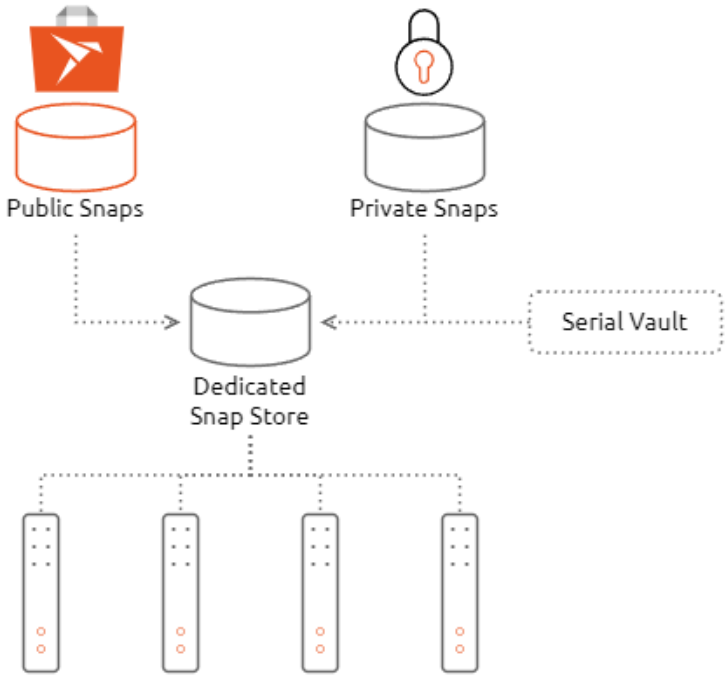
Canonical

# Serial Vault

Canonical

# How-to




*The serial vault serves an important function in the management of a Dedicated Snap Store*

The Serial Vault is a Canonical-hosted multi-tenant web portal offered to customers. Customers may choose to set up their own dedicated Serial Vault service.

The main purpose of the Serial Vault is to provide a signed Serial Assertion to devices, which allows them to authenticate against a dedicated Snap Store. The Serial Vault also enables other functions, such as remodeling[1] support.

The Serial Vault provides a user interface and an API to enable a device to receive digitally signed documents(assertions[2]) that are used for authentication and authorization. An authenticated device can have access to restricted snaps from a Dedicated Snap Store, providing a customized experience to the device owners.

The main assertions that are handled by the Serial Vault are:

- Account Assertion
- Account Key Assertion
- Serial-Request Assertion
- Model Assertion
- Serial Assertion

All of these are used by the device, Serial Vault and Dedicated Snap Store to verify and manage the access of a device.

[1] https://ubuntu.com/core/docs/uc20/remodelling
[2] https://ubuntu.com/core/docs/reference/assertions

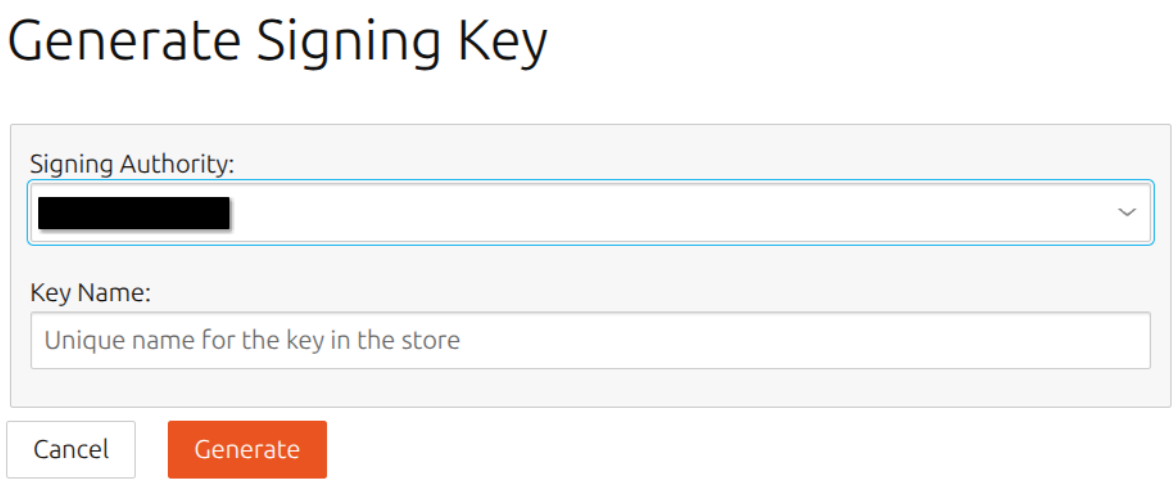
# 1. Generate a serial signing key

You can generate serial signing keys within the Serial Vault. For security reasons, this is preferred over generating a key locally and uploading it to the Serial Vault (as described in *Import a serial signing key* (page 5)).

The operations on this page are to be performed from the "Signing Keys" section of the Serial Vault, while logged in with an administrator account.

## 1.1. Generating the key

From the "Signing Keys" section, click on the gear icon .



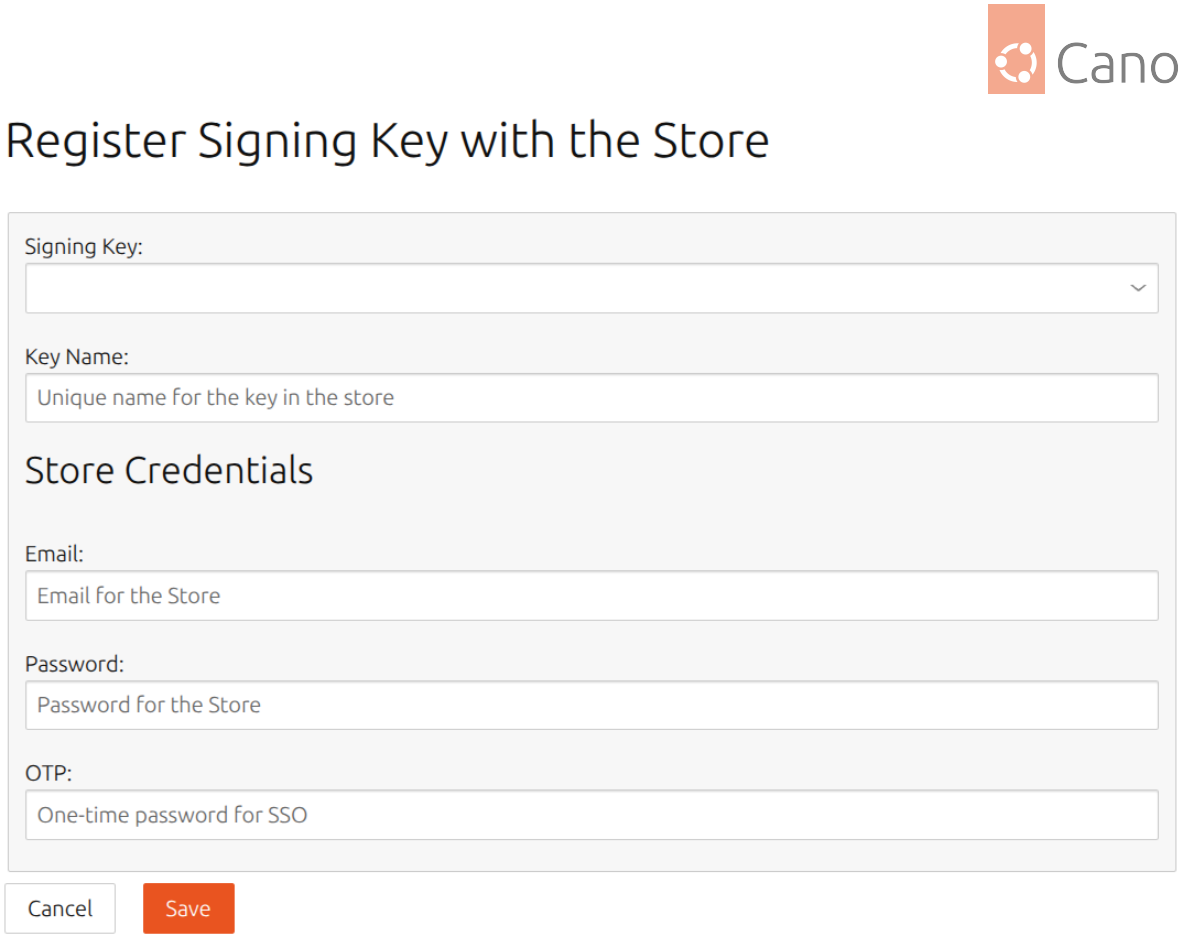

The "Signing Authority" field will be pre-populated with your brand ID.

Input a name for the key that is unique for your store in the "Key Name" field, and click on **Generate**.

## 1.2. Register the key to the store

Now you can register your serial key. Ensure you use the correct ID from the "Signing Keys" section, listed in the "KEY ID" column.

Click on the shopping cart icon .




Choose the key you just created from the “Signing Key” drop-down menu. All keys stored in the Serial Vault will be visible, with each entry listed as `brand-id/key-id`.

Input the same name for the key in the “Key Name” field.

Finally, input your **Brand account** credentials and click on **Save**.

# 2. Import a serial signing key

**Note:**

This option for generating serial signing keys is not recommended. We suggest creating your serial signing keys in the Serial Vault so that the private keys are never available externally, reducing the attack surface. For instructions, please see *Generate a serial signing key* (page 3).

## 2.1. Generating a serial signing key locally

To proceed, you need to generate a signing key as the Brand Account. This will be uploaded to the Serial Vault and used to sign Serial Assertions. The key must abide by the following conditions:

- This key must be generated and registered as the Brand Account
- This key must be passwordless

**Note:**

Generating the key takes some time. Moving the mouse or typing can help to speed up the process, as does installing the `rng-tools` Debian package beforehand.

Ensure you **do not enter a passphrase** for this key when prompted. This special key is used exclusively for providing devices access to the store and must be passwordless. When prompted in the terminal, or if a pop-up displays, do not enter text as a Passphrase and do not enter text to Confirm it. If you do, this key will not function to sign Serial Assertions.

```
$ snapcraft create-key serial
Passphrase:  # must be a passwordless key, so press Enter twice
Confirm passphrase:

[ . . . ]
```

## 2.2. Register a serial signing key

Register the serial key with the following command:

```
$ snapcraft register-key serial

[ . . . ]
```

When prompted, enter the credentials for the Brand Account.

**Note:**

The key does not function if another account is specified. This is by design in order to ensure the Brand Account's scope of authority.

## 2.3. Verify the serial key exists

Use `snapcraft list-keys` to verify a key named serial exists:

```
$ snapcraft list-keys
    Name          SHA3-384 fingerprint
*   serial        <fingerprint>
```

## 2.4. Armor the serial key

Export the serial key as an armored file so that it can be safely uploaded to the Serial Vault, as follows:

```
$ gpg --homedir ~/.snap/gnupg --armor --export-secret-key serial > serial.asc
```

You should now have a `serial.asc` file registered that can be imported to the Serial Vault.

## 2.5. Uploading the key to the Serial Vault

The ASCII-armored file can be uploaded to the Serial Vault. To import the serial signing key, Log in to the Serial Vault with the Administrator account. Select the Signing Keys section at the top, and click the + button to import a signing key. Note that the Serial Vault cannot be logged in via the Brand Account.


Enter `serial` in the Key Name field as shown in the screenshot above.

In the Signing Authority field, select the account ID of the Brand Account. This is the same value as the `developer-id` displayed by `snapcraft whoami` when you are logged into snapcraft as the Brand Account.

Click **Choose File**, and browse to and select the `serial.asc` file on your local system as created above.

Click **Save** to complete the upload.

## 2.6. Backing up and protecting keys

The signing keys that have been generated are an essential part of verifying the authenticity of a device at the Dedicated Snap Store. So these keys need to be protected and backed up. They are stored by default in `~/.snap/gnupg` on the current machine.

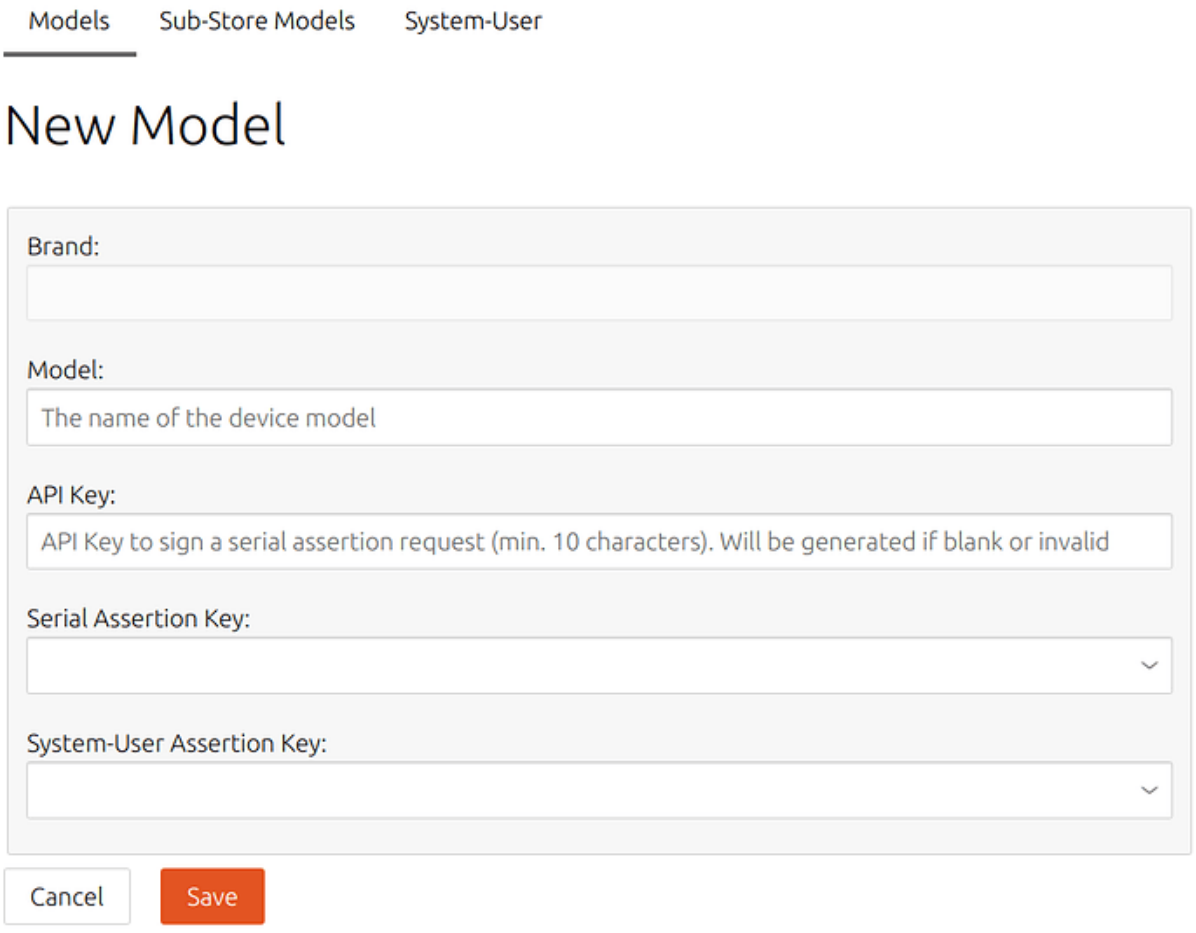
Whilst the Serial Vault does hold the private key (encrypted in the database), it does not provide a mechanism to download a signing key. The Dedicated Snap Store only has the public part of the key. So it is important to ensure that the generated keys are backed up.

# 3. Register a new device model name

When you build an image, you provide a Model Assertion that defines the devices running the image. For example, the Model Assertion defines the devices as part of your Brand. Here, you create a corresponding Model definition in the Serial Vault.

Note: Both the Serial Vault Model configuration and the Model Assertion (created later) must use the same Model name.

Log in to the Serial Vault[3] with the Administrator account. Select the Models section at the top, click the + button to create a new model.




The Brand field is pre-populated with your Brand account ID and is uneditable.

Enter the desired model name in the Model field. The name you enter here must be the same as used on your Model Assertions that you will create.

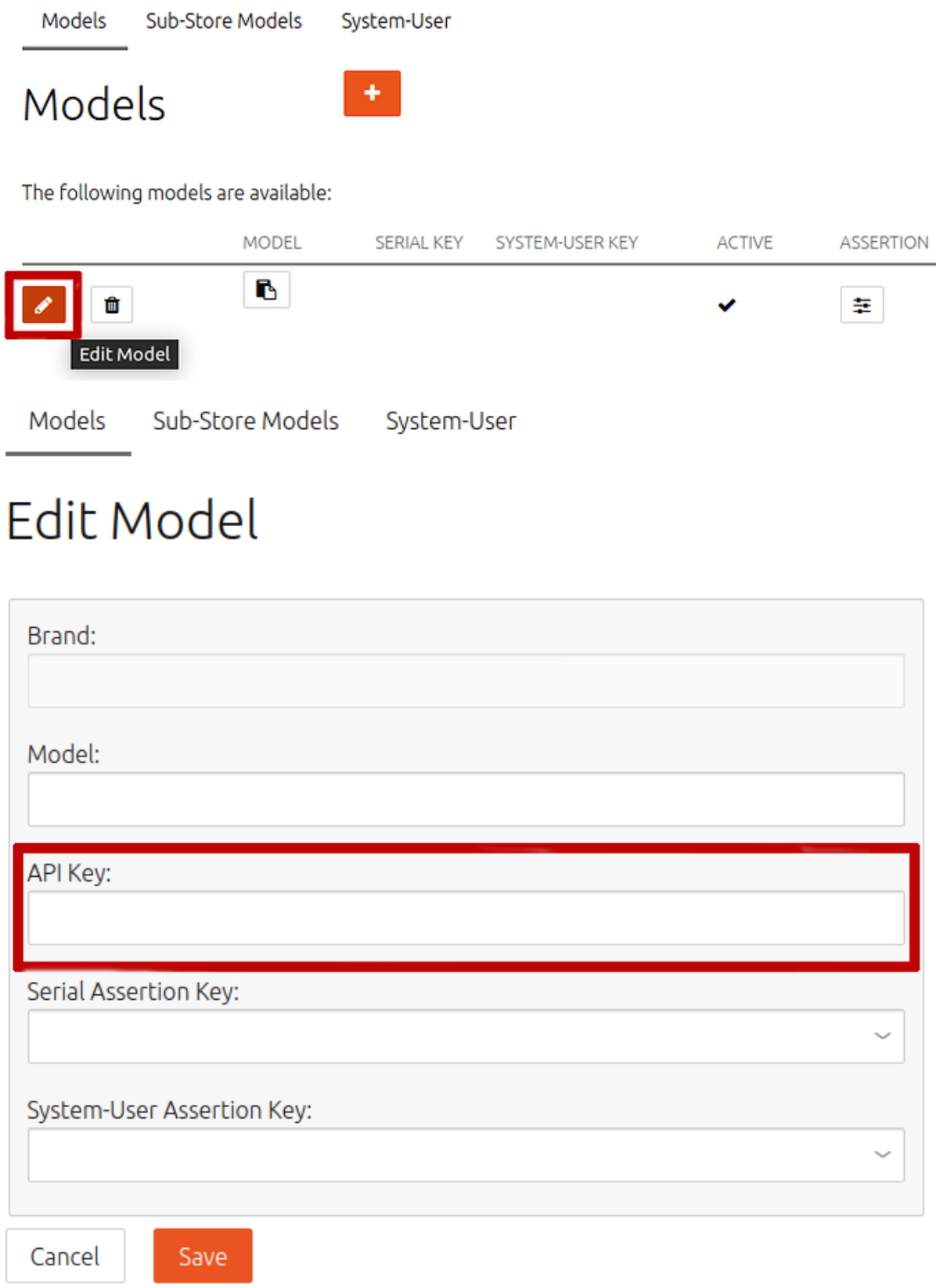
For Serial Assertion Key, select the reference for an already uploaded serial key.

For System-User Assertion Key, select a key that will be used for system-user generation. In production, you should create and use a dedicated key for the system-user.

The API Key field can be left blank. Its value is generated after the form is submitted. The API Key is used in the creation of a gadget snap that can be included in a factory image. The

[3] https://serial-vault-admin.canonical.com/

generated API Key can be retrieved from the edit model menu by pressing the pencil icon as shown below.



# 4. Generate a model signing key

To sign model assertions you will need to create and register a model signing key. The procedure is similar to generating a serial signing key locally.

Logged in as the Brand account in snapcraft and in the same home directory used before for generating the serial key for the Brand account, create and register a new key named `model`. This will be used exclusively to sign model assertions.

Usually the model key is not uploaded to the Serial Vault. However, some advanced Serial Vault functionality does involve uploading a model key, so you may want to upload this or another registered model signing key to the Serial Vault later.

## 4.1. Verify you are logged into snapcraft as the Brand Account

Use `snapcraft whoami` to verify you are logged on as the Brand account:

```
$ snapcraft whoami
email:        <YOUR-BRAND-EMAIL>
developer-id: <YOUR-BRAND-ACCOUNT-ID>
```

## 4.2. Create the model signing key

Create a key named model, and verify it is created and not yet registered,

**The model key must have a passphrase.**

```
$ snapcraft create-key model
Passphrase: ...      # Passphrase is needed
Confirm passphrase: ...

$ snapcraft list-keys
    Name          SHA3-384 fingerprint
*   serial        <fingerprint>
*   model         <fingerprint> (not registered)
```

## 4.3. Register the model key

Register the key providing the Brand Account credentials, and verify it is registered. The key will not function if another account is specified. This is by design in order to ensure the Brand Account's scope of authority.

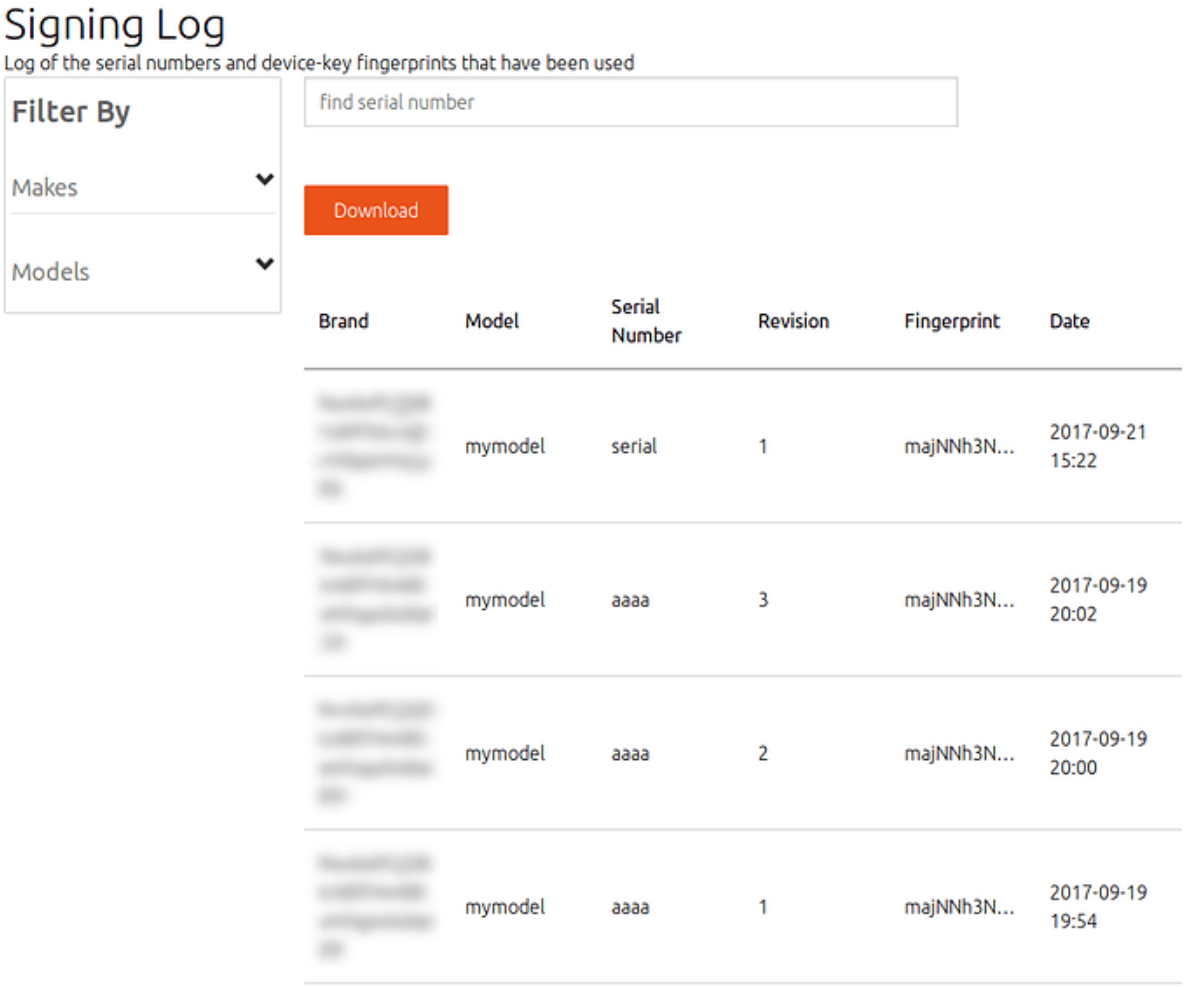
```
$ snapcraft register-key model
[ . . . ]

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

# 5. Check the signing log

You can review the Signing Log to check whether serial assertions have been signed and issued. This could be used to inspect the registered devices.

The Signing Log displays the list of serial numbers and device keys that have been used to sign serial assertions. You can also filter the log.



# 6. Create a system-user assertion

**Note:**

This feature has been deprecated. Please see the updated system-user assertion functionality[4].

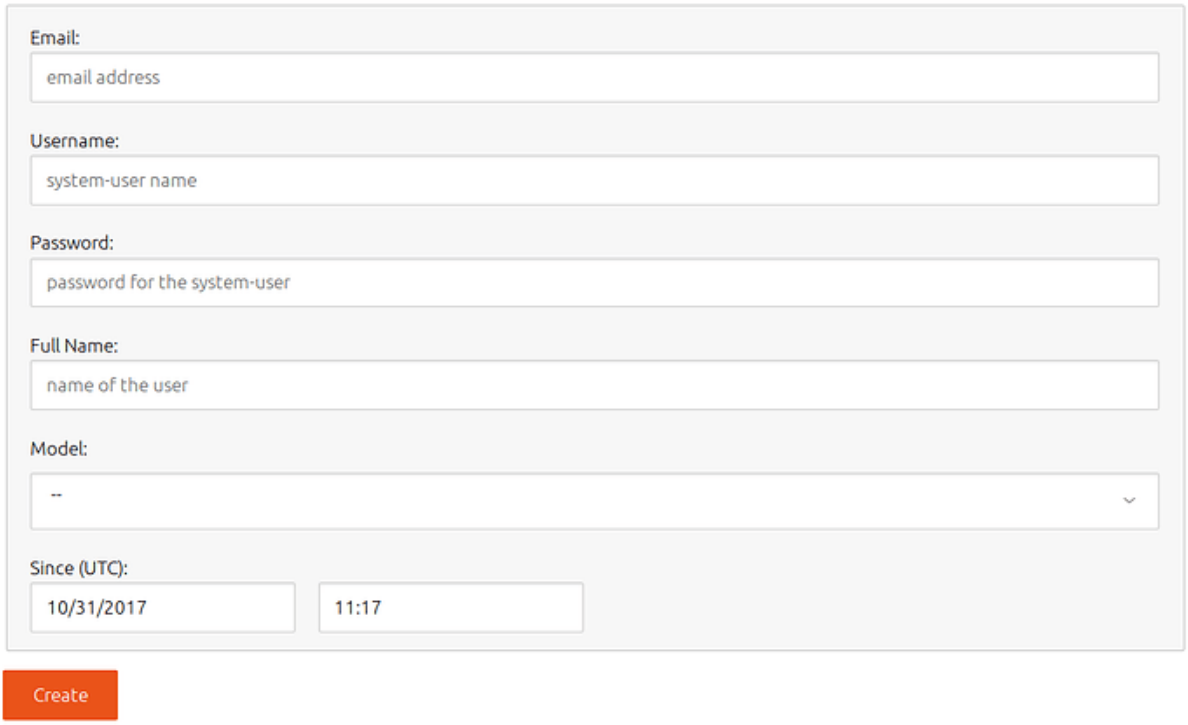
[4] https://ubuntu.com/core/docs/system-user

A system-user assertion[5] allows you to create a user on an unmanaged Ubuntu Core system. The system-user assertion can be pre-populated into an image, or it can be auto-imported from a USB drive. More about the system user configuration[6] can be found in the Ubuntu Core docs.

Through the System-User menu in the Serial Vault you can create a system-user assertion for your brand and model as follows:

1. Click System-User.
2. Enter the email of the system user.
3. Enter the desired login name of the system user in the Username field.
4. Enter the system user password in the Password field.
5. Enter the complete name of the user in the Full Name field.
6. Select the model of the device this user is being created for from the Model combo box.
7. Set date and time in UTC since this assertion is valid in Since (UTC) field. This date must be after the date on which the key used to sign the system user assertion was registered.

[5] https://ubuntu.com/core/docs/reference/assertions/system-user
[6] https://ubuntu.com/core/docs/system-user




After pressing **Create**, the generated assertion is displayed. Click **Download** to save it to a local file.

You can name this file `auto-import.assert` and place it in the root directory of a USB drive. If the USB drive is inserted in an unmanaged Ubuntu Core system, the assertion is imported and a system user is created.

# 7. Signing keys

The Serial Vault provides encrypted storage for signing keys intended for use with a Dedicated Snap Store. Signing keys can be generated locally and uploaded, and users with Admin permissions can generate keys from within the Serial Vault. A signing key needs to be registered with the Dedicated Snap Store so it can be used to verify the signature of serial assertions. The Serial Vault holds the private key, whereas the Dedicated Snap Store holds only the public key as an account-key assertion.

**Note:**

The private key of any key pair generated within the Serial Vault cannot be downloaded for local use. If you wish to sign both model and serial assertions with the same key, ensure the key is generated locally and then uploaded to the Serial Vault.

The steps needed with local key generation and registration are:

1. Generating the key(s)
2. Registering the key with the Dedicated Snap Store
    - See *Register a signing key with limited roles* (page 14) on constraining a key for only certain assertion types
3. Exporting the key as ASCII-armored
4. Uploading the ASCII-armored key file to the Serial Vault

The Serial Vault will need the private keys uploaded to its database for the following keys:

- Key for signing serial assertions
- Key for signing system-user assertions
- (if model pivoting is needed) Key for signing model assertions

Although one key can be used for each of these three purposes, it is recommended to use a separate signing key for each type of assertion.

Signing keys that are to be uploaded to the Serial Vault need to be passwordless (i.e. keeping a blank password when generating the key). Using a key that has a password results in an error when uploading the signing key.

## 7.1. Register a signing key with limited roles

Role-scoped keys are signing keys that are limited to be able to sign only specific assertion types. These limitations can be set when first registering the key with the Dedicated Snap Store. For example, a key can be registered for only signing serial assertions, and can be further constrained to the serial assertions for only certain models. This improves security as it limits the potential impact of a leaked key.

Customers currently cannot register a role-scoped signing key themselves. They must raise a support request as documented below.

A key can be scoped to a single or a combination of the following assertion types:

- Serial
- Model
- System-user
- Preseed

For the serial, model, and preseed assertion types, keys can be further constrained on model names matching a certain regular expression, e.g. `foo-.*`

**Note:**

If role-scoped keys are in use, it is recommended to limit the roles of all available keys for increased security.

### 7.1.1. Procedure for requesting a role-scoped key

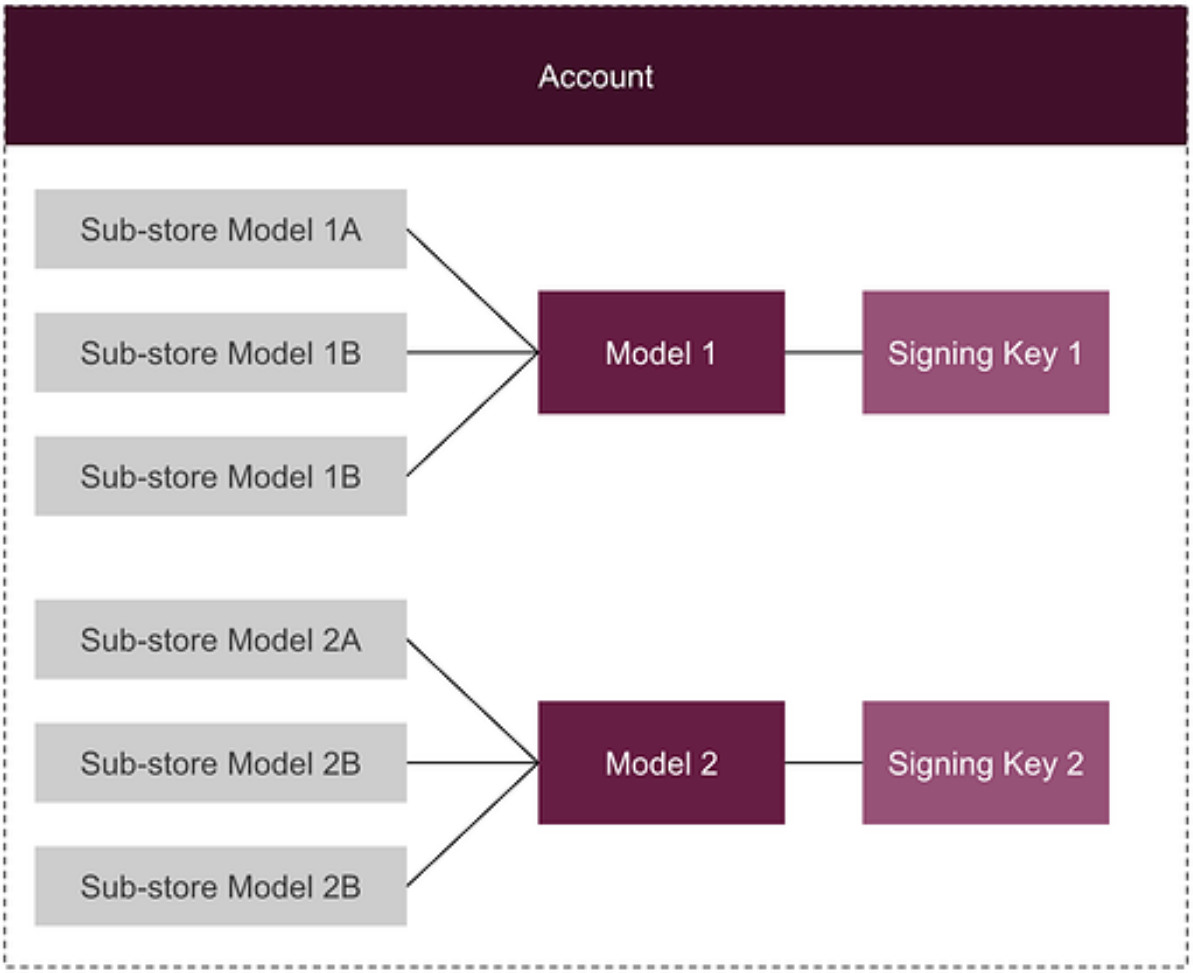
To register a role-scoped key, raise a support request with the Dedicated Snap Store with the following details:

- Account ID of the brand account to which the key would be registered
- Name for the key
- Verbatim base64-encoded output of `snap export-key`, ran against the locally created signing key they wish to register
- Model name constraints for the serial, model, and preseed assertion types, if any

Currently, roles can only be defined for new keys. Once registered, the role(s) of a role-scoped signing key cannot be changed.

# 8. Device model and identity

The Serial Vault needs to be configured with the details of the models and signing keys that will be authorized to use the functionality for an account.




The diagram above shows how the different entities relate to each other. The account, model and signing keys all belong to the same Brand, which is the account-id of the Brand in the store[7]. The ID is shared across the entities, though they may be referred to differently for each entity e.g. brand-id on the model and authority-id on the signing-key.

## 8.1. Models

Every device is provisioned using an Ubuntu Core image that includes a signed model assertion and Gadget snap[8]. The model assertion[9] holds important information related to the identity of the device:

- brand-id: the ID of the Brand
- model: the model name of the device
- store: the ID of the Dedicated Snap Store

[7] https://dashboard.snapcraft.io/dev/account/
[8] https://ubuntu.com/core/docs/gadget-snaps
[9] https://ubuntu.com/core/docs/reference/assertions/model

- sign-key-sha3-384: the key ID of the signing key (must be registered with the Dedicated Snap Store)

An essential part of a device's identity is its serial number. This and that is recorded in the serial assertion[10], rather than the model assertion. The serial and model assertions are used to authenticate the device with the Dedicated Snap Store. The connecting new devices page[11] provides an overview of the onboarding process.

The serial assertion is not a part of the device image, as the serial number will differ from one device to another. Each device retrieves its serial assertion from the Serial Vault. For the Serial Vault to recognize a device as valid, it needs to have a model entity defined for the device.

The Model entity provides a number of attributes that must match the device (brand-id and model), and an API key that must match that held in the Gadget snap. The model must be associated with a signing key so it can generate and sign a serial assertion.

Models can be added, changed and deleted from the Models section of the Serial Vault. Once a model is added, it is immediately available to provide a signed serial assertion to snapd.

[10] https://ubuntu.com/core/docs/reference/assertions/serial
[11] https://ubuntu.com/core/services/guide/connecting-devices

# 9. Environment setup

It is recommended that Ubuntu desktop[12] is used for the creation and management of signing keys.

## 9.1. Install snapcraft and login as the brand

Snapcraft is usually pre-installed. If it is not, you can install it as follows:

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

Login to snapcraft as the Brand Account:

```
$ snapcraft login
Enter your Ubuntu One Email address and password.
If you do not have an Ubuntu One account, you can create one at https://snapcraft.
io/account
Email: <YOUR-BRAND-EMAIL>
Password: ******

Login successful.

$ snapcraft whoami
email:        <YOUR-BRAND-EMAIL>
developer-id: <YOUR-BRAND-ACCOUNT-ID>
```

Now you should be logged in with the Brand Account.

[12] https://ubuntu.com/download/desktop

# 10. IoT Professional Services

Bringing IoT products to market involves varied requirements, ranging from embedded engineering, app development, backend hosting, software update infrastructure, maintenance and after-sales customer support. Deploying these capabilities at scale typically requires large upfront investments. The complexity involved in acquiring and orchestrating these requirements delays time-to-market.

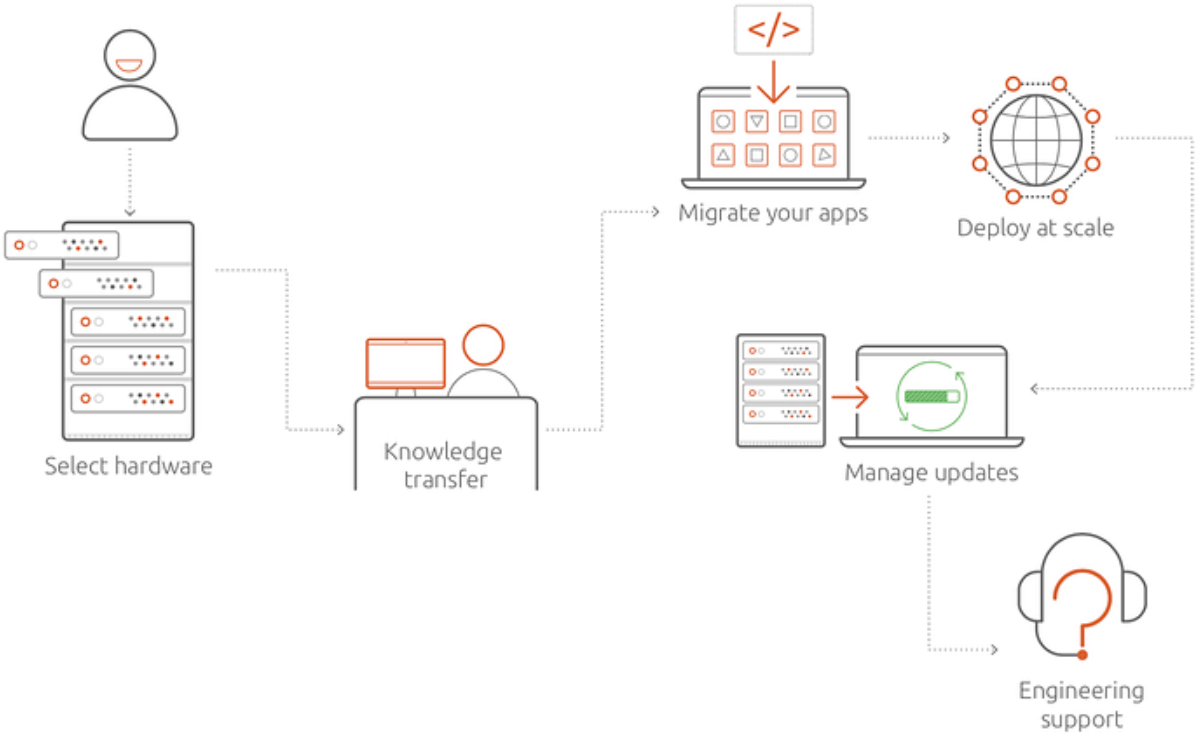



*Illustration of common factors to be considered when bringing an IoT product to market*

Our IoT Professional Services bring agility to enterprise IoT projects. Through our offerings, enterprises embrace a lean methodology for IoT product commercialization. Innovative companies can leverage our IoT portfolio to bring the first version of a product to market quickly, carry out product discovery and gather initial customer feedback, without committing more resources to scaling their product.

# 10.1. What do we offer?

| | |
|---|---|
| Ubuntu Core | A container operating system built specifically for IoT and devices, optimized for high security, performance and reliability |
| Hardware certification and enablement | Pre-certified hardware available, or hardware certification and enablement packages for any board |
| IoT app embedding | Porting of up to 3 IoT applications to snaps[13] |
| Hosted cloud infrastructure | Dedicated Snap Store for up to 1,000 devices |
| Over-the-air (OTA) update services | Monthly OTA software updates for one year |
| Technical support | Embedded technical support options |
| Consulting | 3 days of professional consulting |
| Lead time | Standard delivery in two weeks, advanced hardware options have longer delivery time |
| Add-ons (optional) | Board bring-up (custom kernel with BSP integration)Full disk encryptionSecure bootKernel LivepatchDedicated training workshops |

The following advanced security, integrity and resilience options harden smart devices exposed to challenging environments. All are available as part of our IoT Professional Services packages.

## 10.1.1. Secure boot

Ensures the integrity of both the boot mechanism and the operating system environment it bootstraps.

- Guarantees a device can only run a certified workload
- Secures a device against both physical and remote attacks
- Verifies boot binaries, and kernel, against known keys held in the device firmware

Find out more about secure boot[14]

## 10.1.2. Full disk encryption

Essential for devices with personal information in regulated industries:

- Hardware key management
- Optional key escrow
- Choice of ciphers and hardware acceleration
- Minimal performance impact

[13] http://snapcraft.io/docs/getting-started
[14] https://ubuntu.com/core/features/secure-boot

- TPM integration with the current CA (x86 only)

Find out more about full disk encryption (FDE)[15]

### 10.1.3. FIPS certification

Allows your devices to meet Federal information processing requirements:

- FIPS-certified kernel and cryptographic libraries
- FIPS certification takes place every six months
- Fully compliant devices must restrict updates to certified versions (x86 only)

### 10.1.4. Kernel Livepatch

Reduce the number of reboots by live patching the running kernel against critical vulnerabilities. Requires specific certified kernel and x86 architecture.

- Maximize service availability
- Fixes are applied automatically, without restarting your system
- Reduces downtime, keeping systems both secure and compliant

### 10.1.5. High availability Kubernetes

With Canonical MicroK8s[16] and Charmed Kubernetes[17], you gain a fully CNCF conformance cloud-native Kubernetes for device application operations, including clustering for high availability, service mesh support and automatic security updates.

### 10.1.6. Helpful resources

- Ubuntu IoT pricing[18]
- Security certifications[19]
- Canonical Livepatch[20]
- Embedded Kubernetes for secure IoT Edge[21]
- Ubuntu certified IoT hardware[22]
- IoT and devices pricing[23]
- Ubuntu Core[24]

[15] https://ubuntu.com/core/features/full-disk-encryption
[16] https://microk8s.io/
[17] https://ubuntu.com/kubernetes
[18] https://ubuntu.com/pricing/devices
[19] https://ubuntu.com/security/certifications
[20] https://ubuntu.com/livepatch
[21] https://ubuntu.com/engage/embedded-kubernetes-for-secure-iot-edge-webinar
[22] https://certification.ubuntu.com/iot
[23] https://ubuntu.com/pricing/devices
[24] https://ubuntu.com/core

# 11. Technical support

Technical support enables access to Canonical Engineering and TechOps teams to troubleshoot and resolve issues. Canonical can provide technical support for IoT projects that are built on pre-certified[25] or enabled[26] hardware.

## 11.1. Process

Canonical provides you with access to the support portal[27]. The support portal is the preferred channel for opening support cases. For emergency cases, customers will also have access to telephone support.

## 11.2. Response time and severity levels

Canonical provides a workaround or permanent solution as soon as possible, balanced against higher severity level cases. If a workaround is provided, Canonical's support engineers continue to work on developing a permanent resolution to the case.

The target initial response time for severity levels is 12 business hours, defined as Monday - Friday between 8:00 and 18:00 excluding public holidays.

## 11.3. Customer involvement

Your involvement facilitates the delivery of technical support. The following contributions are expected:

- Resolving all issues your end-users face.
- Specifying how an issue arises and in what sub-system it is taking place.
- Providing a repeatable test case.
- Provide any debugging or further testing required.
- Provide technical information as requested to resolve the problem.
- Verifying issue resolution, once the final solution has been provided by Canonical.

## 11.4. Helpful resources

- Canonical's support portal[28]

[25] https://ubuntu.com/smart-start/guide/hardware-setup
[26] https://ubuntu.com/smart-start/guide/device-enablement
[27] http://support.canonical.com/
[28] http://support.canonical.com/

# 12. Training workshops

Canonical transfers knowledge through training and consulting services. Training workshops empower OEMs to create IoT solutions with Ubuntu Core. Workshops are adapted to customers' in-house capabilities.

The workshops blend conceptual discussion with hands-on labs. They may be delivered on-site or remotely. Following is a list of training and consulting services available to customers.

## 12.1. Introduction to Ubuntu Core

This workshop dives into Ubuntu Core and sheds light on the OS architecture. Canonical engineers demonstrate the effective use of Ubuntu Core. We show you how to create images customized to work on any board. We walk you through the disk layout, device recovery and update mechanisms. And we also showcase advanced capabilities like bootloader integration, secure boot and full disk encryption.

## 12.2. Snapcraft 101

A hands-on workshop to learn how to create, build, publish and maintain your own snaps using Snapcraft. The workshop alternates short instructor-led discussions with task-focused practical labs. At the end of the two day workshop, your development team will be fully capable of deploying and maintaining your own snaps.

## 12.3. Project consulting

On-site or remote engagement with Canonical experts in board and SoC enablement, snap development or Ubuntu management.

## 12.4. Smart device strategy – executive workshop

What's your IoT strategy? Whether you already ship Linux-based devices or are just thinking about entering the IoT market, we help you understand the challenges and opportunities for a successful go to market strategy. This executive workshop covers devices, edge clusters, cloud operations and strategy. Learn how to operate at high speed and precision for your all-digital business models.

## 12.5. Helpful resources

- Webinar: A technical introduction to Ubuntu Core 20[29]
- Ubuntu IoT pricing[30]

[29] https://ubuntu.com/engage/technical-intro-to-ubuntu-core-20
[30] https://ubuntu.com/pricing/devices