Container Networking Basics

Objectives

- Describe how containers communicate with each other.

Container Networking Basics

You can use Podman networking to improve security and ease the communication of containerized applications. For example, if an application has separate containers for running the UI, the back-end API, and the database, then you might want to restrict the UI container from accessing the database container. To do this you can create an isolated podman network to share between the UI and the API containers, and a different isolated network to share between the API and the database containers.

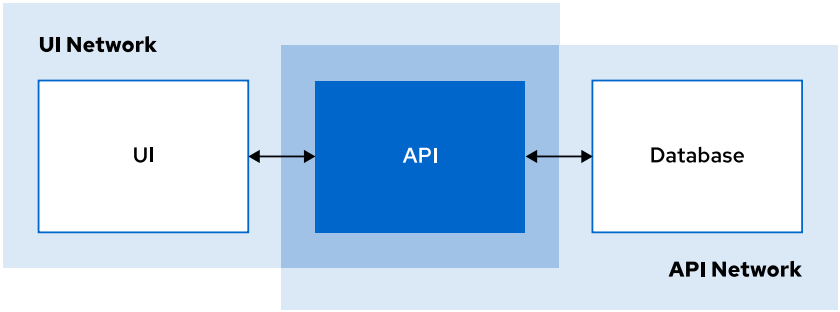

Figure 2.8: Example of isolated communication by using Podman networks

In the preceding example diagram, the UI and API containers are attached to the `ui-network` Podman network. The API and database containers are attached to the `api-network` Podman network.

Managing Podman Networks

Podman network management is done via the `podman network` subcommand. This subcommand includes the following operations:

`podman network create`
Creates a new Podman network. This command accepts various options to configure properties of the network, including gateway address, subnet mask, and whether to use IPv4 or IPv6.

`podman network ls`
Lists existing networks and a brief summary of each. Options for this command include various filters and an output format to list other values for each network.

`podman network inspect`
Outputs a detailed JSON object containing configuration data for the network.

`podman network rm`

Removes a network.

`podman network prune`
Removes any networks that are not currently in use by any running containers.

`podman network connect`
Connects an already running container to an existing network. Alternatively, connect containers to a Podman network on container creation by using the `--net` option.

`podman network disconnect`
Disconnects a container from a network.
For example, the following command creates a new Podman network called `example-net`:

```
[user@host ~]$ podman network create example-net
```

To connect a new container to this Podman network, use the `--net` option. The following example command creates a new container called `my-container`, which is connected to the `example-net` network.

```
[user@host ~]$ podman run -d --name my-container \
  --net example-net container-image:latest
```

When you create new containers, you can connect them to multiple networks by specifying network names in a comma-separated list. For example, the following command creates a new container called `double-connector` that connects to both the `postgres-net` and `redis-net` networks.

```
[user@host ~]$ podman run -d --name double-connector \
  --net postgres-net,redis-net container-image:latest
```

Podman uses the `pasta` network mode when you create a container without specifying a `--net` option. You cannot connect a container that uses the `pasta` network mode to a Podman `bridge` network.

For example, if the container called `my-container` is already running and has no networks assigned, attempting to connect it to a Podman `bridge` network fails.:

```
[user@host ~]$ podman run -d --name my-container example-image
CONTAINER_ID
[user@host ~]$ podman network connect example-net my-container
Error: "pasta" is not supported: invalid network mode
```

To connect the container to the network you must recreate the container with the `--net` option set to the desired network.

> **NOTE**
>
> Starting in Podman v4.2.0, the `podman network create` command supports the `isolate` option with the default `bridge` driver. This option isolates the network by blocking any traffic from it to any

other network with the `isolate` option enabled. Use the `podman network create` command with the `-o isolate` option to enable isolation.

Rooful and Rootless Container Networks

Podman networking behaves differently when you run containers as the root user, *rootful containers*, compared to when you run containers as a non-root user, *rootless containers* .

When you run rootful containers, then Podman uses the `podman` network which is pre-configured and attaches containers to it. Note, that the `podman` default network comes with DNS disabled by default. Without DNS, containers attached to a network can reach other containers by using the container's IP address, but they cannot use container names to communicate with each other.

If you run a rootless container, then the non-root user cannot create networking bridges or manage virtual interfaces on the host. Instead, Podman creates the required networking interfaces on a network namespace. By default, Podman does not attach rootless containers to the `podman` default network. If you want rootless containers to communicate with each other, then you must create a Podman network and attach the containers to it, or attach the containers to the `default` Podman network.

IMPORTANT

Starting in Podman v5, the rootless network back end changed from `slirp4netns` to `pasta`.

For more information, see the *Pasta default for rootless networking* section from the *Podman 5.0 Breaking Changes* reference and the *Official pasta Site* reference in the references section.

Container Domain Name Resolution

When using a network with DNS enabled, a container's hostname or alias is the name assigned to the container.

For example, if you start a container called `nginx-container` with the following command, then containers on the `test-net` network can connect to the first container by using the `nginx-container` hostname. The `nginx-container` hostname resolves to the current IP address of the container called `nginx-container`.

```
[user@host ~]$ podman run --net test-net --name nginx-container nginx-image
```

Then if the container called `nginx-container` exposes an HTTP server on port 8080, you can run the following `curl` command from another container on the `test-net` network.

```
$ curl http://nginx-container:8080
```

Referencing External Hosts by Name

When you create a container, Podman adds the `host.containers.internal` and `host.docker.internal` hostnames to the `/etc/hosts` file by default. You can use those hostnames to connect to any service that the host machine exposes.

Also, you can add host and IP address pairs to the container's `/etc/hosts` file by using the `--add-host=host:ip` parameter. This parameter adds a line in the `/etc/hosts` file with the given hostname and IP, so applications inside the container can resolve the hostname to its IP address.

```
[user@host ~]$ podman run --add-host=myhostname:192.168.1.1 example-image
```

REFERENCES

Podman Networking

Podman Network Create

Basic Networking Guide for Podman

Official pasta Site

Podman 5.0 Breaking Changes