Guided Exercise: Container Logging and Troubleshooting

Learn how to troubleshoot and solve some common problems when running containers.

Outcomes

You should be able to:

- Search container logs.
- Verify a container definition.
- Use commands inside a container for troubleshooting.
- Use host commands that are not present in the container for container troubleshooting.
- Troubleshoot networking issues.
- Troubleshoot file permission issues.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

The `lab start` command performs the following actions:

- Creates a PostgreSQL database container called `smart-home-db` that is attached to the `troubleshooting-lab` network.
- Creates the `smart-home-api` container, which fails to start.
- Copies the `automations.yaml` configuration file to the project directory.

The smart home API uses a server called `uvicorn` to run a Python back end.

```
[student@workstation ~]$ lab start troubleshooting-logging
```

Instructions

1. Read the `smart-home-api` container logs to see the cause of the failure.

   ```
   [student@workstation ~]$ podman logs smart-home-api
   ...output omitted...
     File "/opt/app-root/lib64/python3.9/site-packages/psycopg2/__init__.py", line 122, in connect
       conn == _connect(dsn, connection_factory=connection_factory, **kwasync)
   sqlalchemy.exc.OperationalError: (psycopg2.OperationalError) could not translate host name "smart-home-db" to address:
   Name or service not known

   (Background on this error at: https://sqlalche.me/e/14/e3q8)
   ```

   The container fails with an error that indicates that there is a DNS problem. The application in the `smart-home-api` container cannot resolve the `smart-home-db` database hostname.

2. Verify that you can resolve the database container name by using DNS.

   Inspect the `smart-home-db` container to verify the networks that it is using.

   ```
   [student@workstation ~]$ podman inspect \
     smart-home-db --format='{{.NetworkSettings.Networks}}'
   map[troubleshooting-lab:0xc000f92900]
   ```

   Verify that the `troubleshooting-lab` network has DNS enabled.

   ```
   [student@workstation ~]$ podman network inspect troubleshooting-lab
   [
        {
             "name": "troubleshooting-lab",
             ...output omitted...
             "dns_enabled": true,
             ...output omitted...
        }
   ]
   ```

   Inspect the `smart-home-api` container to verify the networks that it is using.

   ```
   [student@workstation ~]$ podman inspect \
     smart-home-api --format='{{.NetworkSettings.Networks}}'
   map[pasta:0xc00059fc20]
   ```

   The `smart-home-api` container is not using any network. The container requires access to the `troubleshooting-lab` network to reach the `smart-home-db` container.

Remove the `smart-home-api` container.

```
[student@workstation ~]$ podman rm smart-home-api
8eee...ef70
```

Run the `smart-home-api` container and attach it to the `troubleshooting-lab` network with the following configuration.

- Use the `--rm` and `-d` options.
- Call the container `smart-home-api`.
- Attach the container to the `troubleshooting-lab` network.
- Use the following environment variables:
  - `DB_HOST=smart-home-db`
  - `DB_USER=backend`
  - `DB_PASSWORD=secret_pass`
- Map the host port `8080` to container port `8080`.
- Use the `registry.ocp4.example.com:8443/redhattraining/smart-home-api:1.0` image.

```
[student@workstation ~]$ podman run -d --rm \
  --name smart-home-api \
  -e DB_HOST=smart-home-db -e DB_USER=backend -e DB_PASSWORD=secret_pass \
  -p 8080:8080 \
  --network troubleshooting-lab \
  registry.ocp4.example.com:8443/redhattraining/smart-home-api:1.0
a361...241a
```

Verify that the `smart-home-api` container starts successfully by reading the container logs.

```
[student@workstation ~]$ podman logs smart-home-api
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

3. Test the API endpoint of the `smart-home-api` container and troubleshoot the connection problems.

   Test the API from inside the `smart-home-api` container by querying the `http://localhost:8080/device/1` address. Verify that the request fails due to a connection error on port 8080.

```
[student@workstation ~]$ podman exec \
  smart-home-api curl http://localhost:8080/device/1
...output omitted...
curl: (7) Failed to connect to localhost port 8080: Connection refused
```

   Look for the server active connections. Use the `socket statistics` (`ss`) command with the following options.

   - `-p`: show the process using the socket
   - `-a`: show listening and established connections
   - `-n`: display numeric ports instead of mapped service names
   - `-t`: display TCP sockets

```
[student@workstation ~]$ podman exec smart-home-api ss -pant
Error: crun: executable file ss not found in $PATH: No such file or directory: OCI runtime attempted to invoke a command
that was not found
```

   The `ss` command is not available in the container. Images usually lack many commands. This is a good security practice because it reduces the container attack surface.

   Use the `nsenter` host command to run the host `ss` command in the container.

   Copy the container process ID (PID) for later use with the `nsenter` command. The PID number might differ for your container.

```
[student@workstation ~]$ podman inspect smart-home-api --format '{{.State.Pid}}'
2809
```

   Run the `nsenter` command. Use the `-t` option to set the container PID obtained previously as the target. Use the `-n` option to enter the `smart-home-pid` network namespace.

```
[student@workstation ~]$ sudo nsenter -t 2809 -n ss -pant
Netid State  Recv-Q Send-Q Local Address:Port  Peer Address:Port Process
tcp   LISTEN 0      2048       0.0.0.0:8000       0.0.0.0:*  users: ("uvicorn",pid=76641,fd=23
...output omitted...
```

The application server `uvicorn` is listening on the `0.0.0.0` address on port 8000. In the preceding steps, you attempted to test the server on port 8080, which is incorrect.

Confirm that the application responds on port 8000 from within the container. Send a request to the `http://localhost:8000/device/1` endpoint.

```
[student@workstation ~]$ podman exec -it \
  smart-home-api curl  http://localhost:8000/device/1
{"id":1,"name":"Bedroom Light","state":"on"}%
```

Test whether the application is reachable from the host by sending a request to the `http://localhost:8080/device/1` endpoint.

```
[student@workstation ~]$ curl http://localhost:8080/device/1
curl: (56) Recv failure: Connection reset by peer
```

The request fails because the port mapping is still using the incorrect container port.

Recreate the container with the correct port mapping. Map port `8080` from the host to port `8000` in the container.

Stop the `smart-home-api` container.

```
[student@workstation ~]$ podman stop smart-home-api
smart-home-api
```

Rerun the container. Map the `8000` container port to the `8080` host port.

```
[student@workstation ~]$ podman run -d --rm \
  --name smart-home-api \
  -e DB_HOST=smart-home-db -e DB_USER=backend -e DB_PASSWORD=secret_pass \
  -p 8080:8000 \
  --network troubleshooting-lab \
  registry.ocp4.example.com:8443/redhattraining/smart-home-api:1.0
4a57...4f1g
```

Confirm that the application responds from the host by sending a request to the `http://localhost:8080/device/1` endpoint.

```
[student@workstation ~]$ curl http://localhost:8080/device/1
{"id":1,"name":"Bedroom Light","state":"on"}%
```

4. Customize the automation scripts of the `smart-home-api` container. To do this, bind mount the `automations.yaml` configuration file to the `/config/automations.yaml` container path.

   The API provides an `/automations` endpoint to validate the current container configuration.

   Change to the `/home/student/DO188/labs/troubleshooting-logging/smart-home` directory.

   ```
   [student@workstation ~]$ cd ~/DO188/labs/troubleshooting-logging/smart-home
   no output expected
   ```

   Stop the `smart-home-api` container.

   ```
   [student@workstation smart-home]$ podman stop smart-home-api
   smart-home-api
   ```

   Recreate the container with the `./automations.yaml` file mounted to the `/config/automations.yaml` file within the container.

   ```
   [student@workstation smart-home]$ podman run -d --rm \
     --name smart-home-api \
     -e DB_HOST=smart-home-db -e DB_USER=backend -e DB_PASSWORD=secret_pass \
     -p 8080:8000 \
     --network troubleshooting-lab \
     -v ./automations.yaml:/config/automations.yaml \
     registry.ocp4.example.com:8443/redhattraining/smart-home-api:1.0
   8n57...4f1g
   ```

   Confirm that the container has access to the configuration.

```
[student@workstation smart-home]$ curl http://localhost:8080/automations
{"detail":"No automations defined"}%
```

Verify that the `/config/automations.yaml` file exists in the container.

```
[student@workstation smart-home]$ podman exec \
  smart-home-api ls -l /config/
ls: cannot access '/config/automations.yaml': Permission denied
total 0
-????????? ? ? ? ?            ? automations.yaml
```

The file exists but the container cannot read the file permissions.

Troubleshoot SELinux permissions.

Run `ls` with the `-Z` option to get the SELinux context for the `automations.yaml` file on the host.

```
[student@workstation smart-home]$ ls -Z automations.yaml
unconfined_u:object_r:cache_home_t:s0 automations.yaml
```

The `user_home_t` SELinux context indicates that Podman must change the SELinux permissions to access the file.

Stop the `smart-home-api` container.

```
[student@workstation smart-home]$ podman stop smart-home-api
smart-home-api
```

To fix the SELinux permissions, recreate the container and add the `z` option to the volume mount.

```
[student@workstation smart-home]$ podman run -d --rm \
  --name smart-home-api \
  -e DB_HOST=smart-home-db -e DB_USER=backend -e DB_PASSWORD=secret_pass \
  -p 8080:8000 \
  --network troubleshooting-lab \
  -v ./automations.yaml:/config/automations.yaml:Z \
  registry.ocp4.example.com:8443/redhattraining/smart-home-api:1.0
1n57...4f1g
```

Verify that the SELinux context changed for the `automations.yaml` file.

```
[student@workstation smart-home]$ ls -Z automations.yaml
system_u:object_r:container_file_t:s0:c539,c793 automations.yaml
```

The `container_file_t` SELinux context permits Podman to access the host file from the container.

Validate that the `/automations` endpoint returns the custom configuration from the host `automations.yaml` file.

```
[student@workstation smart-home]$ curl http://localhost:8080/automations
[{"automation":"Turn garage lights ON when presence detected","wait_for_trigger":[{"platform":"event","event_type":"PRESEN
CE_DETECTED"},{"platform":"state","device_id":1,"to":"on","for":60}]}]%
```

5. Return to the home directory:

```
[student@workstation smart-home]$ cd
no output expected
```

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 smart-home]$ lab finish troubleshooting-logging
```