

Configuration Management

55-70 minutes

Fieldnotes

A collection of guides about creating Salt formulas in Qubes OS and distributing them as RPM packages to take advantage of the secure updates mechanism for *dom0*.

Contents

- [Configuration Management](#)
 - [Introduction](#)
 - [Automating qubes configuration with Salt states](#)
 - [Using Salt in Qubes OS](#)
 - [The qubesctl command](#)
 - [How to enable the Salt user directories in Qubes OS](#)
 - [Tutorial: create a split-SSH Salt state for Qubes OS](#)
 - [Overview](#)
 - [Identify the qubes involved](#)
 - [Create state files to ensure the qubes presence](#)
 - [Identify which packages need to be installed](#)
 - [Create state files to ensure the required packages are installed](#)
 - [Create top files to group related state files](#)
 - [Identify the configuration files needed for each qube](#)
 - [Create state files to manage configuration files in dom0](#)
 - [Create state files to manage configuration files in AppVMs](#)
 - [Apply the split-SSH state anytime](#)
 - [Use split-SSH as usual](#)
 - [Recap and takeaways](#)
 - [How to design a Salt state for Qubes OS](#)
 - [Configurable Salt formulas for Qubes OS](#)
 - [RPM Packaging](#)
 - [How to create an RPM package](#)
 - [How to publish RPM packages](#)
 - [Overview](#)
 - [Create a RPM repository](#)

- [Add packages to the RPM repository](#)
- [Test the RPM repository locally](#)
- [Publish the RPM repository](#)
- [How to add an RPM package repository to dom0](#)
- [Threat model](#)
 - [Assets](#)
 - [Assumptions](#)

Introduction¶

Using [Qubes OS](#) involves managing different virtual machines (*qubes*) for different purposes. While some activities can be performed using the programs installed by default in the operating system of a given qube, some activities require the qube to be specifically configured.

It is perfectly possible to perform that configuration manually. In theory, persistent qubes shouldn't need to be configured more than once. However, in practice, it can be convenient to record and automate some of the steps required to configure your qubes. That is the purpose of *configuration management software*. Conveniently, Qubes OS uses [Salt](#) internally and supports user-defined configuration.

The following guides present examples of how Salt can be used in Qubes OS, and demonstrate workflows that can be used to perform configuration management in Qubes OS over time. As such, the guides aim at answering the following questions:

- How can I write a Salt state specifically for Qubes OS?
- How can I organize a Salt formula to allow different people to use it in different ways?
- How can I develop my Salt formulas in a less-trusted qube, and mitigate risks when installing them in *dom0*?

Automating qubes configuration with Salt states¶

Salt can be used to manage the configuration of custom *qubes* in Qubes OS. While the most common uses of Salt are extensively documented, using Salt in Qubes OS is a little different.

This guide provides a practical introduction to the use of Salt in Qubes OS and demonstrates a reasonable workflow to create custom Salt states. It also highlights the specificities of the Qubes OS setup, so that you can use the traditional Salt documentation to inform your use of Salt in Qubes OS.

Using Salt in Qubes OS¶

The `qubesctl` command¶

When using Salt outside of Qubes OS, the `salt` command is used to enable top files or apply states. Qubes OS defines its own command `qubesctl` for that purpose. Whenever the [official Salt documentation](#) uses the `salt` command, you can use the `qubesctl` command in Qubes OS.

How to enable the Salt user directories in Qubes OS¶

Qubes OS uses Salt internally for configuration management. The Salt files are stored in several directories, mainly `/srv/salt` and `/srv/pillar`. The *user directories* `/srv/user_salt` and `/srv/user_pillar` can be used to store user-defined configuration.

Besides making management easier, making use of the user directories prevents user-defined configuration

from being modified by accident during system updates.

The user directories are disabled by default, and can be enabled using Salt. 🤖

1. Install **qubes-mgmt-salt-base-config** in *dom0* if necessary:

```
# dom0

sudo dnf install qubes-mgmt-salt-base-config
```

2. Enable the `qubes.user-dirs` top file:

```
# dom0

sudo qubesctl top.enable qubes.users-dirs
```

As long as the top file is enabled, applying the state will ensure that the user directories exist.

3. Verify that the top file was enabled:

```
# dom0

sudo qubesctl top.enabled

# local: ①
# -----
# base:
#   - /srv/salt/_tops/base/qubes.user-dirs.top ②
#   - /srv/salt/_tops/base/topd.top
```

- ① In the command output, `local` refers to *dom0* because we run `qubesctl` in *dom0*.
- ② This is the line that indicates that the `qubes.user-dirs` top file is enabled.

4. Ensure the user directories exist by applying the state:

```
# dom0

sudo qubesctl state.apply # the implicit target is dom0
```

If for any reason you want to disable the `qubes.user-dirs` top file, you can do it as follows:

```
# dom0

sudo qubesctl top.disable qubes.user-dirs
```

Tutorial: create a split-SSH Salt state for Qubes OS¶

Overview¶

This tutorial explains how to write a Salt state to configure split-SSH. Some familiarity with the split-SSH feature in Qubes OS is useful, but some context will be provided at every step.

The instructions to configure split-SSH in this tutorial are based on [Using split ssh in QubesOS 4.0](#) by Kushal Das.

Split-SSH in a nutshell

In a conventional SSH setup, a qube would use its own SSH agent to retrieve the private key and perform authentication when connecting to a remote server. Because that qube is connected to the network, that same network connection could be used to exfiltrate the private key.

Qubes OS makes it possible for the SSH private keys and the SSH agent from one qube to be used by another qube under the supervision of *dom0*. Such a setup is typically referred to as *split-SSH* because the SSH operation is split across two qubes. How does that work?

We'll call **ssh-vault** the qube that stores the keys and performs every authentication operation. That qube has no network access, the keys never leave it, and *dom0* supervises every access to its SSH agent. Ideally, to further limit the attack surface, **ssh-vault** has no other responsibility than storing keys and performing authentication.

We'll call **ssh-client** the qube that is attempting to connect to a remote machine. When the `ssh` program is used in the **ssh-client**, a request is performed to delegate the authentication operations to the SSH agent of the **ssh-vault**.

That request is evaluated by *dom0* and access is granted or denied based on a pre-defined policy. If the request is denied, the **ssh-client** never got access to the **ssh-vault**. If the request is granted, the **ssh-client** can be used to authenticate with the remote machine without ever having seen the private keys.

Identify the qubes involved¶

Three qubes are involved in a typical split-SSH scenario:

1. The *vault* needs to be able to provide an SSH agent
2. A *client* needs to be configured to request access to the SSH agent of the vault
3. A *policy* needs to be defined in *dom0* to supervise access to the vault

Setting up split-SSH requires to ensure that these three elements exist, and we'll first make sure the vault and client qubes exist. In Qubes OS, *dom0* is a special case because it always exists. The role of our split-SSH state will only be to create a policy in *dom0*.

Create a directory for our Salt state and a directory for each of the qubes involved:

```
# dom0

sudo mkdir -p /srv/user_salt/split-ssh # ①

cd /srv/user_salt
sudo mkdir split-ssh/client split-ssh/policy split-ssh/vault

sudo tree split-ssh
#
# split-ssh
# └─ client
# └─ policy
# └─ vault
```

- ① We create our Salt state in `/srv/user_salt` because it is one of the Salt [user directories](#). This ensures our state is not modified by Qubes OS updates.

Create state files to ensure the qubes presence¶

Create two state files to ensure that both the split-SSH vault and client exist:

```
# dom0

sudo touch client/vm.sls vault/vm.sls

sudo tree split-ssh
#
# split-ssh
# └─ client
#     └─ vm.sls ★
# └─ policy
# └─ vault
#     └─ vm.sls ★
```

Write the content of the state files [1](#):

```
# split-ssh/vault/vm.sls

ssh-vault-present: # ①
  qvm.present: # ②
    - name: ssh-vault # ③
    - template: fedora-32 # ④
    - label: black
    - mem: 400
    - vcpus: 2

ssh-vault-has-no-network-access:
  qvm.prefs: # ⑤
    - name: ssh-vault
    - netvm: none # ⑥
    - default_dispvm: # ⑦

ssh-vault-autostarts:
  qvm.prefs:
    - name: ssh-vault
    - autostart: True # ⑧
```

- ① The ID of the state must be unique and tell what the state is about. This state ensures that `ssh-vault` is present.
- ② This state relies on the Qubes OS [qvm state](#) to ensure that a qube is present. If needed, the qube will be created using the given parameters.
- ③ The name of the qube, if a qube with that name is already present it won't be modified.
- ④ It is important that our qubes are based on a Fedora template for this tutorial.
- ⑤ The `qvm.prefs` function of the Qubes OS [qvm state](#) allows to modify the settings of a qube.
- ⑥ No `netvm` prevents any network access.
- ⑦ Ensuring that no `default_dispvm` is set prevents a security warning in the Qube Manager (as well as accidental network access via disposable VMs).
- ⑧ Starting the `ssh-vault` automatically when Qubes OS starts is optional, but I find it convenient.

```
# split-ssh/client/vm.sls

ssh-client-present: # ①
  qvm.present:
```

```
- name: ssh-client # ②
- template: fedora-32
- label: blue
- mem: 400
- vcpus: 2
```

- ① This state ensures that `ssh-client` is present. This ID will be shown in the logs when applying the state, make sure to choose one that makes sense on its own!
- ② This state is very similar to the one that ensures the vault presence. Don't forget that name of the qube is different 😊


And finally apply both states 1:

```
# dom0

sudo qubesctl state.apply split-ssh.client.vm,split-ssh.vault.vm saltenv=user
#                               ①                               ②                               ③

# local:
# -----
#           ID: ssh-client-present
#   Function: qvm.present
#     Name: ssh-client
#   Result: True                               ④
#   Comment: /usr/bin/qvm-create ssh-client --class=AppVM --
template=fedora-32 --label=blue --property=memory=400 --property=vcpus=2 None ⑤
#   Started: 16:00:00.000000
#   Duration: 6700.000 ms
#   Changes:
# -----
#           ID: ssh-vault-present
#   Function: qvm.present
#     Name: ssh-vault
#   Result: True                               ④
#   Comment: [SKIP] A VM with the name 'ssh-vault' already exists. None ⑥
#   Started: 16:00:06.700000
#   Duration: 600.000 ms
#   Changes:
```

- ① This is the same command we used in the introduction to [using Salt in Qubes OS](#). The implicit target is `dom0`.
- ② When referring to Salt states in Salt commands or files, the directories are separated by dots and the extension is omitted. This parameter corresponds to `split-ssh/client/vm.sls`.
- ③ When applying a single state like we're doing here, it is necessary to specify the *environment* user in order to instruct Salt to look it up in the user directories (`/srv/user_salt`).
- ④ The result is `True` when the state was applied successfully. In this case it means that the qube exists.
- ⑤ The qube `ssh-client` didn't exist before I applied the state. The command that Salt ran is written in the comment. That command was composed by the `qvm.present` function. Note that `None` is not part of the command, it represents the command output.
- ⑥ The qube `ssh-vault` already existed when I applied the state. When the qube already exists, `qvm.present` prints an informational statement and exits successfully.

 **Congratulations!** You've configured your first qubes with Salt! From now on, you can apply those two states as often as you'd like to ensure the qubes exist and have Salt create them automatically if they don't.

1(1,2,3)

Thank you to @0fe6db552f62d773 in the Qubes OS forum for reporting typos!

Identify which packages need to be installed¶

One of the features of Qubes OS is that few changes persist after a qube reboot. Each qube (also known as *AppVM*) is based on a *TemplateVM*. Because it is the default in the current Qubes OS release (R4.0), we'll assume all three qubes are based on the *fedora-32* template.

Every time a qube starts, most of its file system is copied fresh from its *template*. For that reason, when a package is installed in a qube, it isn't available anymore after the qube is rebooted. However, for that same reason, packages that are installed in the qube's template become available as soon as the qube is rebooted.

Since `ssh` is available in Fedora by default, it will be available in both **ssh-vault** and **ssh-client**. For reasons that will become clear later, a program called `ncat` must be available as well in both **ssh-vault** and **ssh-client**.

Create state files to ensure the required packages are installed¶

Let's write a Salt state file that ensures `ncat` is available.

Salt encourages writing re-usable states. Because it will be used by the client and the vault, let's save this state file in its own `split-ssh/packages` directory:

```
# dom0

sudo mkdir split-ssh/packages

sudo touch split-ssh/packages/ncat.sls

sudo tree split-ssh
#
# split-ssh
# └─ client
#     └─ vm.sls
# └─ packages ★
#     └─ ncat.sls ★
# └─ policy
# └─ vault
#     └─ vm.sls
```

```
# split-ssh/packages/ncat.sls

ncat: # ①
  pkg.installed: # ②
    - name: nmap-ncat # ③
```

- ① The ID of the state must be unique and tell what the state is about. The name of the command seems appropriate.
- ② This state relies on the [pkg state](#) to ensure that a package is installed. If needed, the package will be installed.
- ③ The name of the package that provides `ncat` in Fedora is **nmap-ncat**.

Of course, it would be nice if all the states related to the vault and the client were contained in their respective directories. Happily, Salt allows to re-use existing states to create new states. First create two new state files that will specify which packages are needed in the vault and in the client:

```
# dom0

sudo touch split-ssh/client/packages.sls split-ssh/vault/packages.sls

sudo tree split-ssh
#
# split-ssh
# └─ client
#     └─ packages.sls ★
#     └─ vm.sls
# └─ packages
#     └─ ncat.sls
# └─ policy
# └─ vault
#     └─ packages.sls ★
#     └─ vm.sls
```

Now let's include the state we already wrote in both these files:

```
# split-ssh/client/packages.sls

include: # ①
- split-ssh.packages.ncat # ②
```

```
# split-ssh/vault/packages.sls

include:
- split-ssh.packages.ncat
```

- ① The include statement accepts a list of states that will all be added to the file as if they were copied. This allows to avoid unnecessary repetition while keeping the states organised.
- ② Note how the state reference is the same we'd use in the Salt commands.

If we were to apply these states in *dom0*, the package would be installed in *dom0* and that is not what we want. That is why we'll instruct Salt to skip *dom0* and target the **fedora-32** template.

```
# dom0

sudo qubesctl --skip-dom0 --target=fedora-32 state.apply split-ssh.client.packages saltenv=user
#                               ①                               ②                               ③
④

# Fedora-32: OK ⑤

less /var/log/qubes/mgmt-fedora-32.log # ⑥
# press Shift+G to jump to the end of the file, press Q to quit
#
# [...]
# 2020-12-16 00:13:23,195 calling 'state.apply split-ssh.client.packages saltenv=user'... ⑦
# 2020-12-16 00:15:03,399 output: fedora-32:
# 2020-12-16 00:15:03,400 output: -----
# 2020-12-16 00:15:03,400 output: ID: ncat ⑧
# 2020-12-16 00:15:03,400 output: Function: pkg.installed
# 2020-12-16 00:15:03,400 output: Name: nmap-ncat
```



```

# 2020-12-16 00:15:03,400 output:      Result: True          ⑨
# 2020-12-16 00:15:03,400 output:      Comment: The following packages were
installed/updated: nmap-ncat
# 2020-12-16 00:15:03,400 output:      Started: 00:14:01.929264
# 2020-12-16 00:15:03,400 output:      Duration: 61565.657 ms
# 2020-12-16 00:15:03,400 output:      Changes:
# 2020-12-16 00:15:03,400 output:      -----
# 2020-12-16 00:15:03,401 output:      nmap-ncat:
# 2020-12-16 00:15:03,401 output:      -----
# 2020-12-16 00:15:03,401 output:      new:
# 2020-12-16 00:15:03,401 output:      2:7.80-4.fc32
# 2020-12-16 00:15:03,401 output:      old:
# 2020-12-16 00:15:03,401 output:      ⑩

```

- ① Skipping *dom0* to make sure we don't install the packages in *dom0*.
- ② Targeting *fedora-32* so that the packages are installed in the template.
- ③ When targeting a qubes that is not *dom0*, only one state can be applied at a time.
- ④ Reminder: when applying a specific state from the user directories, the Salt environment must be specified.
- ⑤ The command executed successfully in *fedora-32*. For security reasons, *dom0* doesn't print the logs from other qubes.
- ⑥ A management qube was started to apply the state to *fedora-32*, and its logs were appended to a file in *dom0*.
- ⑦ This is the command that was executed in *fedora-32*. At that stage, the targeting already happened.
- ⑧ The ID of our state.
- ⑨ Success!
- ⑩ The package wasn't installed before, so there was no "old" version number.

```

sudo qubesctl --skip-dom0 --target=fedora-32 state.apply split-
ssh.vault.packages saltenv=user          ①

# Fedora-32: OK

less /var/log/qubes/mgmt-fedora-32.log
# press Shift+G to jump to the end of the file, press Q to quit
#
# [...]
# 2021-03-21 18:22:10,318 calling 'state.apply split-ssh.vault.packages
saltenv=user'...
# 2021-03-21 18:23:32,486 output: fedora-32:
# 2021-03-21 18:23:32,487 output: -----
# 2021-03-21 18:23:32,488 output:      ID: ncat
# 2021-03-21 18:23:32,488 output:      Function: pkg.installed
# 2021-03-21 18:23:32,489 output:      Name: nmap-ncat
# 2021-03-21 18:23:32,489 output:      Result: True          ②
# 2021-03-21 18:23:32,489 output:      Comment: All specified packages are
already installed
# 2021-03-21 18:23:32,490 output:      Started: 18:23:27.400582
# 2021-03-21 18:23:32,490 output:      Duration: 2789.409 ms
# 2021-03-21 18:23:32,490 output:      Changes:          ③
# 2021-03-21 18:23:32,491 output:

```

```
# 2021-03-21 18:23:32,491 output: Summary for fedora-32
# 2021-03-21 18:23:32,491 output: -----
# 2021-03-21 18:23:32,492 output: Succeeded: 1
# 2021-03-21 18:23:32,492 output: Failed:    0
# 2021-03-21 18:23:32,492 output: -----
# 2021-03-21 18:23:32,492 output: Total states run:    1
# 2021-03-21 18:23:32,493 output: Total run time:    2.789 s
# 2021-03-21 18:23:32,493 exit code: 0 ④
```

- ① Applying the vault state this time.
- ② Success!
- ③ Because the package was already installed, there were no changes this time.
- ④ Salt exited successfully in fedora-32, that is why Salt in *dom0* printed Fedora-32: OK.

You made it! ❤️ That was a lot to go through, but you know now how to apply Salt states to qubes other than *dom0*.

You probably are also starting to realize that applying states one by one requires many different commands, and accurate targeting requires a lot of attention. Good news is that Salt provides a way to make that simpler and less error-prone: it's time to write some *top files*.

Identify the configuration files needed for each qube¶

The **vault** requires:

- A *remote procedure call* (RPC) definition to allow access to the SSH agent from other qubes. Like all other RPC definitions, it should be located in */etc/qubes-rpc/*.
- A [desktop entry](#) for *ssh-add*. That file should be placed in */home/user/.config/* *autostart* to ensure that *ssh-add* starts when the qube starts.

A *policy* that supervises access to the vault must be defined in **dom0**. This policy will specifically supervise the usage of the vault's SSH Agent RPC. Like all other RPC policies, it should be located in */etc/qubes-rpc/* *policy*.

The **client** requires a Unix socket to be available to send data to the vault when using the SSH Agent RPC. That requires managing two files: one to create the socket and another to ensure it is discoverable when needed.

Create state files to manage configuration files in dom0¶

Create a state file to manage the RPC policy:

```
# dom0

sudo touch split-ssh/policy/init.sls # ①

sudo tree split-ssh
#
# split-ssh
# └─ client
#   └─ packages.sls
#   └─ vm.sls
# └─ client.top
# └─ packages
#   └─ ncat.sls
# └─ policy
#   └─ init.sls ★
# └─ policy.top
```

```
# ┌ vault
# │ ┌ packages.sls
# │ └ vm.sls
# └ vault.top
```

- ① Since we'll only use one state file to configure the policy, we can call it `init.sls`. This allows to refer to it in Salt by `split-ssh.policy`.

Add that state file to the policy top file:

```
# split-ssh/policy.top

user:
  dom0:
    - split-ssh.policy # ①
```

- ① This refers to `split-ssh/policy/init.sls`. The list of states available to `dom0` is no longer empty!

Write the content of the state file 1:

```
# split-ssh/policy/init.sls

/etc/qubes-rpc/policy/qubes.SSHAgent: # ①
  file.managed: # ②
    - user: root # ③
    - group: root
    - mode: '0755'
    - makedirs: True ④
    - source: salt://split-ssh/policy/files/qubes.SSHAgent # ⑤
```

- ① When managing a file, the ID must be the full path of the file.
- ② This state relies on the [file state](#) to manage the presence and content of the RPC policy.
- ③ The ownership and permissions can be managed as well.
- ④ Ensures that directories are created if necessary.
- ⑤ The source that will be used to determine the contents of the file. The prefix `salt://` allows to look up the file in the Salt directories. This address corresponds to `/srv/user_salt/split-ssh/policy/files/qubes.SSHAgent`.

Create the source file for the policy:

```
# dom0

sudo mkdir split-ssh/policy/files # ①
sudo touch split-ssh/policy/files/qubes.SSHAgent # ②

sudo tree split-ssh
#
# split-ssh
# ┌ client
# │ ┌ packages.sls
# │ └ vm.sls
# └ client.top
# ┌ packages
# │ └ ncat.sls
# └ policy
```

```
# ┌── files ★
# │   └── qubes.SSHAgent ★
# │   init.sls
# └── policy.top
# └── vault
#     ├── packages.sls
#     └── vm.sls
# └── vault.top
```

- ① Create a directory to keep the source files organised.
- ② The name of the RPC policy files usually starts with `qubes.`, followed by the name of the program being called. In our case, it is the vault's SSH agent that is called, so `qubes.SSHAgent` seems appropriate.

Write the content of the policy file:

```
# split-ssh/policy/files/qubes.SSHAgent

ssh-client ssh-vault ask
# ①          ②          ③
```

- ① The first column in a [Qubes OS RPC policy](#) is the *source* of the remote procedure call.
- ② The second column is the *destination*.
- ③ The third column is the *permission*. In this case, we want `dom0` to ask if access to the vault should be granted or denied.

Apply the state!

```
# dom0

sudo qubesctl state.apply # ①
#
# local:
# -----
# [Skipping some output for brevity.]
# -----
# ID: /etc/qubes-rpc/policy/qubes.SSHAgent # ②
# Function: file.managed
# Result: True # ③
# Comment: File /etc/qubes-rpc/policy/qubes.SSHAgent updated
# Started: 17:40:47.401244
# Duration: 81.815 ms
# Changes:
# -----
# diff:
# New file # ④
# group:
# root
# mode:
# 0755

cat /etc/qubes-rpc/policy/qubes.SSHAgent # ⑤
# ssh-client ssh-vault ask
```

- ① As usual, `dom0` is the implicit target.

- ② The ID of the state that managed the RPC policy.
- ③ Success!
- ④ The file was created.
- ⑤ The content of the file matches the content of `split-ssh/policy/files/qubes.SSHAgent`.

As long as the `split-ssh.policy` top file is enabled, Salt will ensure the split-SSH policy file exists and matches its source file when the state is applied. This is enough to manage files in `dom0`.

Create state files to manage configuration files in AppVMs¶

Both `ssh-client` and `ssh-vault` are AppVMs (also known as *template-based qubes*, I'll keep writing qube and qubes to keep it short).

When [identifying the required packages](#), we noted that most files don't persist after a qube reboots. One way to ensure a file is always present in a qube is to create the file in the qube's template. We could create all the configuration files in the qubes' template, but that would make them available in *any* qube based on that template. We can do better.

Two parts of a qube file system persist across reboots: the user directory (`/home/user`) and the `/rw` directory. In order to ensure that a file is always available in a qube, we will first ensure the file exists in `/rw/config/split-ssh`, then ensure that is automatically copied to its intended location every time the qube starts.

Conveniently, all qubes contain a script that runs every time the qube starts: `/rw/config/rc.local`. Because that script is located in the `/rw` directory, any changes we make to it will persist across reboots.

With that in mind, we will use Salt to manage our configuration files in `/rw/config/split-ssh`, and also append code snippets to `/rw/config/rc.local` in order to ensure that each configuration file is automatically copied to its intended location when the qube starts.

Create state files to manage the vault configuration files¶

Reminder: the vault requires creating [two configuration files](#). None of those files exist in the qube by default.

Create the source files for both of them:

```
# dom0

sudo mkdir split-ssh/vault/files # ①
sudo touch split-ssh/vault/files/qubes.SSHAgent # ②
sudo touch split-ssh/vault/files/ssh-add.desktop # ③
```

- ① Create a directory to keep the source files organised.
- ② Like the name of the RPC policy files, the name of the RPC files themselves usually starts with `qubes.`, followed by the name of the program being called. In our case, it is the vault's SSH agent that is called, so `qubes.SSHAgent` seems appropriate.
- ③ The name of the desktop files is usually the name of the program being described, `ssh-add` in this case, followed by the `.desktop` extension.

Then create source files for the two corresponding code snippets. Those will be appended to `/rw/config/rc.local` and contain instructions to copy the RPC definition and the desktop file to the location where we need them to be after the qube starts.

```
# dom0

sudo mkdir split-ssh/vault/files/rc.local.d # ①
sudo touch split-ssh/vault/files/rc.local.d/config # ②
```

```
sudo touch split-ssh/vault/files/rc.local.d/rpc # ③
```

- ① When the content of multiple file snippets is appended to a configuration file, it is conventional to place the snippets in a directory named after the configuration file. The extension `.d` represents the “directory”. In this case, the contents of the `rc.local.d` directory will be used to extend the `rc.local` file.
- ② The first snippet will ensure that that `ssh-add` is configured to start automatically when the qube starts.
- ③ The second snippet will ensure the RPC definition is available in the vault.

Finally, create two state files to ensure the source files are present in the qube and the code snippets are appended to the qube’s `rc.local` script.

```
# dom0

sudo touch split-ssh/vault/config.sls # ①
sudo touch split-ssh/vault/rpc.sls # ②

sudo tree split-ssh
#
# split-ssh
# └─ client
#     └─ packages.sls
#     └─ vm.sls
# └─ client.top
# └─ packages
#     └─ ncat.sls
# └─ policy
#     └─ files
#         └─ qubes.SSHAgent
#     └─ init.sls
# └─ policy.top
# └─ vault
#     └─ files ★
#         └─ rc.local.d ★
#             └─ config ★
#             └─ rpc ★
#         └─ qubes.SSHAgent ★
#         └─ ssh-add.desktop ★
#     └─ config.sls ★
#     └─ packages.sls
#     └─ rpc.sls ★
#     └─ vm.sls
# └─ vault.top
```

- ① Will ensure that the desktop file is present in the qube and that `rc.local` is extended to ensure `ssh-add` starts automatically.
- ② Will ensure that the RPC definition is present in the qube and that `rc.local` is extended to ensure it is located in the `/etc/qubes-rpc/` directory.

Write first the content of the files needed to ensure `ssh-add` is configured:

```
# split-ssh/vault/config.sls

/rw/config/split-ssh/ssh-add.desktop: # ①
file.managed: # ②
- user: root
- group: root
- mode: '0755'
```

```

- makedirs: True # ③
- source: salt://split-ssh/vault/files/ssh-add.desktop # ④

ssh-agent-autostart-present: # ⑤
file.append: # ⑥
  - name: /rw/config/rc.local ⑦
  - source: salt://split-ssh/vault/files/rc.local.d/config # ⑧

```

- ① This state ensures that the file `/rw/config/split-ssh/ssh-add.desktop` exists in the targeted qube.
- ② It relies on the [file state](#) to ensure that a file is present. If needed, the file will be copied from the specified source.
- ③ While `/rw/config` likely exists in the qube, the `split-ssh` directory doesn't. Here we ensure Salt creates any missing directory.
- ④ Like we did before with the RPC policy, our salt state contains a source file that can be copied to the targeted qube if necessary.
- ⑤ This state ensures the `ssh-add` autostart configuration is present in the qube, so that the SSH agent is always available once the qube has started.
- ⑥ The `file.append` function ensures the presence of specific lines in a file.
- ⑦ ⑧ If the content of the source file is not already present in `/rw/config/rc.local`, then it will be added to the end of the file.

```
# split-ssh/vault/files/add-ssh.desktop
```

```

[Desktop Entry] # ①
Name=ssh-add
Exec=ssh-add # ②
Type=Application

```

- ① The format of the [desktop entry](#) files is specified by the *freedesktop.org* organisation.
- ② For autostart purposes, the essential information is what command should be executed. Executing `ssh-add` will start the SSH agent.

```
# split-ssh/vault/files/rc.local.d/config
```

```

# ①
# Ensure ssh-add is started automatically
#
# See /srv/user_salt/split-ssh in dom0 for details.
rm -f ~user/.config/autostart/ssh-add.desktop # ②
mkdir -p ~user/.config/autostart/ # ③
ln -s /rw/config/split-ssh/ssh-add.desktop ~user/.config/autostart/ssh-
add.desktop
# ④ ⑤ ⑥

```

- ① The content of this file will eventually be copied to the `rc.local` script of the targeted qube. Because you may want to inspect that script at a later time, it is convenient to include a comment explaining why this snippet of code was added.
- ② In order to start from a clean state, let's remove the autostart script if it exists. The `-f` option prevents the command from failing if the file doesn't exist.
- ③ Ensure the autostart directory exists before adding anything to it.

- ④ Create a symbolic link avoids unnecessary duplication.
- ⑤ The symbolic link will reference the desktop file that is present in the targeted qube `/rw/config` directory. Remember that directory persists across reboots.
- ⑥ By placing the symbolic link in the `~user/.config/autostart` directory, we ensure the SSH agent is started automatically when the qube starts.

Then write the content of the files needed to ensure the RPC definition is present:

```
# split-ssh/vault/rpc.sls

/rw/config/split-ssh/qubes.SSHAgent: # ①
  file.managed:
    - user: root
    - group: root
    - mode: '0755'
    - makedirs: True
    - source: salt://split-ssh/vault/files/qubes.SSHAgent # ②

rpc-present: # ③
  file.append: # ④
    - name: /rw/config/rc.local # ⑤
    - source: salt://split-ssh/vault/files/rc.local.d/rpc # ⑥
```

- ① This state ensures the presence of the `/rw/config/split-ssh/qubes.SSHAgent` file in the targeted qube.
- ② If necessary, the file is copied from the source file that is stored in `dom0` as part of this Salt state.
- ③ This state ensures the RPC definition is available in the targeted qube.
- ④ Reminder: the `file.append` function ensures the presence of specific lines in a given file.
- ⑤ ⑥ Ensures the presence of a snippet of code in the `rc.local` script, so that the SSH Agent RPC is enabled every time the qube starts.

```
# split-ssh/vault/files/qubes.SSHAgent

#!/bin/sh # ①
notify-send "[`qubesdb-read /name`] SSH agent access from:
$QREXEC_REMOTE_DOMAIN"
# ② ③ ④
ncat -U $SSH_AUTH_SOCK
# ⑤ ⑥ ⑦
```

- ① In Qubes OS, the RPC definitions are shell scripts that are executed if the corresponding RPC policy allows it.
- ② `notify-send` prints a notification to inform the user of important events.
- ③ The `qubesdb-read /name` command prints the name of the qube where it is executed. In our case, it will be `ssh-vault`.
- ④ The `$QREXEC_REMOTE_DOMAIN` variable is managed by Qubes OS when a RPC is used. In our case, the remote domain will be `ssh-client`. Note that “domain” in this case refers to a qube, not a Qubes OS domain because `qrexec` is executed in the context of Xen. Xen is the hypervisor used by Qubes OS and it refers to virtual machines as domains.
- ⑤ This is why `ncat` needed to be available in the vault!

- ⑥ `ncat -U` allows to read and write data using Unix sockets. In this case the socket is used to read and write data between `ssh-client` and `ssh-vault` in the Qubes OS.
- ⑦ The variable `SSH_AUTH_SOCK` is managed by Qubes OS. Its value is the name of a Unix socket that was created for the purpose of SSH authentication.

```
# split-ssh/vault/files/rc.local.d/rpc

# ①
# Add the split-SSH RPC
#
# See /srv/salt/split-ssh in dom0 for details.
rm -f /etc/qubes-rpc/qubes.SSHAgent # ②
ln -s /rw/config/split-ssh/qubes.SSHAgent /etc/qubes-rpc/qubes.SSHAgent
# ③          ④          ⑤
```

- ① Like before, adding some information about why the code was added makes the `rc.local` script easier to read on its own.
- ② Start clean.
- ③ Create a symbolic link to avoid unnecessary duplication.
- ④ The `qubes.SSHAgent` definition is stored in the qube's persistent directories.
- ⑤ For a *remote procedure call* to be available, its definition must be placed in the `/etc/qubes-rpc` directory. Unlike `/rw/config`, changes to the `/etc/qubes-rpc` directory are lost when the qube reboots. This snippet of code will run every time the qube starts and ensure the SSH Agent RPC is available.

Finally, add both state files to the vault top file:

```
# split-ssh/vault.top

user:
  dom0:
    - split-ssh.vault.vm

fedora-32:
  - split-ssh.vault.packages

ssh-vault: # ① ★
  - split-ssh.vault.config # ② ★
  - split-ssh.vault.rpc    # ③ ★
```

- ① These two state files must be applied in the `ssh-vault`.
- ② The first one refers to `split-ssh/vault/config.sls`. It ensures the SSH agent is started when the qube starts.
- ③ The second one refers to `split-ssh/vault/rpc.sls`. It ensures the SSH agent of the qube can be accessed from other qubes via a RPC.

Only remains to apply the state:

```
# dom0

sudo qubesctl --skip-dom0 --target=ssh-vault state.apply
#          ①          ②
```

ssh-vault: OK

less /var/log/qubes/mgmt-ssh-vault.log

press Shift+G to jump to the end of the file, press Q to quit

#

[...]

2020-12-17 17:57:23,096 calling 'state.apply'...

2020-12-17 17:58:52,894 output: ssh-vault:

2020-12-17 17:58:52,895 output: -----

2020-12-17 17:58:52,895 output: ID: /rw/config/split-ssh/ssh-add.desktop

2020-12-17 17:58:52,896 output: Function: file.managed

2020-12-17 17:58:52,896 output: Result: True # ③

2020-12-17 17:58:52,896 output: Comment: File /rw/config/split-ssh/ssh-add.desktop updated

2020-12-17 17:58:52,896 output: Started: 17:58:51.533277

2020-12-17 17:58:52,897 output: Duration: 208.409 ms

2020-12-17 17:58:52,897 output: Changes:

2020-12-17 17:58:52,897 output: -----

2020-12-17 17:58:52,897 output: diff:

2020-12-17 17:58:52,898 output: New file

2020-12-17 17:58:52,898 output: mode:

2020-12-17 17:58:52,898 output: 0755

2020-12-17 17:58:52,898 output: -----

2020-12-17 17:58:52,899 output: ID: ssh-agent-autostart-present

2020-12-17 17:58:52,899 output: Function: file.append

2020-12-17 17:58:52,899 output: Name: /rw/config/rc.local

2020-12-17 17:58:52,899 output: Result: True # ④

2020-12-17 17:58:52,899 output: Comment: Appended 7 lines # ⑤

2020-12-17 17:58:52,900 output: Started: 17:58:51.741914

2020-12-17 17:58:52,900 output: Duration: 47.981 ms

2020-12-17 17:58:52,900 output: Changes:

2020-12-17 17:58:52,900 output: -----

2020-12-17 17:58:52,901 output: diff:

2020-12-17 17:58:52,901 output: ---

2020-12-17 17:58:52,901 output:

2020-12-17 17:58:52,901 output: +++

2020-12-17 17:58:52,902 output: @@ -8,3 +8,10 @@

2020-12-17 17:58:52,902 output: # rm -rf /etc/cups

2020-12-17 17:58:52,902 output: # ln -s /rw/config/cups /

2020-12-17 17:58:52,902 output: # systemctl --no-block

2020-12-17 17:58:52,902 output: restart cups # ⑥

2020-12-17 17:58:52,903 output: +

2020-12-17 17:58:52,903 output: +# Ensure ssh-add is started

2020-12-17 17:58:52,903 output: automatically

2020-12-17 17:58:52,903 output: +#

2020-12-17 17:58:52,903 output: +# See /srv/salt/split-ssh

2020-12-17 17:58:52,903 output: in dom0 for details.

2020-12-17 17:58:52,904 output: +rm -f ~user/.config/

2020-12-17 17:58:52,904 output: +mkdir -p ~user/.config/

2020-12-17 17:58:52,904 output: +ln -s /rw/config/split-ssh/

2020-12-17 17:58:52,904 output: ssh-add.desktop ~user/.config/autostart/ssh-add.desktop

2020-12-17 17:58:52,904 output: -----

2020-12-17 17:58:52,904 output: ID: /rw/config/split-ssh/

2020-12-17 17:58:52,904 output: qubes.SSHEnter

2020-12-17 17:58:52,905 output: Function: file.managed


```

# 2020-12-17 17:58:52,905 output:      Result: True      # ⑦
# 2020-12-17 17:58:52,905 output:      Comment: File /rw/config/split-ssh/
qubes.SSHAgent updated
# 2020-12-17 17:58:52,905 output:      Started: 17:58:51.790168
# 2020-12-17 17:58:52,905 output:      Duration: 8.168 ms
# 2020-12-17 17:58:52,905 output:      Changes:
# 2020-12-17 17:58:52,906 output:      -----
# 2020-12-17 17:58:52,906 output:      diff:
# 2020-12-17 17:58:52,906 output:      New file
# 2020-12-17 17:58:52,906 output:      mode:
# 2020-12-17 17:58:52,906 output:      0755
# 2020-12-17 17:58:52,907 output:      -----
# 2020-12-17 17:58:52,907 output:      ID: rpc-present
# 2020-12-17 17:58:52,907 output:      Function: file.append
# 2020-12-17 17:58:52,907 output:      Name: /rw/config/rc.local
# 2020-12-17 17:58:52,907 output:      Result: True
# 2020-12-17 17:58:52,907 output:      Comment: Appended 6 lines
# 2020-12-17 17:58:52,908 output:      Started: 17:58:51.798551
# 2020-12-17 17:58:52,908 output:      Duration: 11.351 ms
# 2020-12-17 17:58:52,908 output:      Changes:
# 2020-12-17 17:58:52,908 output:      -----
# 2020-12-17 17:58:52,908 output:      diff:
# 2020-12-17 17:58:52,908 output:      ---
# 2020-12-17 17:58:52,909 output:      +++
# 2020-12-17 17:58:52,909 output:      @@ -15,3 +15,9 @@
# 2020-12-17 17:58:52,909 output:      rm -f ~user/.config/
autostart/ssh-add.desktop
# 2020-12-17 17:58:52,910 output:      mkdir -p ~user/.config/
autostart/
# 2020-12-17 17:58:52,910 output:      ln -s /rw/config/split-ssh/
ssh-add.desktop ~user/.config/autostart/ssh-add.desktop # ⑧
# 2020-12-17 17:58:52,910 output:      +
# 2020-12-17 17:58:52,910 output:      +# Add the split-SSH RPC
# 2020-12-17 17:58:52,910 output:      +#
# 2020-12-17 17:58:52,910 output:      +# See /srv/salt/split-ssh
in dom0 for details.
# 2020-12-17 17:58:52,911 output:      +rm -f /etc/qubes-rpc/
qubes.SSHAgent
# 2020-12-17 17:58:52,911 output:      +ln -s /rw/config/split-ssh/
qubes.SSHAgent /etc/qubes-rpc/qubes.SSHAgent
# 2020-12-17 17:58:52,911 output:      Summary for ssh-vault
# 2020-12-17 17:58:52,911 output:      -----
# 2020-12-17 17:58:52,911 output:      Succeeded: 4 (changed=4)
# 2020-12-17 17:58:52,912 output:      Failed: 0
# 2020-12-17 17:58:52,912 output:      -----
# 2020-12-17 17:58:52,912 output:      Total states run: 4
# 2020-12-17 17:58:52,912 output:      Total run time: 275.909 ms
# 2020-12-17 17:58:52,912 output:      exit code: 0

```

- ① Since *dom0* is already configured, skipping it saves some time.
- ② Thanks to the top file, only the states that target *ssh-vault* will be applied when the state is applied to *ssh-vault*.
- ③ Success! The desktop file is present in a persistent location of *ssh-vault*. This time it was created by Salt from the source file.

- ④ Success! The snippet of code is present in the `rc.local` script of `ssh-vault`. That means the autostart configuration for `ssh-add` will be set up every time the qube starts.
- ⑤ Because it was originally missing, Salt appended the snippet to `rc.local` in order to ensure its presence.
- ⑥ Note that depending on the content of your vault's `rc.local` script, these context lines may be different. The lines that were appended are identified by the `+` symbol.
- ⑦ Success! The SSH Agent RPC definition is present in the vault.
- ⑧ The snippet of code that will ensure the RPC is active when `ssh-vault` starts is present too. Note that this time, the last few lines of `rc.local` were the ones we appended above!

 You've fully automated the vault configuration. This was the most complex part. In doing so, you learned that it is possible to store persistent configuration in `/rw/config` and that it can be automatically copied to non-persistent location when the qube starts by taking advantage of the qube's `rc.local` script. Automating the client configuration is simpler.

Create state files to manage the client configuration files

Reminder: the client requires [two configuration files](#) to be managed. Both file already exist by default in the qube, so we only need to ensure they contain two snippets of code.

Create the source files for both of them:

```
# dom0

sudo mkdir -p split-ssh/client/files/bashrc.d split-ssh/client/files/rc.local.d
# ①
sudo touch split-ssh/client/files/bashrc.d/sock # ②
sudo touch split-ssh/client/files/rc.local.d/sock # ③
```

- ① Create a directories to keep the source files organised. The contents of the `bashrc.d` directory will be used to extend the user's `.bashrc` file, and the contents of the `rc.local.d` directory will be used to extend the qube's `rc.local` file.
- ② ③ The word “socket” is abbreviated as “sock” in the code that we'll write. Naming these files `sock` keeps the naming consistent.

Finally, create a state file to ensure that both the code snippets are appended to the qube's configuration files.

```
# dom0

sudo touch split-ssh/client/sock.sls # ①

sudo tree split-ssh
#
# split-ssh
# └─ client
#   └─ files ★
#       ├── bashrc.d ★
#       │   └─ sock ★
#       └─ rc.local.d ★
#           └─ sock ★
#   └─ sock.sls ★
#   └─ packages.sls
#   └─ vm.sls
# └─ client.top
# └─ packages
```

```

# ┌─ ncat.sls
# └─ policy
#   └─ files
#       └─ qubes.SSHAgent
#   └─ init.sls
# └─ policy.top
# └─ vault
#   └─ files
#       └─ rc.local.d
#           └─ config
#           └─ rpc
#       └─ qubes.SSHAgent
#       └─ ssh-add.desktop
#   └─ config.sls
#   └─ packages.sls
#   └─ rpc.sls
#   └─ vm.sls
# └─ vault.top

```

- ① This state file will ensure that both `~user/.bashrc` and `rc.local` are extended to ensure a Unix socket is automatically created and discoverable when the qube starts.

Write the content of the state file:

```

# split-ssh/client/sock.sls

ssh-socket-present: # ①
  file.append: # ②
    - name: /rw/config/rc.local ③
    - source: salt://split-ssh/client/files/rc.local.d/sock # ④

ssh-socket-discoverable: # ⑤
  file.append:
    - name: ~user/.bashrc # ⑥
    - source: salt://split-ssh/client/files/bashrc.d/sock # ⑦

```

- ① This state ensures that a socket is created in the targeted qube when `rc.local` is executed.
- ② Reminder: the `file.append` function ensures the presence of specific lines in a given file.
- ③ ④ Ensures the presence of a snippet of code in the `rc.local` script, so that a Unix socket is created every time the qube starts.
- ⑤ This state ensures the socket name is available to `user`, so that they can rely on it to make use of the vault's SSH agent RPC.
- ⑥ ⑦ Ensures the presence of a snippet of code in the `~user/.bashrc` script, so that the socket is discoverable.

Then write the content of the files needed to ensure the socket is present and discoverable:

```

# split-ssh/client/files/bashrc.d/sock

# ①
# Ensure the SSH socket is discoverable
#
# See /srv/user_salt/split-ssh in dom0 for details.
SSH_VAULT_VM="ssh-vault" # ②

if [[ "$SSH_VAULT_VM" != "" ]]; then # ③

```

```

export SSH_AUTH_SOCKET=~user/.SSH_AGENT_$SSH_VAULT_VM # ④
fi

```

- ① The content of this file will eventually be copied to the `~user/.bashrc` script of the targeted qube. Because you may want to inspect that script at a later time, it is convenient to include a comment explaining why this snippet of code was added.
- ② The name of the vault will be used multiple times. Placing it in a variable avoids unnecessary repetition.
- ③ If the name of the vault was left empty, skip exporting the name of the socket.
- ④ Exporting the name of the socket in the `SSH_AUTH_SOCKET` variable will make it available to SSH. When that variable is set, SSH uses it every time access to an SSH agent is needed.

```

# split-ssh/client/files/rc.local.d/socket

# ①
# Create split-SSH socket
#
# See /srv/user_salt/split-ssh in dom0 for details.
SSH_VAULT_VM="ssh-vault" # ②

if [[ "$SSH_VAULT_VM" != "" ]]; then # ③
    export SSH_SOCKET=~user/.SSH_AGENT_$SSH_VAULT_VM # ④
    rm -f "$SSH_SOCKET" # ⑤
    sudo -u user /bin/sh -c "umask 117 && ncat -k -l -U '$SSH_SOCKET' -c
'qrexec-client-vm $SSH_VAULT_VM qubes.SSHAgent' &"
    # ⑥ ⑦ ⑧
fi

```

- ① The content of this file will eventually be copied to the `rc.local` script of the targeted qube. Because you may want to inspect that script at a later time, it is convenient to include a comment explaining why this snippet of code was added.
- ② The name of the vault will be used multiple times. Placing it in a variable avoids unnecessary repetition.
- ③ If the name of the vault was left empty, skip creating the socket.
- ④ The name of the socket will be used multiple times. Placing it in a variable avoids unnecessary repetition. Note that the name of the socket is the same that was set in the `~user/.bashrc` file.
- ⑤ Let's remove the socket if it exists in order to start from a clean state. The `-f` option prevents the command from failing if the socket doesn't exist.
- ⑥ Execute a shell as `user`.
- ⑦ Ensure that when the socket is created, only the user and its group can use it by setting its permissions to `0x770`.
- ⑧ ⑨ This is why `ncat` needed to be available in the client. It will create the socket, and configure it to execute the vault's SSH agent RPC when data becomes available.

Finally, add the state file to the client top file:

```

# split-ssh/client.top

user:
  dom0:
    - split-ssh.vault.vm

fedora-32:

```

```
- split-ssh.vault.packages
```

```
ssh-client: # ① ★
```

```
- split-ssh.client.sock # ② ★
```

- ① The state file must be applied in the ssh-client.
- ② This is a reference to split-ssh/client/sock.sls.

Only remains to apply the state:

```
# dom0
```

```
sudo qubesctl --skip-dom0 --target=ssh-client state.apply
```

```
# ① ②
```

```
# ssh-client: OK
```

```
less /var/log/qubes/mgmt-ssh-client.log
```

```
# press Shift+G to jump to the end of the file, press Q to quit
```

```
#
# [...]
# 2020-12-17 13:34:27,299 calling 'state.apply'...
# 2020-12-17 13:36:10,441 output: ssh-client:
# 2020-12-17 13:36:10,442 output: -----
# 2020-12-17 13:36:10,442 output: ID: ssh-socket-present
# 2020-12-17 13:36:10,442 output: Function: file.append
# 2020-12-17 13:36:10,443 output: Name: /rw/config/rc.local
# 2020-12-17 13:36:10,443 output: Result: True # ③
# 2020-12-17 13:36:10,443 output: Comment: Appended 11 lines # ④
# 2020-12-17 13:36:10,443 output: Started: 00:36:08.427153
# 2020-12-17 13:36:10,444 output: Duration: 376.834 ms
# 2020-12-17 13:36:10,444 output: Changes:
# 2020-12-17 13:36:10,444 output: -----
# 2020-12-17 13:36:10,444 output: diff:
# 2020-12-17 13:36:10,445 output: ---
# 2020-12-17 13:36:10,445 output:
# 2020-12-17 13:36:10,445 output: +++
# 2020-12-17 13:36:10,445 output:
# 2020-12-17 13:36:10,446 output: @@ -8,3 +8,14 @@
# 2020-12-17 13:36:10,446 output:
# 2020-12-17 13:36:10,446 output: # rm -rf /etc/cups
# 2020-12-17 13:36:10,446 output: # ln -s /rw/config/cups /
# 2020-12-17 13:36:10,447 output: etc/cups
# 2020-12-17 13:36:10,447 output: # systemctl --no-block
# 2020-12-17 13:36:10,447 output: restart cups # ⑤
# 2020-12-17 13:36:10,447 output: +
# 2020-12-17 13:36:10,447 output: +# Create split-SSH socket
# 2020-12-17 13:36:10,448 output: +#
# 2020-12-17 13:36:10,448 output: +# See /srv/user_salt/split-
ssh in dom0 for details.
# 2020-12-17 13:36:10,448 output: +SSH_VAULT_VM="ssh-vault"
# 2020-12-17 13:36:10,448 output: +
# 2020-12-17 13:36:10,448 output: +if [["$SSH_VAULT_VM" !=
""]]; then
# 2020-12-17 13:36:10,449 output: +export
SSH SOCK=~user/.SSH_AGENT_$SSH_VAULT_VM
# 2020-12-17 13:36:10,449 output: +rm -f "$SSH SOCK"
# 2020-12-17 13:36:10,449 output: +sudo -u user /bin/sh -c
"umask 117 && ncat -k -l -U '$SSH SOCK' -c 'qrexec-client-vm $SSH_VAULT_VM
```

```

qubes.SSHAgent' &"
# 2020-12-17 13:36:10,449 output: +fi
# 2020-12-17 13:36:10,449 output: -----
# 2020-12-17 13:36:10,450 output: ID: ssh-socket-discoverable
# 2020-12-17 13:36:10,450 output: Function: file.append
# 2020-12-17 13:36:10,450 output: Name: ~user/.bashrc
# 2020-12-17 13:36:10,450 output: Result: True # ⑥
# 2020-12-17 13:36:10,450 output: Comment: Appended 9 lines
# 2020-12-17 13:36:10,451 output: Started: 00:36:08.804326
# 2020-12-17 13:36:10,451 output: Duration: 18.413 ms
# 2020-12-17 13:36:10,451 output: Changes:
# 2020-12-17 13:36:10,451 output: -----
# 2020-12-17 13:36:10,451 output: diff:
# 2020-12-17 13:36:10,452 output: ---
# 2020-12-17 13:36:10,452 output:
# 2020-12-17 13:36:10,452 output: +++
# 2020-12-17 13:36:10,452 output:
# 2020-12-17 13:36:10,452 output: @@ -16,3 +16,12 @@
# 2020-12-17 13:36:10,453 output:
# 2020-12-17 13:36:10,453 output: # export SYSTEMD_PAGER=
# 2020-12-17 13:36:10,453 output:
# 2020-12-17 13:36:10,453 output: # User specific aliases and
functions
# 2020-12-17 13:36:10,453 output: +
# 2020-12-17 13:36:10,454 output: +# Ensure the SSH socket is
discoverable
# 2020-12-17 13:36:10,454 output: +#
# 2020-12-17 13:36:10,454 output: +# See /srv/user_salt/split-
ssh in dom0 for details.
# 2020-12-17 13:36:10,454 output: +SSH_VAULT_VM="ssh-vault"
# 2020-12-17 13:36:10,454 output: +
# 2020-12-17 13:36:10,455 output: +if [["$SSH_VAULT_VM" !=
""]]; then
# 2020-12-17 13:36:10,455 output: +export
SSH_AUTH_SOCK=~user/.SSH_AGENT_$SSH_VAULT_VM
# 2020-12-17 13:36:10,455 output: +fi
# 2020-12-17 13:36:10,455 output:
# 2020-12-17 13:36:10,455 output: Summary for ssh-client
# 2020-12-17 13:36:10,456 output: -----
# 2020-12-17 13:36:10,456 output: Succeeded: 2 (changed=2)
# 2020-12-17 13:36:10,456 output: Failed: 0
# 2020-12-17 13:36:10,456 output: -----
# 2020-12-17 13:36:10,456 output: Total states run: 2
# 2020-12-17 13:36:10,457 output: Total run time: 395.247 ms
# 2020-12-17 13:36:10,457 exit code: 0

```

- ① Since *dom0* is already configured, skipping it saves some time.
- ② Thanks to the top file, only the states that target *ssh-client* will be applied when the state is applied to *ssh-client*.
- ③ Success! The snippet of code is present in the *rc.local* script of *ssh-client*. That means a socket for SSH will be created every time the qube starts.
- ④ Because it was originally missing, Salt appended the snippet to *rc.local* in order to ensure its presence.
- ⑤ Note that depending on the content of your client's *rc.local* script, these context lines may be different. The lines that were appended are identified by the + symbol.
- ⑥ Success! The snippet of code is present in *~user/.bashrc*. That means the socket will be used by

the client's SSH process in order to access the vault's SSH agent.

🚀 You've fully automated the client configuration! In doing so, you took advantage of the fact that `/etc/config/rc.local` and `~user/.bashrc` are persistent files to keep the state simple.

Apply the split-SSH state anytime¶

We've been applying the state after each step, so at this point your split-SSH setup is ready to use. The advantage of automating its configuration, however, is that you can ensure that split-SSH is ready to use at any time with a single command:

```
# dom0

sudo qubesctl --target=fedora-32,ssh-client,ssh-vault state.apply
#                                     ①                               ②

# [Skipping the output for dom0. It can be verbose but contains the same
# snippets demonstrated above.]
# Fedora-32: OK
# ssh-client: OK ③
# ssh-vault: OK

less /var/log/qubes/mgmt-fedora-32.log
# press Shift+G to jump to the end of the file, press Q to quit
#
# [...]

less /var/log/qubes/mgmt-ssh-client.log
# press Shift+G to jump to the end of the file, press Q to quit
#
# [...]

less /var/log/qubes/mgmt-ssh-vault.log
# press Shift+G to jump to the end of the file, press Q to quit
#
# [...]
```

- ① The content of the top files will determine which qubes are able to apply which states. Targeting a qube instructs it to apply all the state files available to it. Because all three split-SSH top files are enabled, we have state to apply for dom0 (implicit target), fedora32, ssh-client and ssh-vault.
- ② As long as the top files are enabled, split-SSH is now part of the state of your Qubes OS installation. Applying the state includes ensuring that split-SSH is set up.
- ③ The output of the command will contain the output of each of the state that you created in this tutorial. For each of them, the line `Result: True` indicates success.

Use split-SSH as usual¶

Whether you set up split-SSH manually or using Salt, the usage remains the same.

1. Generate a new SSH key in the vault:

```
# ssh-vault

ssh-keygen -t ed25519 -C "your_email@example.com"
```

2. Add your new SSH private key to the SSH agent:

```
# ssh-vault  
ssh-add ~/.ssh/id_ed25519
```

3. Add the SSH key to the server you want to connect to. (Example: [Add a new SSH key to your GitHub account](#))
4. Connect to the server as you would normally do. A dialog will be created by *dom0* asking you to authorize the operation **qubes.SSHAgent** from *ssh-client*.

Type `ssh-vault` in the target field and press “OK” to allow the operation. Example:

```
# ssh-client  
  
ssh -T git@github.com  
# Hi <your_username>! You've successfully authenticated, but GitHub does  
not provide shell access.
```

Recap and takeaways¶

All done! Congratulations, you’ve successfully written, applied and tested a split-SSH Salt state in Qubes OS! 🌱

This was no small feat, and along the way you learned:

- how to create qubes and adjust preferences from *dom0*
- how to install packages in a qube’s template so that they are available in the qube
- how to ensure specific files are present in a qube (and only in that qube) even in non-persistent locations
- how to ensure that a qube scripts contain necessary code snippets
- how to apply specific states to specific qubes
- how to use top files in order to apply state to multiple qubes at once
- where to find and how to read the output of the `qubesctl` commands
- where to find the official Salt documentation for the most useful built-in states
- and maybe some things about how split-SSH works in Qubes OS 😊

Further reading¶

The [last section of this guide](#) builds on this tutorial to collect a list of steps to follow when designing your own Salt states for Qubes OS.

The [following guide](#) explains how Salt pillar can be used in Qubes OS to turn Salt states into configurable formulas.

Finally, the [last guide in this series](#) explains how Salt states and formulas can be distributed as RPM packages and why that may be useful.

How to design a Salt state for Qubes OS¶

🐾 *I haven’t written this section yet, it is a generalization of the process demonstrated in the split-SSH tutorial.*

RPM Packaging¶

When using Qubes OS, *dom0* is the most trusted and the most critical *qube* of all. Because of that, it has no network access, and you probably want to perform as many activities as possible in dedicated qubes *outside* of *dom0*.

One of the benefits of using configuration management software like Salt is that you can use version control systems to keep track of the changes to your configuration. More often than not, you'll want to store those changes online (e.g. on GitHub). Besides that, to develop your configuration files (Salt states, Salt formulas) you'll probably want to use a set of tools that you don't want to use in *dom0*.

But copying files from less-trusted to more-trusted qubes should be avoided. Because *dom0* is the most trusted qube and the most critical qube, copying files to *dom0* from a work qube on a regular basis doesn't seem reasonable to me (your circumstances may be different!)

There are, however, mechanisms by which *dom0* can be updated. In particular, the [dom0 secure updates](#) mechanism provides [better security](#) than copying files between qubes. In order to take advantage of the *secure updates* mechanism, we need to package our Salt formulas as RPM packages, and use the security features of the RPM workflow to allow *dom0* to verify them. This guide explains how to do that.

How to publish RPM packages¶

This guide provides a starting point to publish personal RPM packages for use in Qubes OS.

Note

This guide assumes that the RPM packages do not contain confidential or sensitive information.

Overview¶

At the time of writing, Qubes OS' *dom0* runs Fedora. Fedora relies on DNF to install, update and remove RPM packages. DNF downloads RPM packages from *repositories*. The repositories can be either *local* (the repository is located in a directory in your file system), or *remote* (the repository can be reached across a network). Any number of repositories can be used. Some repositories are provided by the Fedora project, others by the Qubes OS project, but you can add your own as well.

This guide describes how to create a repository for your personal RPM packages, how to test it locally, and how make it available through the internet so that you can use it to update your own *dom0*.

Test the RPM repository locally¶

🐱 I haven't written this section yet.

Publish the RPM repository¶

Note

The following instructions deploy your RPM repository using [Surge](#), of course you can deploy it using your preferred web hosting provider, and you can deploy it as a *local* repository if you wish. The RPM repository is contained in the `public/` directory and the content is entirely static.

1. Follow the README instructions to [deploy the RPM repository](#)
2. Commit your changes

How to add an RPM package repository to dom0¶

🐱 I haven't written this section yet.

Threat model¶

Assets¶

The goal is to protect the trustworthiness of *dom0*. Specifically, to avoid to the extent possible the parsing of untrusted data and the execution of untrusted code in *dom0*.

Assumptions¶

- GPG signatures can be trusted to:
 - prove a file was signed by the owner of the key
 - prove the integrity of a file
- *dom0* is trustworthy
- *dom0*'s *secure update mechanism* is trustworthy
- *gpg-vault* is trustworthy when:
 - generating GPG keys
 - copying a GPG key from it into *dom0* (`sudo qvm-run --pass-io 'cat file'`)
- *gpg-backend* is trustworthy when:
 - signing files (*gpg*)
- *work* is trustworthy when:
 - building RPM packages (*tito*, *rpm*)
 - signing RPM packages (*rpmsign*)
 - signing files (*qubes-gpg-client-wrapper*)
- the content of the RPM packages themselves, if authentic, is trustworthy
- the *qube* used to deploy the package repository doesn't need to be trusted
- the server of the package repository doesn't need to be trusted