Guided Exercise: Accessing Containerized Network Services

Run two applications that communicate by using the Podman DNS system.

Outcomes

You should be able to understand how DNS works in Podman networks.

```
[student@workstation ~]$ lab start basics-exposing
```

This exercise uses two applications: `times` and `cities`. The `times` application returns the current time of a given city, and the `cities` application returns information about a specific city.

To fetch the current time of the city, the `cities` application fetches data from the `times` application.

Instructions

1. Examine the source code of the applications.

   Go to the `/home/student/DO188/labs/basics-exposing` directory, where the applications are available.

   ```
   [student@workstation ~]$ cd ~/DO188/labs/basics-exposing
   no output expected
   ```

   View the contents of the `podman-info-times/app/main.go` file.

   ```
   [student@workstation basics-exposing]$ cat podman-info-times/app/main.go
   ...output omitted...
     http.ListenAndServe(":8080", r)
   ...output omitted...
   ```

   The application exposes an HTTP server on port `8080` that returns the current time for a city.

   Examine the Containerfile for the times application.

   ```
   [student@workstation basics-exposing]$ cat podman-info-times/Containerfile
   FROM registry.access.redhat.com/ubi9/go-toolset:1.22 as build
   WORKDIR /app
   USER root
   COPY app .
   RUN go build

   FROM registry.access.redhat.com/ubi9/ubi-minimal:9.5
   WORKDIR /app
   COPY --from=build /app/times-app .
   EXPOSE 8080
   ENTRYPOINT ["/app/times-app"]
   ```

   The `registry.ocp4.example.com:8443/redhattraining/podman-info-times` container image is based on this Containerfile.

   View the `podman-info-cities/app/main.go` file.

   ```
   [student@workstation basics-exposing]$ cat podman-info-cities/app/main.go
   ...output omitted...
     http.ListenAndServe(":8090", r)
   ...output omitted...
   ```

   The application exposes an HTTP server on port `8090` that returns information for a city.

   Examine the Containerfile for the cities application.

   ```
   [student@workstation basics-exposing]$ cat podman-info-cities/Containerfile
   FROM registry.access.redhat.com/ubi9/go-toolset:1.22 as build
   WORKDIR /app
   USER root
   COPY app .
   RUN go build

   FROM registry.access.redhat.com/ubi9/ubi-minimal:9.5
   WORKDIR /app
   COPY --from=build /app/cities-app .
   EXPOSE 8090
   ENTRYPOINT ["/app/cities-app"]
   ```

The `registry.ocp4.example.com:8443/redhattraining/podman-info-cities` container image is based on this Containerfile.

Go to the home directory. The rest of the exercise does not involve local files.

```
[student@workstation basics-exposing]$ cd ~
no output expected
```

2. Create a Podman network with DNS enabled.

   Observe that DNS is disabled in the default Podman network.

```
[student@workstation ~]$ podman network inspect podman
...output omitted...
    "dns_enabled": false,
...output omitted...
```

   Create a network by using the `podman network create` command.

```
[student@workstation ~]$ podman network create cities
cities
```

   Inspect the `cities` network to verify that DNS is enabled.

```
[student@workstation ~]$ podman network inspect cities
...output omitted...
    "dns_enabled": true,
...output omitted...
```

3. Start and test the `times` application attached to the `cities` network.

   Create a container for the times application that forwards port `8080` from the container to the host. Attach the container to the `cities` network.

```
[student@workstation ~]$ podman run --name times-app \
  --network cities -p 8080:8080 -d \
  registry.ocp4.example.com:8443/redhattraining/podman-info-times:v0.1
...output omitted...
256...ee0
```

   Inspect the `times-app` container to find its private IP within the Podman network and copy the IP address.

```
[student@workstation ~]$ podman inspect times-app \
  -f '{{.NetworkSettings.Networks.cities.IPAddress}}'
10.89.1.2
```

> **NOTE**
>
> If the name of the network includes characters such as a dash (`-`), then the preceding command fails. To retrieve the IP address of the container attached to such a network, you must use alternative methods to limit the output of the `podman inspect` command. For example, the `grep` or `jq` tools can search or parse the output, respectively.

   Use the private IP address to make a request to the `/times` API endpoint from another container. Run the `curl` command inside a `ubi-minimal` container to fetch the current time for Bangkok, which has the code `BKK`. Attach the container to the `cities` network.

   Replace `IP_ADDRESS` with the IP address from the previous step.

```
[student@workstation ~]$ podman run --rm --network cities \
  registry.ocp4.example.com:8443/ubi9/ubi-minimal:9.5 \
  curl -s http://IP_ADDRESS:8080/times/BKK && echo
...output omitted...
2022-06-28T19:59:55.241Z
```

> **NOTE**
>
> Without providing arguments, the `echo` command prints a new line character, which improves the output formatting.

Use the DNS name to make a request to the `/times` API endpoint from another container.

```
[student@workstation ~]$ podman run --rm --network cities \
  registry.ocp4.example.com:8443/ubi9/ubi-minimal:9.5 \
  curl -s http://times-app:8080/times/BKK && echo
...output omitted...
2022-06-28T20:04:37.241Z
```

Note that `times-app` is used as the hostname instead of the IP address.

4. Start and test the `cities` application attached to the `cities` network.

   Create a container called `cities-app` and start the application on port `8090`. Attach the container to the `cities` network.

   Set the `TIMES_APP_URL` environment variable to the URL of the `times` application. Notice that the URL uses the name of the `times-app` container.

```
[student@workstation ~]$ podman run --name cities-app \
  --network cities -p 8090:8090 -d \
  -e TIMES_APP_URL=http://times-app:8080/times \
  registry.ocp4.example.com:8443/redhattraining/podman-info-cities:v0.1
...output omitted...
dcc...1fd
```

   Fetch the city information of Madrid by using the `MAD` city code.

```
[student@workstation ~]$ curl -s http://localhost:8090/cities/MAD | jq
{
  "name": "Madrid",
  "time": "2022-06-01T12:34:56.123Z",
  "population": 3223000,
  "country":"Spain"
}
```

> **NOTE**
>
> Passing the output of the `curl` command to the `jq` command provides better formatting of the JSON response compared to viewing the raw output.

   Fetch the city information of San Diego by using the `SAN` city code.

```
[student@workstation ~]$ curl -s http://localhost:8090/cities/SAN | jq
{
  "name": "San Diego",
  "time": "2022-06-01T12:34:56.123Z",
  "population": 1415000,
  "country": "United States of America"
}
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish basics-exposing
```