Canonical Ubuntu

Contents

Requirements 2

Step 1: Create an Ubuntu One account 2
- 1.1 Snapcraft credentials . . . 2
- 1.2 Retrieve your developer account ID . . . 3

Step 2: Create the model assertion 3
- 2.1 Download a model file . . . 3
- 2.2 Edit the model file . . . 4
  - “authority-id” and “brand-id” . . . 4
  - timestamp . . . 4
  - snaps . . . 4
  - Complete model example . . . 5

Step 3: Sign the model assertion 6
- 3.1 Create a key . . . 6
- 3.2 Register the key . . . 6
- 3.3 Update the timestamp . . . 7
- 3.4 Sign the model . . . 7

Step 4: Build the image 8

Step 5: Boot the image 9

This tutorial will guide you through the steps required to create your own Ubuntu Core image, with your own selection of snaps, and install it on a Renesas RZ device.

Requirements

In addition to having a basic understanding of Linux and running commands from the terminal, this tutorial has hardware requirements.

For the host system used to build the image:
- Ubuntu 24.04 LTS or later installed
- MicroSD card reader
- Internet connectivity
- 10GB of free storage space

The target device:
- RZ/G2L or RZ/G2LC or RZ/G2UL or RZ/V2L
- 4GB+ microSD card
- keyboard and display (for setup only)
- Ethernet network connectivity

Step 1: Create an Ubuntu One account

You will need an Ubuntu One account with an uploaded public key of a locally generated SSH key pair.

See Use Ubuntu One for SSH for instructions on how to create an account and register an SSH key.

With your account created, ensure you first login and accept the Terms and Conditions. With this done, your Ubuntu One account is ready to use.

You will now need to retrieve your developer account identifier. This is part of your Ubuntu One account and is used to link your account to any Ubuntu Core images you create.

The next steps need to be performed in an existing Ubuntu LTS environment.

1.1 Snapcraft credentials

Your developer identifier can be retrieved with the `snapcraft` command, the tool that's also used to build and publish snaps. It can be installed by running:

```
sudo snap install snapcraft --classic
```

We now need to use `snapcraft` to export your login authentication credentials, and to place them within an environment variable.

```
snapcraft export-login credentials.txt
export SNAPCRAFT_STORE_CREDENTIALS=$(cat credentials.txt)
```

If you have yet to login to your Ubuntu One account with the `snapcraft` command, you will first be prompted for your email address, password, and second-factor authentication (if used).

1.2 Retrieve your developer account ID

With your authentication in place, the `snapcraft whoami` command will now display your developer identifier after the `id` field:

```
$ snapcraft whoami
email: <email-address>
username: <username>
id: xSfWKGdLoQBoQx88
permissions: package_access, package_manage, package_metrics,
package_push, package_register, package_release, package_update
channels: no restrictions
expires: 2024-04-17T10:25:13.675Z
```

In the output above, the example id is `xSfWKGdLoQBoQx88` – we'll use this ID for subsequent examples, but you should obviously use your own ID from now on.

Step 2: Create the model assertion

At the heart of custom Ubuntu Core image creation is the model assertion. An assertion is a signed recipe that describes the components that comprise a complete image. An assertion is provided as JSON in a text file which is signed by a GPG key associated with the publisher's Ubuntu One account.

The model contains: * identification information, such as the developer-id and model name. * which essential snaps make up the device system. * other required or optional snaps that implement the device application functionality.

See below for details on how to download and modify a model file to include your own selection of snaps.

2.1 Download a model file

The quickest way to create a new model assertion is to edit a model that already exists. Reference models for every supported Ubuntu Core device can be found in the canonical/models GitHub repository.

For this project, we're going to modify the 64-bit reference model for the Renesas RZ devices: ubuntu-core24-renesas-rzg2-arm64.json.

Download and save the file locally with the following wget command. We've called the file `my-model.json`:

```
wget -O my-model.json \
https://raw.githubusercontent.com/canonical/models/refs/heads/master\
/devices/renesas/rzg2/ubuntu-core24-renesas-rzg2-arm64.json
```

2.2 Edit the model file

We now need to edit `my-model.json` using a text editor:

```
nano my-model.json
```

The following fields in `my-model.json` need to be changed:

“authority-id” and “brand-id”

```
"authority-id": "canonical",
"brand-id": "canonical",
```

These properties define the authority responsible for the image. Change both instances of the string “canonical” to your developer id, retrieved with the `snapcraft whoami` command. (“xSfWKGdLoQBoQx88”, in our example output). This links the image to your Ubuntu One account and ensures that only you can push image updates to devices using your model.

timestamp

```
   "timestamp": "2025-08-19T11:18:21+00:00",
```

This needs to be provided at the end of the process; we’ll come back to this.

snaps

```
    "snaps": [
        {
            "default-channel": "24/stable",
            "id": "8icb75cOHIxavWWXmkVR2UJePXX3HFkF",
            "name": "renesas-rz",
            "type": "gadget"
        }
    ]
```

This section lists the snaps to be included in the image. renesas-rz (shown above), renesas-kernel, core24 and snapd are the four snaps required for a functioning Ubuntu Core device. The additional console-conf snap is required for Ubuntu Core 24 devices.

Console-conf is the interactive setup utility that’s used to configure the network and the default user when the device is first booted. This is marked as optional, but for this tutorial, it needs to be mandatory to configure the device when it first boots. To do this, delete the `"presence": "optional"` line (line 41) and delete the comma at the end of the preceding line.

Additional snaps are included using the same schema, with each snap requiring the following fields: - `name`: simply the snap name. - `type`: the type of snap. This is `app` for standard application snaps. - `default-channel`: the channel to install the snap from. - `id`: a unique snap identifier associated with every published snap. This is `snap-id` in the output from `snap info <snap-name>`.

Snaps do not have dependencies, but they do require the presence of the base snap they were built on. This can be added with the following:

```
        {
            "default-channel": "latest/stable",
            "id": "dwTAh7MZZ01zyriOZErqd1JynQLiOGvM",
            "name": "core24",
            "type": "base"
        }
```

The `snap-id` for a snap is in the output of the `snap info <snap-name>` command.

Complete model example

After finishing all your edits, the completed my-model.json text file should now contain the following:

```
{
  "type": "model",
  "authority-id": "your-id",
  "revision": "1",
  "series": "16",
  "brand-id": "your-id",
  "model": "renesas-rzg2",
  "architecture": "arm64",
  "base": "core24",
  "grade": "signed",
  "timestamp": "2025-08-14T08:03:54+00:00",
  "snaps": [
    {
      "default-channel": "24/stable",
      "id": "8icb75cOHIxavWWXmkVR2UJePXX3HFkF",
      "name": "renesas-rz",
      "type": "gadget"
    },
    {
      "default-channel": "24/stable",
      "id": "TfkSeypPcvFMoNky5XzwqV6pJSKZ43Hi",
      "name": "renesas-kernel",
      "type": "kernel"
    },
    {
      "default-channel": "latest/stable",
      "id": "dwTAh7MZZ01zyriOZErqd1JynQLiOGvM",
      "name": "core24",
      "type": "base"
    },
    {
      "default-channel": "latest/stable",
      "id": "PMrrV4ml8uWuEUDBT8dSGnKUYbevVhc4",
```

```
      "name": "snapd",
      "type": "snapd"
    },
    {
      "default-channel": "24/stable",
      "id": "ASctKBEHzVt3f1pbZLoekCvcigRjtuqw",
      "name": "console-conf",
      "type": "app"
    }
  ]
}
```

Step 3: Sign the model assertion

After a model has been created or modified, it must be signed with a GPG key to become a model assertion. This ensures the model cannot be altered without the key and also links the created image to both the signed version of the model and your Ubuntu One account

3.1 Create a key

First make sure there are no keys already associated with your account by running the `snapcraft list-keys` command (you will only have a key if you've previously signed an assertion; if you already have a key, you can use that one):

```
$ snapcraft list-keys
No keys have been registered.
See 'snapcraft register-key --help' to register a key.
```

Now use `snapcraft` to create a key called my-model-key (the name is arbitrary):

```
$ snapcraft create-key my-model-key
Passphrase: <passphrase>
Confirm passphrase: <passphrase>
```

As shown above, you will be asked for a passphrase. You need to remember this as you'll be prompted to enter it whenever you use the key, including the very next step.

TIP: Rather than creating a key for every device, the same key is typically used across all models or model families.

3.2 Register the key

We now need to upload the key and register it with your Ubuntu One account. This is accomplished with register-key:

```
$ snapcraft register-key my-model-key
Enter your Ubuntu One e-mail address and password.
If you do not have an Ubuntu One account,
```

```
you can create one at https://snapcraft.io/account
Email: <Ubuntu-SSO-email-address>
Password: <Ubuntu-SSO-password>

Registering key ...
Done. The key "my-model-key" (<key fingerprint>) may be used
to sign your assertions.
```

Regardless of whether you're logged in with snapcraft, you will be asked for your account and password details. You'll also need to unlock the key with your passphrase, and when the process is complete, the `snapcraft list-keys` command will now list the registered key:

```
$ snapcraft list-keys
    Name          SHA3-384 fingerprint
*   my-model-key  <key fingerprint>
```

3.3 Update the timestamp

As mentioned earlier, the timestamp in the model assertion must be set to a time and date after the creation of our key. This means we need to edit `my-model.json` to update the timestamp with the current time.

```
    "timestamp": "2025-08-14T08:03:54+00:00",
```

This is a UTC-formatted time and date value, used to denote the assertion's creation time. It needs to be replaced with the current time and date, which can be generated with the following command:

```
$ date -Iseconds --utc
2025-08-19T11:18:21+00:00
```

3.4 Sign the model

A model assertion is created by feeding the JSON file into the `snap sign` command with your recently-created key name and capturing the output in the corresponding model file:

```
snap sign -k my-model-key my-model.json > my-model.model
```

You will again be asked for your key's passphrase.

The resultant `my-model.model` file contains the signed model assertion and can now be used to build the image.

If you encounter a gpg: signing failed error while signing your assertion from a non-desktop session, such as over SSH, run `export GPG_TTY=$(tty)` first.

Step 4: Build the image

Images are built from the recipe contained in the model assertion using ubuntu-image, a tool to generate a bootable image.

First, install the `ubuntu-image` command from its snap:

```
sudo snap install ubuntu-image --classic --edge
```

The `ubuntu-image` command requires three arguments; `snap` to indicate we're building a snap-based Ubuntu Core image, `--allow-snapd-kernel-mismatch` to ignore a difference in the versions of snapd we're using, and finally, the filename of our previously-signed model assertion to build an image:

```
$ ubuntu-image snap --allow-snapd-kernel-mismatch my-model.model
[0] prepare_image
WARNING: proceeding to download snaps ignoring validations,
this default will change in the future.
For now use --validation=enforce for validations to be taken
into account, pass instead --validation=ignore to preserve
current behavior going forward
WARNING: the kernel for the specified UC20+ model does not
carry assertion max formats information, assuming possibly
incorrectly the kernel revision can use the same formats
as snapd
WARNING: snapd 2.68+ is not compatible with a kernel
containing snapd prior to 2.68
[1] load_gadget_yaml
[2] set_artifact_names
[3] populate_rootfs_contents
[4] generate_disk_info
[5] calculate_rootfs_size
[6] populate_bootfs_contents
[7] populate_prepare_partitions
[8] make_disk
[9] generate_snap_manifest
Build successful
```

You can safely ignore the warnings, and the entire process should only take a few minutes (depending on your connectivity), with the creation of a `ubuntu.img` Ubuntu Core image file being the end result.

TIP: The console-conf user-interface that configures the network and system user when a device first boots has migrated to an optional snap in Ubuntu Core 24 and later.

This is covered in Create a model assertion, but `ubuntu-image` can add `console-conf` at image build time with an additional `--snap console-conf` argument. For more details on these changes, see console-conf for device onboarding.

Step 5: Boot the image

Now that you have a custom image for a Renesas RZ/G devices on a microSD card. Follow the intructions here to flash the image and boot the device.

The instruction is for G2L, but it will work for G2LC, G2UL as well as V2L. The image is identical, the only difference are the bootassets - they are specific per device. You can distinguish them easily: the board name is mentioned in the filename.

Boot assets can be found pre-built at the official ubuntu renesas-iot page. Please make sure to download the boot assets for Ubuntu Core.

Each board has a Secure Boot variant, please make sure to pick the `-secure` file if you have this variant board and want to enable the Secure Boot.

Contents of each bootasset tarball is as follows:

Name	Flash Writer
G2L	Flash_Writer_SCIF_RZG2L_ SMARC_DDR4_2GB.mot
G2LC	Flash_Writer_SCIF_RZG2LC_ SMARC_DDR4_2GB.mot
G2UL	Flash_Writer_SCIF_RZG2UL_ SMARC_DDR4_1GB_1PCS.mot
V2L	Flash_Writer_SCIF_RZV2L_SMARC_DDR4_2GB.mot

Name	FIP file
G2L	fip-smarc-rzg2l_pmic.srec
G2LC	fip-smarc-rzg2lc.srec
G2UL	fip-smarc-rzg2ul.srec
V2L	fip-smarc-rzv2l.srec

Name	2nd stage BL
G2L	bl2_bp-smarc-rzg2l_pmic.srec
G2LC	bl2_bp-smarc-rzg2lc.srec
G2UL	bl2_bp-smarc-rzg2ul.srec
V2L	bl2_bp-smarc-rzv2l_pmic.srec