



Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'quadlet.5'

\$ man quadlet.5

podman-systemd.unit(5) File Formats Manual podman-systemd.unit(5)

NAME

podman-systemd.unit - systemd units using Podman quadlet

SYNOPSIS

name.container, name.volume

Podman unit search path

? /etc/containers/systemd/

? /usr/share/containers/systemd/

Podman user unit search path

? \$XDG_CONFIG_HOME/containers/systemd/

? ~/.config/containers/systemd/

DESCRIPTION

Podman supports starting containers (and creating volumes) via systemd by using a systemd generator. These files are read during boot (and when systemctl daemon-reload is run) and generate corresponding regular systemd service unit files. Both system and user systemd units are supported.

The Podman generator reads the search paths above and reads files with

the extensions `.container` and `.volume`, and for each file generates a similarly named `.service` file. These units can be started and managed with `systemctl` like any other `systemd` service.

The Podman files use the same format as regular `systemd` unit files. Each file type has a custom section (for example, `[Container]`) that is handled by Podman, and all other sections will be passed on untouched, allowing the use of any normal `systemd` configuration options like dependencies or `cgroup` limits.

Enabling unit files

The services created by Podman are considered transient by `systemd`, which means they don't have the same persistence rules as regular units. In particular, it is not possible to "`systemctl enable`" them in order for them to become automatically enabled on the next boot.

To compensate for this, the generator manually applies the `[Install]` section of the container definition unit files during generation, in the same way `systemctl enable` would do when run later.

For example, to start a container on boot, add something like this to the file:

```
[Install]
WantedBy=default.target
```

Currently, only the `Alias`, `WantedBy` and `RequiredBy` keys are supported.

NOTE: To express dependencies between containers, use the generated names of the service. In other words `WantedBy=other.service`, not `WantedBy=other.container`. The same is true for other kinds of dependencies, too, like `After=other.service`.

=====

Container units `[Container]`

Container units are named with a `.container` extension and contain a `[Container]` section describing the container that should be run as a service. The resulting service file will contain a line like `ExecStart=podman run -i image-name`, and most of the keys in this section control the command-line options passed to Podman. However, some options also affect the details of how `systemd` is set up to run and in?

interact with the container.

By default, the Podman container will have the same name as the unit, but with a `systemd-` prefix. I.e. a `$name.container` file will create a `$name.service` unit and a `systemd-$name` Podman container.

There is only one required key, `Image`, which defines the container image the service should run.

Supported keys in Container section are:

`AddCapability=`

By default, the container runs with no capabilities (due to `DropCapabilities='all'`). If any specific caps are needed, then add them with this key. For example using `AddCapability=CAP_DAC_OVERRIDE`.

This is a space separated list of capabilities. This key can be listed multiple times.

For example:

```
AddCapability=CAP_DAC_OVERRIDE CAP_IPC_OWNER
```

`AddDevice=`

Adds a device node from the host into the container. The format of this is `HOST-DEVICE[:CONTAINER-DEVICE][:PERMISSIONS]`, where `HOST-DEVICE` is the path of the device node on the host, `CONTAINER-DEVICE` is the path of the device node in the container, and `PERMISSIONS` is a list of permissions combining 'r' for read,

This key can be listed multiple times.

`Annotation=`

Set one or more OCI annotations on the container. The format is a list of `key=value` items, similar to `Environment`.

This key can be listed multiple times.

`ContainerName=`

The (optional) name of the Podman container. If this is not specified, the default value of `systemd-%N` will be used, which is the same as the service name but with a `systemd-` prefix to avoid conflicts with user-managed containers.

`DropCapability=` (defaults to all)

Drop these capabilities from the default podman capability set, or all

to drop all capabilities.

This is a space separated list of capabilities. This key can be listed multiple times.

For example:

```
DropCapability=CAP_DAC_OVERRIDE CAP_IPC_OWNER
```

Environment=

Set an environment variable in the container. This uses the same format as services in systemd and can be listed multiple times.

EnvironmentFile=

Use a line-delimited file to set environment variables in the container. The path may be absolute or relative to the location of the unit file. This key may be used multiple times, and the order persists when passed to podman run.

EnvironmentHost= (defaults to no)

Use the host environment inside of the container.

Exec=

If this is set then it defines what command line to run in the container. If it is not set the default entry point of the container image is used. The format is the same as for systemd command lines.

ExposeHostPort=

Exposes a port, or a range of ports (e.g. 50-59), from the host to the container. Equivalent to the Podman --expose option.

This key can be listed multiple times.

Group=

The (numeric) gid to run as inside the container. This does not need to match the gid on the host, which can be modified with RemapUsers, but if that is not specified, this gid is also used on the host.

Image=

The image to run in the container. This image must be locally installed for the service to work when it is activated, because the generated service file will never try to download images. It is recommended to use a fully qualified image name rather than a short name, both for performance and robustness reasons.

The format of the name is the same as when passed to podman run, so it supports e.g., using :tag or using digests guarantee a specific image version.

Label=

Set one or more OCI labels on the container. The format is a list of key=value items, similar to Environment.

This key can be listed multiple times.

Network=

Specify a custom network for the container. This has the same format as the --network option to podman run. For example, use host to use the host network in the container, or none to not set up networking in the container.

As a special case, if the name of the network ends with .network, a Podman network called systemd-\$name will be used, and the generated systemd service will contain a dependency on the \$name-network.service. Such a network can be automatically created by using a \$name.network quadlet file.

This key can be listed multiple times.

NoNewPrivileges= (defaults to no)

If enabled (which is the default), this disables the container processes from gaining additional privileges via things like setuid and file capabilities.

Notify= (defaults to no)

By default, Podman is run in such a way that the systemd startup notify command is handled by the container runtime. In other words, the service is deemed started when the container runtime starts the child in the container. However, if the container application supports sd_notify, then setting Notify to true will pass the notification details to the container allowing it to notify of startup on its own.

PodmanArgs=

This key contains a list of arguments passed directly to the end of the podman run command in the generated file (right before the image name in the command line). It can be used to access Podman features other?

wise unsupported by the generator. Since the generator is unaware of what unexpected interactions can be caused by these arguments, is not recommended to use this option.

The format of this is a space separated list of arguments, which can optionally be individually escaped to allow inclusion of whitespace and other control characters. This key can be listed multiple times.

PublishPort=

Exposes a port, or a range of ports (e.g. 50-59), from the container to the host. Equivalent to the Podman `--publish` option. The format is similar to the Podman options, which is of the form `ip:hostPort:containerPort`, `ip::containerPort`, `hostPort:containerPort` or `containerPort`, where the number of host and container ports must be the same (in the case of a range).

If the IP is set to 0.0.0.0 or not set at all, the port will be bound on all IPv4 addresses on the host; use `:::` for IPv6.

Note that not listing a host port means that Podman will automatically select one, and it may be different for each invocation of service.

This makes that a less useful option. The allocated port can be found with the `podman port` command.

This key can be listed multiple times.

ReadOnly= (defaults to no)

If enabled, makes image read-only, with `/var/tmp`, `/tmp` and `/run` a tmpfs (unless disabled by `VolatileTmp=no`).

NOTE: Podman will automatically copy any content from the image onto the tmpfs

RemapGid=

RemapGid key to force a particular host uid to be mapped to the container.

In `keep-id` mode, the running user is mapped to the same id in the container. This is supported only on user systemd units.

If `RemapUsers` is enabled, this specifies a gid mapping of the form `container_gid:from_gid:amount`, which will map amount number of gids on the host starting at `from_gid` into the container, starting at `container_gid`.

tainer_gid.

RemapUid=

If RemapUsers is enabled, this specifies a uid mapping of the form container_uid:from_uid:amount, which will map amount number of uids on the host starting at from_uid into the container, starting at container_uid.

tainer_uid.

RemapUidSize=

If RemapUsers is enabled and set to auto, this specifies the count of the ids to remap

RemapUsers=

If this is set, then host user and group ids are remapped in the container. It currently supports values: auto, manual and keep-id.

In manual mode, the RemapUid and RemapGid options can define an exact mapping of uids from host to container. You must specify these.

In auto mode mode, the subuids and subgids allocated to the containers user is used to allocate host uids/gids to use for the container. By default this will try to estimate a count of the ids to remap, but

RemapUidSize can be specified to use an explicit size. Use RemapUid and

RunInit= (default to no)

If enabled, the container will have a minimal init process inside the container that forwards signals and reaps processes.

SeccompProfile=

Set the seccomp profile to use in the container. If unset, the default podman profile is used. Set to either the pathname of a json file, or unconfined to disable the seccomp filters.

Timezone= (if unset uses system-configured default)

The timezone to run the container in.

User=

The (numeric) uid to run as inside the container. This does not need to match the uid on the host, which can be modified with RemapUsers, but if that is not specified, this uid is also used on the host.

VolatileTmp= (default to no, or yes if ReadOnly enabled)

If enabled, the container will have a fresh tmpfs mounted on /tmp.

NOTE: Podman will automatically copy any content from the image onto the tmpfs

Volume=

Mount a volume in the container. This is equivalent to the Podman `--volume` option, and generally has the form `[[SOURCE-VOLUME|HOST-DIR:]CONTAINER-DIR[:OPTIONS]]`.

As a special case, if SOURCE-VOLUME ends with `.volume`, a Podman named volume called `systemd-$name` will be used as the source, and the generated systemd service will contain a dependency on the `$name-volume.service`. Such a volume can be automatically be lazily created by using a `$name.volume` quadlet file.

This key can be listed multiple times.

=====

Kube units [Kube]

Kube units are named with a `.kube` extension and contain a `[Kube]` section describing how podman kube play should be run as a service. The resulting service file will contain a line like `ExecStart=podman kube play ? file.yml`, and most of the keys in this section control the command-line options passed to Podman. However, some options also affect the details of how systemd is set up to run and interact with the container.

There is only one required key, `Yaml`, which defines the path to the Kubernetes YAML file.

Supported keys in the Kube section are:

ConfigMap=

Pass the Kubernetes ConfigMap YAML at path to podman kube play via the `--configmap` argument. Unlike the `configmap` argument, the value may contain only one path but it may be absolute or relative to the location of the unit file.

This key may be used multiple times

Network=

Specify a custom network for the container. This has the same format as the `--network` option to podman kube play. For example, use `host` to use the host network in the container, or `none` to not set up networking in

the container.

As a special case, if the name of the network ends with `.network`, a Podman network called `systemd-$name` will be used, and the generated `systemd` service will contain a dependency on the `$name-network.service`.

Such a network can be automatically created by using a `$name.network` quadlet file.

This key can be listed multiple times.

`PublishPort=`

Exposes a port, or a range of ports (e.g. 50-59), from the container to the host. Equivalent to the podman kube play's `--publish` option. The format is similar to the Podman options, which is of the form `ip:host?Port:containerPort`, `ip::containerPort`, `hostPort:containerPort` or `containerPort`, where the number of host and container ports must be the same (in the case of a range).

If the IP is set to `0.0.0.0` or not set at all, the port will be bound on all IPv4 addresses on the host; use `:::` for IPv6.

The list of published ports specified in the unit file will be merged with the list of ports specified in the Kubernetes YAML file. If the same container port and protocol is specified in both, the entry from the unit file will take precedence.

This key can be listed multiple times.

`RemapGid=`

If `RemapUsers` is enabled, this specifies a gid mapping of the form `container_gid:from_gid:amount`, which will map amount number of gids on the host starting at `from_gid` into the container, starting at `container_gid`.

`RemapUid=`

If `RemapUsers` is enabled, this specifies a uid mapping of the form `container_uid:from_uid:amount`, which will map amount number of uids on the host starting at `from_uid` into the container, starting at `container_uid`.

`RemapUidSize=`

If `RemapUsers` is enabled and set to `auto`, this specifies the count of

the ids to remap.

RemapUsers=

If this is set, then host user and group ids are remapped in the container. It currently supports values: auto, and keep-id.

In auto mode mode, the subuids and subgids allocated to the containers user is used to allocate host uids/gids to use for the container. By default this will try to estimate a count of the ids to remap, but RemapUidSize can be specified to use an explicit size. Use RemapUid and RemapGid key to force a particular host uid to be mapped to the container.

In keep-id mode, the running user is mapped to the same id in the container. This is supported only on user systemd units.

Yaml=

The path, absolute or relative to the location of the unit file, to the Kubernetes YAML file to use.

=====

Network units [Network]

Network files are named with a .network extension and contain a section [Network] describing the named Podman network. The generated service is a one-time command that ensures that the network exists on the host, creating it if needed.

For a network file named \$NAME.network, the generated Podman network will be called systemd-\$NAME, and the generated service file \$NAME-network.service.

Using network units allows containers to depend on networks being automatically pre-created. This is particularly interesting when using special options to control network creation, as Podman will otherwise create networks with the default options.

Supported keys in Network section are:

DisableDNS= (defaults to no)

If enabled, disables the DNS plugin for this network.

This is equivalent to the Podman --disable-dns option

Driver= (defaults to bridge)

Driver to manage the network. Currently bridge, macvlan and ipvlan are supported.

This is equivalent to the Podman --driver option

Gateway=

Define a gateway for the subnet. If you want to provide a gateway address, you must also provide a subnet option.

This is equivalent to the Podman --gateway option

This key can be listed multiple times.

Internal= (defaults to no)

Restrict external access of this network.

This is equivalent to the Podman --internal option

IPAMDriver=

Set the ipam driver (IP Address Management Driver) for the network.

Currently host-local, dhcp and none are supported.

This is equivalent to the Podman --ipam-driver option

IPRange=

Allocate container IP from a range. The range must be a complete subnet and in CIDR notation. The ip-range option must be used with a subnet option.

This is equivalent to the Podman --ip-range option

This key can be listed multiple times.

IPv6=

Enable IPv6 (Dual Stack) networking.

This is equivalent to the Podman --ipv6 option

Label=

Set one or more OCI labels on the network. The format is a list of key=value items, similar to Environment.

This key can be listed multiple times.

Options=

Set driver specific options.

This is equivalent to the Podman --opt option

Subnet=

The subnet in CIDR notation.

This is equivalent to the Podman --subnet option

This key can be listed multiple times.

=====

Volume units [Volume]

Volume files are named with a .volume extension and contain a section [Volume] describing the named Podman volume. The generated service is a one-time command that ensures that the volume exists on the host, creating it if needed.

For a volume file named \$NAME.volume, the generated Podman volume will be called systemd-\$NAME, and the generated service file \$NAME-volume.service.

Using volume units allows containers to depend on volumes being automatically pre-created. This is particularly interesting when using special options to control volume creation, as Podman will otherwise create volumes with the default options.

Supported keys in Volume section are:

Copy= (default to yes)

If enabled, the content of the image located at the mountpoint of the volume is copied into the volume on the first run.

Device=

The path of a device which should be mounted for the volume.

Group=

The host (numeric) gid, or group name to use as the group for the volume

Label=

Set one or more OCI labels on the volume. The format is a list of key=value items, similar to Environment.

This key can be listed multiple times.

Options=

The mount options to use for a filesystem as used by the mount(8) command -o option.

Type=

The filesystem type of Device as used by the mount(8) commands -t op?

tion.

User=

The host (numeric) uid, or user name to use as the owner for the volume

EXAMPLES

Example test.container:

[Unit]

Description=A minimal container

Before=local-fs.target

[Container]

Use the centos image

Image=quay.io/centos/centos:latest

Volume=test.volume:/data

In the container we just run sleep

Exec=sleep 60

[Service]

Restart service when sleep finishes

Restart=always

[Install]

Start by default on boot

WantedBy=multi-user.target default.target

Example test.kube:

[Unit]

Description=A kubernetes yaml based service

Before=local-fs.target

[Kube]

Yaml=/opt/k8s/deployment.yml

[Install]

Start by default on boot

WantedBy=multi-user.target default.target

Example test.volume:

[Volume]

User=root

Group=projectname

Label=org.test.Key=value

Example test.network:

[Network]

Subnet=172.16.0.0/24

Gateway=172.16.0.1

IPRange=172.16.0.0/28

Label=org.test.Key=value

SEE ALSO

systemd.unit(5), systemd.service(5), podman-run(1) podman-network-cre?

ate(1)

podman-systemd.unit(5)