial, we are going to install a simple hello world Application to our previously installed and claimed Pantavisor device.

8.5.1 What you will need

To install Application on your Pantavisor device on Pantacor Hub:

- Any x86_64 based target board (we are using Odroid-H2 in our example)
- Power cable for target board
- Internet-facing Ethernet cable
- Your computer

8.5.2 Instructions

COMPILE A HELLO WORLD C PROGRAM

What we are going to do now, is to compile a hello world app that will just print Hello Pantacor Hub on console continously in a forever loop with some delay. First, create a directory anywhere you wish, execute the following commands:

```
mkdir Hello_World_App
cd Hello_World_App
```

Now open a new file `hello_world.c` in your favourite text editor and copy and paste the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(void) {
    unsigned long long int i = 0;
    while(1) {
        printf("Hello Pantacor Hub %llu\n", i);
        i++;
        sleep(3);
    }
    exit(0);
}
```

Save the file and quit the text editor. Now compile the C program, make sure to link the libc library statically, execute the follwing command:
`$ gcc hello_world.c -o hello_world -static`
This will produce a binary hello_world, make sure the binary is statically linked libc libary, to check execute the following command, it should produce output:
`$  not a dynamic executable`

```
$ ldd hello_world
        not a dynamic executable
```

CREATE A DOCKERFILE TO BUILD A DOCKER IMAGE OF YOUR APPLICATION

A dockerfile is required to build a docker image, create a dockerfile
`$ vi Dockerfile`
Copy and paste the following content in your dockerfile:

```
FROM scratch
ADD hello_world /usr/bin/hello_world
ENTRYPOINT [ "/usr/bin/hello_world" ]
```

Save the file and quit the text editor. To build a docker image of your application, execute the following command:
`$ docker build -t helloworldbin -f Dockerfile ./`
`helloworldbin` is the name of the image, you can give any name of your choice for the image.
Verify that the image is available under the list of images, run:
`$ docker images`
OR

`$ docker image ls`

The above command will list all the available docker images on your host computer.

VERIFY THE DOCKER IMAGE

Verify that the docker image is working correctly, execute the following command to run the docker image:

`$ docker run --rm-t helloworldbin`

If the above command doesn't produce any error, it means the image is build succssfully.

The above command won't get terminated with `ctrl+c`, it needs to be stopped via docker, open a new terminal window or tab and execute the follwing command:

```
docker rm -f $(docker ps -a -q -f "ancestor=helloworldbin")
```

INSTALL THE HELLO WORLD APP

To install the Hello world app to your Pantacor Hub device we are going to use the pvr tool from your host computer, execute the following commands:

```
cd ../<clone-checkout> #your cloned device repesentation
fakeroot pvr app add --arg PV_RUN_TMPFS_DISABLE=yes --from helloworldbin hello_binary
pvr add .
pvr commit
pvr post
```

This will automatically commit the new app and post it to the device, which will then process the update. You can follow this process on Pantacor Hub, checking out the console of your device. Once the device is in done state, you can view the logs from console choose hello_binary-console from Filter Logs dropdown.

8.6 ADU Agent Container

Pantavisor turns the all the runtime into a set of containers. This means the updates you push from Azure Device Update UI are going to be simply that: a set of containers that can be added, updated or removed. This includes the Azure Device Update agent, which have been containerized to be run on a Pantavisor enabled device.

The container contents are defined in the `Dockerfile.template` and built for arm32, arm64 and x64 architectures from GitLab CI with the help of container-ci. To sum up, the components of the container are:

- Azure Device Update service with PVControl content handler
- Delivery Optimization service: dependecy of ADU agent used to download the updates
- pvcontrol: used by the ADU PVControl Handler for communication with Pantavisor

NOTE: the containers are not natively run by Docker but by Pantavisor. The dockerfile is only used to define the container root file system.

8.6.1 Get Started

1. Download latest flashable image:

Target	Image
BPi r64	https://pantavisor-ci.s3.amazonaws.com/adu-ci/tags/001/371237583/bpi_r64_azure_stable.img.xz
RPi64	https://pantavisor-ci.s3.amazonaws.com/adu-ci/tags/001/371237583/rpi64_azure_stable.img.xz
x64	https://pantavisor-ci.s3.amazonaws.com/adu-ci/tags/001/371237583/x64_azure_stable.img.xz

NOTE: RPi or BPi images are recommended, as x64 lacks OEM partition support. x64 will work too, but you will have to manually edit the configuration in the device during runtime.

1. Flash the downloaded image in an SD card:

Uncompress and flash your device:

```
unxz rpi64_azure_stable.img.xz
umount /dev/mmcblk0*
sudo dd if=rpi64_azure_stable.img of=/dev/mmcblk0 bs=32M
sync
```

1. Set your adu-conf.txt into the 2nd partition of the newly flashed SD card:

NOTE: Remember your device has to be previously registered in Azure IOT Hub.

Prepare your adu-conf.txt with this format:

```
connection_string=<your connection string here>
aduc_manufacturer=Fabrikam
aduc_model=Toaster
```

Flash adu-conf.txt:

```
sudo dd if=adu-conf.txt of=/dev/mmcblk0p2 bs=1M
sync
```

1. Insert your SD card in the device, turn it on and check its connected in Azure IOT Hub
2. Clone the current running revision in your host computer with pvr

First, you will need to know your device IP. If your host computer is in the same network, you can use pvr device scan command.

After that, clone the revision into your computer with:

```
pvr clone http://192.168.1.122:12368/cgi-bin/pvr my-checkout
```

1. Make changes in the checkout. For example, we are going to install a new NGINX container from DockerHub.

```
cd my-checkout
pvr app add --from nginx:stable-alpine webserver
pvr add .
pvr commit
```

1. Convert the pvr checkout into ADU format with the pvr2adu script

```
pvr2adu -p Pantacor -n Toaster -m Fabrikam -d Toaster -v 1.1 -o out
```

1. Deploy the resulting manifest from Device Update

8.6.2 TODO

- Step 3 reduced if we flash the configuration with a mechanism similar to the one we use in our image-service.
- Steps that require interaction with Azure UI (steps 3 and 4) could be automated using Azure API
- Pantabox could make the updation requests easier (steps 5, 6, 7 and 8) if we could deploy updates using Azure API

9. Downloads

9.1 Pantavisor Initial Devices

9.1.1 Update a Pantavisor Initial Device

To use these devices, you will need an already flashed and ready-to-use Pantavisor device. If you don't have one, go to the guide on how-to prepare your device.

If you want to update the BSP plus the development containers from any of the initial devices, you can upgrade your device checkout using pvr. Just remember to change the pvr clone URL with the one corresponding to your platform:

```
pvr merge https://pvr.pantahub.com/pantahub-ci/rpi64_5_10_y_bsp_latest
pvr checkout
```

Stable

> **Note**
>
> This devices contain both the stable and the release candidate versions. Navigate through the device *History* tab to switch to the version you want to use.

Platform	Target	Pantacor Hub devices
aarch64-appengine	aarch64-appengine	device
arm-appengine	arm-appengine	device
bpi-r2	arm-bpi-r2	device
bpi-r64	arm-bpi-r64	device
toradex-colibri-imx6_7	arm-toradex-colibri-imx6_7	device
malta	malta-qemu	device
mips-appengine	mips-appengine	device
nv-tegra-4_9	nv-tegra-4_9	device
odroid-c2	arm-odroid-c2	device
rock64	rock64	device
rpi0w-5.10.y	rpi0w-5.10.y	device
rpi64-5.10.y	rpi64-5.10.y	device
x64-appengine	x64-appengine	device
x64-ubuntu	x64-ubuntu	device
x64-uefi	x64-uefi	device

Latest

Platform	Target	Pantacor Hub devices
aarch64-appengine	aarch64-appengine	device
arm-appengine	arm-appengine	device
bpi-r2	arm-bpi-r2	device
bpi-r64	arm-bpi-r64	device
toradex-colibri-imx6_7	arm-toradex-colibri-imx6_7	device
malta	malta-qemu	device
mips-appengine	mips-appengine	device
nv-tegra-4_9	nv-tegra-4_9	device
odroid-c2	arm-odroid-c2	device
rock64	rock64	device
rpi0w-5.10.y	rpi0w-5.10.y	device
rpi64-5.10.y	rpi64-5.10.y	device
x64-appengine	x64-appengine	device
x64-ubuntu	x64-ubuntu	device
x64-uefi	x64-uefi	device

9.1.2 About Pantavisor Initial Devices

Our initial devices CI keeps a set of devices of different architectures up to date with both stable and daily BSP builds plus some containers that will help you start developing your own Pantavisor apps.

These are the containers included in the initial devices:

- **awconnect**: automatically brings up basic cabled networking. If this fails, it creates a hotspot with SSID "Wifi Connect" with a captive portal that allows you to manually configure your device for WiFi connection. See more in our reference.
- **pv-avahi**: uses DNS multicast for device discoverability.
- **pvr-sdk**: contains the Pantabox utilities to help with your development and debugging. Also, it incorporates an endpoint for pvr local experience. More in our reference pages.
- **container-ready-demo**: mock-up container that sends a ready signal to Pantavisor for testing and demostration purposes.

The CI also generates flashable images for the stable versions. Check out the jobs in the pipeline list and the release notes for each stable version in the list of tags.

9.2 Pantavisor Initial Images (Yocto)

We provide images generated by our CI using the Yocto project.

Here, you can download our flashable base Pantavisor image for your device. For instructions on how to install one of these Pantavisor images, see our how-to guide.

Note that the images in these lists are generic and will later require the device to be claimed if you want to interact with it from Pantacor Hub.

The Initial Images are referred to as `pantavisor-starter`, which includes the following components:

- **BSP**: Includes Pantavisor, kernel, device-trees, and u-boot files.
- **Containers**:
  - `pv-alpine-connman`: Networking management.
  - `pv-pvr-sdk`: Container with tooling to manage Pantavisor.
  - `pv-wificonnect`: WiFi captive portal to help configuring WiFi-enabled boards.

The initial images are stored at releases.json.

9.2.1 Latest Stable Images

Loading latest stable images...

9.2.2 Latest Release Candidate Images

Loading latest release candidate images...

9.3 This page is for archive purposes, for initial images check

9.4 Pantavisor Initial Images(legacy)

Here, you can download our flashable base Pantavisor image for your device. For instructions on how to install one of these Pantavisor images, see our how-to guide.

Note that the images in these lists are generic and will later require the device to be claimed if you want to interact with it from Pantacor Hub. If you prefer a customized image with a more streamlined remote setup, try our download page.

9.4.1 Stable Initial Images

This is our stable version images generated from the initial devices.

- Release: 019 - release notes
- Date: 2023-02-15 03:04:37 +0000
- BSP: 019 - release notes
- Pantavisor: 019 - release notes
- pvr version: 035-49-ge7fed375

Downloads:

Platform	Target	Flashable images	sha256sum
aarch64-appengine	aarch64-appengine (default)	appengine download	eb58f7431ff0ee6edaf17863d6034e918ec1d73147ed4a4704419bdd520acc82
arm-appengine	arm-appengine (default)	appengine download	ada8f383d43361e6e3d6d03ad8c25f8f7930fcbdbbe40b6362f9c00ff795ebbc
beaglev	beaglev (default)	img download	483d3a95139e4db8f045940e6362d0680b96744e541e87e36caed8f57165638f
bpi-r2	arm-bpi-r2 (default)	img download	1606f4a23334ac2f1905e809650c51ed8a532788dae036a34a62b5d8ac865efe
bpi-r64	arm-bpi-r64 (default)	img download	39d9542138fc2932e66a6e6bd4c003b739874904d2456ce45656ea143a132f04
bpi-r64	arm-bpi-r64 (default)	img download	ad9cf5a68870250697edea07f36ef6becfd44f5b2a5ace87845a3174b5339428
toradex-colibri-imx6_7	arm-toradex-colibri-imx6_7 (default)	tezi download	628dd7ed437d5e2aff8b46bf6f8493b71400be8dd7e91ebe9632ffc21c450c00
malta	malta-qemu (default)	img download	9f1d58881b4e20dd8246ac0a24c818eab3d6224a423ea052fda6e83dc72b7884
mips-appengine	mips-appengine (default)	appengine download	4818b063a679ba63071829912e68374bcc42a3ee6ae463f84a479d45fbf3dd77
nv-tegra-4_9	nv-tegra-4_9 (default)	rootfs download	71da7eb50e382362e245e6b259c63367ff429b7f1c8343ceb11b694fd30d6349
odroid-c2	arm-odroid-c2 (default)	img download	0c7593a84eef2cff605df1c9b4b4ad4f42d79b3bf8f01c6bcfa00ef5aae23c5a
rock64	rock64 (default)	img download	8969b40f6724421b9af4ad4bae7b134d799836f272849741bd24454c15afe974
rpi0w-5.10.y	rpi0w-5.10.y (default)	img download	a4549afbd14ec105fd12212c184842ebf13c9b8eb88c3545fe33b3c86aedd0bd
rpi64-5.10.y	rpi64-5.10.y (default)	img download	b8dee1d6b535ed577d55493df6c095ca9d561b9f461ede23e6d421caf28d47cd
x64-appengine	x64-appengine (default)	appengine download	74e1b792a5b38177ff6c8070308cbbb4feaca806be362859c364ba9b7aba4dd0
x64-uefi	x64-uefi (default)	img download	1fe5c993860d6b7221a28a9e6071277d0ee19d3315625c56b40ca675cada66c8
x64-uefi	x64-uefi (installer)	img download	286ead98a50f90ea1d4ff4d7812be19c07ad927ef47b5d5c263c3002796c6560
x64-uefi	x64-uefi (installer-sdo)	img download	2b762e3e2054aad6a8cd61bebbed6b8601cb33d4280c097ae8fe0bd6172c286b

9.4.2 Stable Initial Images List

This is a list of the all the tag pipelines generated from our initial devices.

Tag	Pipeline	Date	pvr	Flashable images
020-rc46	1989955075	2025-08-18 17:53:55 +0000	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc45	1985834402	2025-08-15 02:08:45 +0000	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc44	1984112049	2025-08-14 02:10:37 +0000	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc43	1707068219			

Tag	Pipeline	Date	pvr	Flashable images
		2025-03-09 03:12:51 +0000	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc42	1639573596	2025-01-24 11:46:19 +0000	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc41	1564605565	2024-11-28 03:08:47 +0000	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc40	1396540704	2024-08-01 10:01:43 +0200	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-

Tag	Pipeline	Date	pvr	Flashable images
				appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc39	1394222943	2024-07-30 02:08:59 +0000	037-66-g2b9cdeef	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | UNKNOWN-image x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc38	1237117735	2024-04-02 02:07:11 +0000	037-66-g2b9cdeef	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc37	1232562758	2024-03-28 10:24:26 +0000	037-66-g2b9cdeef	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-

Tag	Pipeline	Date	pvr	Flashable images
				appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc36	1231740764	2024-03-28 10:24:26 +0000	037-66-g2b9cdeef	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc33	1228425709	2024-03-26 16:58:28 +0000	037-66-g2b9cdeef	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | rpi-connman | rpi-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc32	1225089757	2024-03-23 19:10:03 +0000	037-66-g2b9cdeef	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | visionfive2-default | x64-appengine-

Tag	Pipeline	Date	pvr	Flashable images
				default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc31	1225077006	2024-03-23 18:36:31 +0000	037-66-g2b9cdeef	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc30	1224465047	2024-03-22 16:34:41 -0300	037-66-g2b9cdeef	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc29	1224407452	2024-03-22 18:29:12 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc28	1217390832	2024-03-18 09:47:08 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-

Tag	Pipeline	Date	pvr	Flashable images
				default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-connman | rpi64ab-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc27	1216848545	2024-03-17 20:49:07 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | rpi64ab-default | rpi64ab-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc26	1216699855	2024-03-17 14:57:41 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | UNKNOWN-image rpi64ab-default | UNKNOWN-image rpi64ab-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc25	1216297660	2024-03-16 21:56:46 +0000	037-63-ge0e79500	aarch64-appengine-default | arm-appengine-default | UNKNOWN-image arm-bpi-r2-default | UNKNOWN-image arm-bpi-r64-default | UNKNOWN-image arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | UNKNOWN-image arm-odroid-c2-default | UNKNOWN-image rock64-default | UNKNOWN-image rpi0w-5.10.y-default | UNKNOWN-image rpi64-5.10.y-

Tag	Pipeline	Date	pvr	Flashable images
				connman | UNKNOWN-image rpi64-5.10.y-default | UNKNOWN-image rpi64-5.10.y-pvwc_connman | UNKNOWN-image rpi64-5.10.y-pvwificonnect | UNKNOWN-image rpi64ab-default | UNKNOWN-image rpi64ab-default | UNKNOWN-image visionfive2-default | x64-appengine-default | UNKNOWN-image x64-uefi-default | UNKNOWN-image x64-uefi-installer | UNKNOWN-image x64-uefi-installer-sdo | UNKNOWN-image x64-ubuntu-default
020-rc24	1202228145	2024-03-05 03:05:08 +0000	037-63-ge0e79500	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc23	1182575826	2024-02-17 03:05:17 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwc_connman | rpi64-5.10.y-pvwificonnect | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc22	1150004151	2024-01-24 12:31:51 -0300	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | rpi64-5.10.y-pvwificonnect | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-

Tag	Pipeline	Date	pvr	Flashable images installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc21	1124609844	2024-01-02 16:03:05 +0100	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-connman | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc19	1124333714	2024-01-02 11:10:53 +0100	037	x64-uefi-installer | x64-uefi-installer-sdo
020-rc18	1117164049	2023-12-22 15:53:10 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc17	1116805207	2023-12-22 11:05:30 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc16	1116094347	2023-12-21 19:00:10 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default |

Tag	Pipeline	Date	pvr	Flashable images
				x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc15	1114650736	2023-12-20 16:27:35 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc14	1112569658	2023-12-19 11:47:30 +0100	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-ubuntu-default
020-rc13	1112434131	2023-12-19 03:04:02 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-uefi-default
020-rc12	1098122518	2023-12-06 20:14:20 -0300	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-uefi-default

Tag	Pipeline	Date	pvr	Flashable images
020-rc11	1097813893	2023-12-05 03:03:52 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo | x64-uefi-default
020-rc10	1031996045	2023-10-10 13:19:13 +0000	037	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
020-rc9	855619376	2023-05-03 08:58:37 +0000	037-28-g0a05035d	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
020-rc8	855565238	2023-05-03 08:07:59 +0000	037-28-g0a05035d	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
020-rc7	855519137	2023-05-03 07:21:37 +0000	037-28-g0a05035d	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-

Tag	Pipeline	Date	pvr	Flashable images
				default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
020-rc6	837049138	2023-04-14 06:12:36 +0000	037-26-g1aef7b14	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
020-rc5	835913963	2023-04-13 02:05:23 +0000	037-25-gd9d9f0f8	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
020-rc4	827675434	2023-04-04 12:58:29 +0000	037-23-gdc6e90e0	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | visionfive2-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
020-rc3	826792018	2023-04-03 20:21:08 +0000	037-21-g188cc5ec	aarch64-appengine-default | arm-appengine-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | visionfive2-

Tag	Pipeline	Date	pvr	Flashable images
				default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019	780054802	2023-02-15 03:04:37 +0000	035-49-ge7fed375	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc22	778545425	2023-02-15 03:04:37 +0000	035-49-ge7fed375	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc21	777719650	2023-02-14 21:58:20 +0000	035-49-ge7fed375	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc19	777332281	2023-02-14 15:17:51 +0000	035-48-g441a41aa	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc18	777241088	2023-02-14 03:05:42 +0000	035-48-g441a41aa	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-

Tag	Pipeline	Date	pvr	Flashable images
				default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc15	775551131	2023-02-13 03:04:55 +0000	035-44-g7b935e51	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc14	772776588	2023-02-09 15:06:55 +0000	035-44-g7b935e51	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc12	772430981	2023-02-09 10:33:04 +0000	035-44-g7b935e51	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc11	771641868	2023-02-08 18:34:30 +0000	035-44-g7b935e51	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default |

Tag	Pipeline	Date	pvr	Flashable images
				x64-uefi-installer | x64-uefi-installer-sdo
019-rc10	770983373	2023-02-08 10:16:54 +0000	035-44-g7b935e51	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc9	766836232	2023-02-03 15:22:10 +0000	035-42-gd12cd259	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc7	765917195	2023-02-02 20:49:58 +0000	035-38-gb60278cf	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc6	765706252	2023-02-02 17:29:58 +0100	035-38-gb60278cf	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | rpi0w-5.10.y-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc5	765691770	2023-02-02 17:18:28 +0100	035-38-gb60278cf	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-

Tag	Pipeline	Date	pvr	Flashable images
				toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | rpi64-5.10.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc4	765577344	2023-02-02 16:04:36 +0100	035-38-gb60278cf	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | rpi64-5.10.y-default | rpi64-5.4.y-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc3	762635522	2023-01-30 03:06:36 +0000	035-36-g8e45a08c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
019-rc1	729795200	2022-12-22 03:05:07 +0000	035-9-g3c26ad12	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018	692179061	2022-11-11 10:34:38 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-

Tag	Pipeline	Date	pvr	Flashable images
				qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc21	690665399	2022-11-10 03:06:17 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc20	689640188	2022-11-09 03:05:15 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc19	688633676	2022-11-08 11:44:13 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo

Tag	Pipeline	Date	pvr	Flashable images
018-rc18	686411499	2022-11-05 03:05:17 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc17	685629434	2022-11-04 10:46:18 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
015-rc16	683160956	2022-11-02 03:04:59 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc15	680044119	2022-10-28 16:14:33 +0000	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-

Tag	Pipeline	Date	pvr	Flashable images
				default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc14	679689835	2022-10-28 12:41:59 +0200	034-37-g84c3032c	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc13	657638599	2022-09-29 02:05:11 +0000	034-34-g52038ee2	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | UNKNOWN-image x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc12	622479249	2022-08-25 02:05:32 +0000	031-38-gdd67217a	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc11	620283205	2022-08-23 02:06:34 +0000	031-38-gdd67217a	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-

Tag	Pipeline	Date	pvr	Flashable images
				default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc10	619982679	2022-08-20 02:05:30 +0000	031-38-gdd67217a	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc9	608824813	2022-08-09 23:12:13 +0000	031-34-g9ebfd89d	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.10.y-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc8	607343369	2022-08-08 10:50:32 +0000	031-34-g9ebfd89d	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-

Tag	Pipeline	Date	pvr	Flashable images
				default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc7	605850541	2022-08-05 15:26:00 +0200	031-34-g9ebfd89d	aarch64-appengine-default | arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc6	605733885	2022-08-05 13:06:20 +0200	031-34-g9ebfd89d	UNKNOWN-image aarch64-appengine-default | UNKNOWN-image arm-appengine-default | beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image mips-appengine-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | UNKNOWN-image x64-appengine-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc5	604088790	2022-08-03 02:05:47 +0000	031-30-gf7c9d63a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc4	570092496	2022-06-22 12:23:04 +0200	031-21-g428d8c6b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default |

Tag	Pipeline	Date	pvr	Flashable images
018-rc3	570017565	2022-06-21 02:05:37 +0000	031-21-g428d8c6b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc2	562416919	2022-06-10 02:05:31 +0000	031-21-g428d8c6b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | rock64-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
018-rc1	488950540	2022-03-10 11:46:59 +0100	030-1-g9339533e	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
001-qa	485919589	2022-03-07 10:18:46 +0100	030-1-g9339533e	arm-rpi64-default
qa-test	445717769	2022-01-11 16:11:47 +0100	026-53-g355644a6	arm-rpi64-default
017-rc7	436336603	2021-12-24 03:05:45 +0000	026-52-gbf3bd5d6	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default |

Tag	Pipeline	Date	pvr	Flashable images
				rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
017-rc6	435983686	2021-12-21 03:06:19 +0000	026-52-gbf3bd5d6	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
017-rc5	432538666	2021-12-18 20:13:41 +0000	026-46-g58359d1f	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
017-rc4	431775980	2021-12-17 03:06:58 +0000	026-46-g58359d1f	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
017-rc3	376231931	2021-09-23 14:46:56 +0000	024-13-g489e3750	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
017-rc1	356407775	2021-08-20 02:06:10 +0000	022-15-g26ebb342	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-

Tag	Pipeline	Date	pvr	Flashable images
				default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
016	348823585	2021-08-05 20:30:10 +0000	022-11-g0399aa1d	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
016-rc1	348783113	2021-08-05 20:30:10 +0000	022-11-g0399aa1d	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
015	345315071	2021-07-30 02:06:44 +0000	022-11-g0399aa1d	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
015-rc2	345131805	2021-07-29 20:20:44 +0000	019-23-g7e76cc8b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
015-rc1	345056765	2021-07-29 16:56:47 +0000	019-23-g7e76cc8b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-

Tag	Pipeline	Date	pvr	Flashable images
				default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014	344887638	2021-07-29 12:53:41 +0000	019-23-g7e76cc8b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc25	344841125	2021-07-29 11:43:23 +0000	019-23-g7e76cc8b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc24	344535392	2021-07-28 22:43:11 +0000	019-23-g7e76cc8b	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc23	344441921	2021-07-28 18:31:17 +0000	019-21-gefb7396a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc22	344206396			

Tag	Pipeline	Date	pvr	Flashable images
		2021-07-28 02:05:31 +0000	019-21-gefb7396a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc21	341722025	2021-07-23 02:08:20 +0000	019-21-gefb7396a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc20	341514677	2021-07-22 22:39:03 +0200	019-21-gefb7396a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc19	339899178	2021-07-20 10:59:18 +0000	019-21-gefb7396a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc18	339858733	2021-07-20 02:05:09 +0000	019-21-gefb7396a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc16	339214551			

Tag	Pipeline	Date	pvr	Flashable images
		2021-07-19 02:05:15 +0000	019-21-gefb7396a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc14	334150042	2021-07-08 23:01:13 +0000	019-15-g93500baf	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc13	332471398	2021-07-06 13:49:08 +0200	019-6-g3dcef025	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc11	332051651	2021-07-05 15:38:43 +0000	019-6-g3dcef025	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc10	330279019	2021-07-01 10:55:10 +0000	019-1-g96682d6a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc9	328578605	2021-06-28 23:26:36 +0200	019-1-g96682d6a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default |

Tag	Pipeline	Date	pvr	Flashable images
				UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc7	328419720	2021-06-28 17:15:56 +0200	019-1-g96682d6a	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default
014-rc6	321283144	2021-06-15 11:39:52 +0000	016-58-gfb175095	beaglev-default | arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | UNKNOWN-image nv-tegra-4_9-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc5	321057432	2021-06-15 08:25:28 +0000	016-58-gfb175095	arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | rpi64-5.4.y-default | rpi64-5.4.y-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc4	300119129	2021-05-10 02:09:22 +0000	016-17-gedb1e4b1	arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc3	299149821	2021-05-07 02:10:14 +0000	016-17-gedb1e4b1	arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default |

Tag	Pipeline	Date	pvr	Flashable images
				x64-uefi-installer | x64-uefi-installer-sdo
014-rc2	263546967	2021-03-01 03:10:00 +0000	016-7-g9a6b0673	arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
014-rc1	260378112	2021-02-23 09:05:34 +0000	016-5-g2ffec40c	arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013	242287610	2021-01-15 03:11:09 +0000	013-13-ge94ede14	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc15	241655520	2021-01-14 13:12:16 +0000	013-13-ge94ede14	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc14	240487188	2021-01-12 13:15:42 +0100	013-13-ge94ede14	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc12	240427624	2021-01-12 10:20:22 +0000	013-13-ge94ede14	arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo

Tag	Pipeline	Date	pvr	Flashable images
013-rc13	240350849	2021-01-12 13:05:25 +0530	013-13-ge94ede14	arm-bpi-r2-default | arm-bpi-r64-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc11	240156956	2021-01-11 19:40:45 +0000	013-13-ge94ede14	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc10	239158957	2021-01-08 16:09:04 +0000	013-11-g5c5d70fb	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc9	232846255	2020-12-21 13:52:38 +0000	013-11-g5c5d70fb	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc8	232762507	2020-12-19 03:08:22 +0000	013-11-g5c5d70fb	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc7	231440397	2020-12-17 15:48:22 +0000	013-11-g5c5d70fb	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi2-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc6	228436627	2020-12-11 10:54:03 +0000	013-11-g5c5d70fb	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default |

Tag	Pipeline	Date	pvr	Flashable images
				arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc5	227950547	2020-12-10 13:15:29 +0000	013-11-g5c5d70fb	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc4	227797521	2020-12-10 03:09:33 +0000	013-11-g5c5d70fb	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc3	223206874	2020-11-30 14:55:05 +0000	013-9-g663452ce	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | arm-rpi64-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc2	220629526	2020-11-24 21:08:12 +0530	013-9-g663452ce	arm-bpi-r2-default | arm-bpi-r64-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
013-rc1	217480633	2020-11-17 03:09:21 +0000	013-7-gbb0e4139	arm-bpi-r2-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-odroid-c2-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012.1-rc8	211109922	2020-11-03 12:18:55 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012.1-rc7	205709150	2020-10-21 02:09:13 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo

Tag	Pipeline	Date	pvr	Flashable images
012.1-rc6	203253363	2020-10-15 20:29:06 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012.1-rc5	196885013	2020-10-01 11:48:46 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012.1-rc4	196831611	2020-09-30 20:54:26 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6_7-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012.1-rc3	190724831	2020-09-15 21:08:42 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6dl-default | arm-toradex-colibri-imx6ull-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012.1-rc2	189771696	2020-09-14 18:10:28 +0200	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6dl-default | arm-toradex-colibri-imx6ull-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012.1-rc1	187539624	2020-09-08 21:31:57 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6dl-default | arm-toradex-colibri-imx6ull-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012	187002288	2020-09-07 19:47:56 +0000	013-1-g3606fc94	arm-bpi-r2-default | arm-toradex-colibri-imx6dl-default | arm-toradex-colibri-imx6ull-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012-rc23	184879063	2020-09-02 17:10:53 +0200	012-11-g7117b07c	arm-bpi-r2-default | arm-toradex-colibri-imx6dl-default | arm-toradex-colibri-imx6ull-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | UNKNOWN-image x64-uefi-

Tag	Pipeline	Date	pvr	Flashable images
				installer | UNKNOWN-image x64-uefi-installer-sdo
012-rc20	178872310	2020-08-17 21:11:06 +0200	012-11-g7117b07c	arm-bpi-r2-default | arm-toradex-colibri-imx6ull-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | UNKNOWN-image x64-uefi-installer | UNKNOWN-image x64-uefi-installer-sdo
012-rc19	178835734	2020-08-17 19:20:27 +0200	012-11-g7117b07c	arm-bpi-r2-default | UNKNOWN-image arm-toradex-colibri-imx6ull-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | UNKNOWN-image x64-uefi-installer | UNKNOWN-image x64-uefi-installer-sdo
012-rc18	178786242	2020-08-17 17:31:50 +0200	012-11-g7117b07c	UNKNOWN-image arm-bpi-r2-default | UNKNOWN-image arm-toradex-colibri-imx6ull-default | UNKNOWN-image malta-qemu-default | UNKNOWN-image arm-rpi0w-default | UNKNOWN-image arm-rpi3-default | UNKNOWN-image arm-rpi4-default | UNKNOWN-image x64-uefi-default | UNKNOWN-image x64-uefi-installer | UNKNOWN-image x64-uefi-installer-sdo
012-rc11	176074132	2020-08-10 11:06:22 +0200	012-11-g7117b07c	arm-bpi-r2-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012-rc10	175466469	2020-08-07 16:21:49 +0200	012-11-g7117b07c	arm-bpi-r2-default | malta-qemu-default | arm-rpi0w-default | arm-rpi3-default | arm-rpi4-default | x64-uefi-default | x64-uefi-installer | x64-uefi-installer-sdo
012-rc9	173118037	2020-07-31 18:24:34 +0000	012-11-g7117b07c	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
012-rc8	172114925	2020-07-29 13:28:56 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
012-rc7	172030522	2020-07-28 21:30:52 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
012-rc6	171994497	2020-07-28 21:30:52 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
0.12-rc6	171798523	2020-07-28 19:52:17 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer

Tag	Pipeline	Date	pvr	Flashable images
012-rc5	171655513	2020-07-27 15:04:38 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
012-rc4	171230644	2020-07-27 15:04:38 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
012-rc3	164715233	2020-07-09 09:27:52 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
012-rc2	162943661	2020-07-03 18:34:24 +0200	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
012-rc1	162756662	2020-07-02 13:55:59 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
011	159326749	2020-06-23 17:18:38 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
011-rc6	158954616	2020-06-22 07:16:44 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
011-rc5	158694291	2020-06-22 07:16:44 +0000	012-3-g618ff901	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
011-rc4	157409434	2020-06-18 01:26:52 +0200	012-1-g4b9c576a	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
011-rc3	156508842	2020-06-15 19:29:40 +0000	011-16-gdab27a36	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
011-rc2	155143457	2020-06-11 10:16:53 +0200	011-12-g2c401cfa	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
011-rc1	150456356	2020-05-27 02:09:08 +0000	009-145-g24642965	arm-bpi-r2 | malta-qemu | arm-rpi0w | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer
010	139580621	2020-04-24 14:38:36 +0200	010	arm-bpi-r2 | malta-qemu | arm-rpi3 | arm-rpi4 | x64-uefi | x64-uefi-installer

9.5 Pantavisor BSP Devices

To use these devices, you will need an already flashed and ready-to-use Pantavisor device. If you don't have one, go to the guide on how-to prepare your device.

The BSP for these devices are stored at releases.json.

9.5.1 How to update your BSP

Once you have your device ready, you can update its BSP using `pvr merge`. Here is an example for a Raspberry Pi board:

```
pvr merge https://pantavisor-ci.s3.amazonaws.com/meta-pantavisor/024/raspberrypi-armv8-scarthgap/pantavisor-bsp-raspberrypi-armv8.pvrexport.tgz
pvr checkout
```

9.5.2 Latest Stable BSP Releases

Loading latest stable BSP releases...

9.6 Pantavisor BSP

9.6.1 Update a Pantavisor BSP

To use these devices, you will need an already flashed and ready-to-use Pantavisor device. If you don't have one, go to the guide on how-to prepare your device.

These BSPs are stored on Pantahub devices, and can be used from your device checkout with pvr:

```
pvr merge https://pvr.pantahub.com/pantahub-ci/rpi64_5_10_y_bsp_latest
pvr checkout
```

Stable

> **Note**
>
> This devices contain both the stable and the release candidate versions. Navigate through the device *History* tab to switch to the version you want to use.

Platform	Target	Pantahub devices
aarch64-appengine	aarch64-appengine	device
aarch64-generic	aarch64-generic	device
arm-appengine	arm-appengine	device
arm-generic	arm-generic	device
beaglev	beaglev	device
bpi-r2	arm-bpi-r2	device
bpi-r64	arm-bpi-r64	device
toradex-colibri-imx6_7	arm-toradex-colibri-imx6_7	device
malta	malta-qemu	device
mips-appengine	mips-appengine	device
mips-generic	mips-generic	device
nv-tegra-4_9	nv-tegra-4_9	device
odroid-c2	arm-odroid-c2	device
rock64	rock64	device
rpi0w-5.10.y	arm-rpi0w-5.10.y	device
rpi64-5.10.y	rpi64-5.10.y	device
x64-appengine	x64-appengine	device
x64-generic	x64-generic	device
x64-ubuntu	x64-ubuntu	device
x64-uefi	x64-uefi	device

Latest

Platform	Target	Pantahub devices
aarch64-appengine	aarch64-appengine	device
aarch64-generic	aarch64-generic	device
arm-appengine	arm-appengine	device
arm-generic	arm-generic	device
beaglev	beaglev	device
bpi-r2	arm-bpi-r2	device
bpi-r64	arm-bpi-r64	device
toradex-colibri-imx6_7	arm-toradex-colibri-imx6_7	device
malta	malta-qemu	device
mips-appengine	mips-appengine	device
mips-generic	mips-generic	device
nv-tegra-4_9	nv-tegra-4_9	device
odroid-c2	arm-odroid-c2	device
rock64	rock64	device
rpi0w-5.10.y	arm-rpi0w-5.10.y	device
rpi64-5.10.y	rpi64-5.10.y	device
x64-appengine	x64-appengine	device
x64-generic	x64-generic	device
x64-ubuntu	x64-ubuntu	device
x64-uefi	x64-uefi	device

9.6.2 About Pantavisor BSP

The BSP CI builds daily and stable versions of our BSP for several targets so you can use them and focus on your Pantavisor app development.

The BSP devices are automatically upgraded with GitLab CI in this project. Check out the jobs in the pipeline list and the release notes for each stable version in the list of tags.