Chapter 8

Troubleshooting Containerized Applications

Troubleshooting S2I Builds and Deployments

S2I Image Creation

- Two Major Steps:
  - **Build step:** Responsible for compiling source code, downloading library dependencies, and packaging the application as a container image. Furthermore, the build step pushes the image to the OpenShift registry for the deployment step. The BuildConfig (BC) OpenShift resources drive the build step.
  - **Deployment step:** Responsible for starting a pod and making the application available for OpenShift. This step executes after the build step, but only if the build step succeeded. The Deployment OpenShift resources drive the deployment step.
- Upon a successful build, the application starts on a separate pod named as <application-name>-<string>.

OpenShift UI

- To identify any build issues, the logs for a build can be evaluated and analyzed by clicking the **Builds** link from the left panel
- The **Logs** tab of the build details page shows the output generated by the build execution. Those logs are handy to identify build issues.
- Use the **Deployment** link under **Workloads** section from the left panel to identify issues during the deployment step.
- After selecting the appropriate deployment object, details show in the **Details** section.
- **Note:** These options are also available from the terminal using the ***oc command.***

Describing Common Problems

- Troubleshooting Permission Issues
- Troubleshooting Invalid Parameters
- Troubleshooting Volume Mount Errors
- Troubleshooting Obsolete Images

Troubleshooting Permission Issues

- The following Dockerfile creates a Nexus container. Note the USER instruction indicating the nexus user should be used.
- Trying to use the image generated by this Dockerfile without addressing volume permissions drives to errors when the container starts.

```
FROM ubi8/ubi:8.1
...contents omitted...
RUN chown -R nexus:nexus ${NEXUS_HOME}

USER nexus
WORKDIR ${NEXUS_HOME}

VOLUME ["/opt/nexus/sonatype-work"]
...contents omitted...
```

```
[user@host ~]$ oc logs nexus-1-wzjrn
...output omitted...
... org.sonatype.nexus.util.LockFile - Failed to write lock file
...FileNotFoundException: /opt/nexus/sonatype-work/nexus.lock (Permission denied)
...output omitted...
... org.sonatype.nexus.webapp.WebappBootstrap - Failed to initialize
...StateException: Nexus work directory already in use: /opt/nexus/sonatype-work
...output omitted...
```

- To solve this issue, relax the OpenShift project security with the *command oc adm policy*

```
[user@host ~]$ oc adm policy add-scc-to-user anyuid -z default
```

Troubleshooting Invalid Parameters

- Need to ensure that the same values for parameters reach all containers in the application.
- A solution is to centralize shared parameters is to store them in **ConfigMaps**. Those ConfigMaps can be injected through the Deployment into containers as environment variables. Injecting the same ConfigMap into different containers ensures that not only the same environment variables are available, but also the same values.

Troubleshooting Volume Mount Errors

- When redeploying an application that uses a persistent volume on a local file system, a pod might not be able to allocate a persistent volume claim even though the persistent volume indicates that the claim is released.
- To resolve the issue, delete the persistent volume claim and then the persistent volume. Then recreate the persistent volume:

```
[user@host ~]$ oc delete pv <pv_name>

[user@host ~]$ oc create -f <pv_resource_file>
```

Troubleshooting Obsolete Images

- If you push a new image to the registry with the same name and tag, you must remove the image from each node the pod is scheduled on with the command *podman rmi*.
- Run the *oc adm prune* command for an automated way to remove obsolete images and other resources.

Guided Exercise: Troubleshooting an OpenShift Build

- **[student@workstation ~]$ lab troubleshoot-s2i start**
- https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch08s02

Troubleshooting Containerized Applications

Forwarding Ports for Troubleshooting

- Podman provides port forwarding features by using the -p option along with the run subcommand.

```
[user@host ~]$ podman run --name db -p 30306:3306 mysql
```

- OpenShift provides the oc port-forward command for forwarding a local port to a pod port.

```
[user@host ~]$ oc port-forward db 30306 3306
```

Enabling Remote Debugging with Port Forwarding

- Another use for the port forwarding feature is enabling remote debugging.
- For example, Red Hat CodeReady Studio allows users to utilize the Java Debug Wire Protocol (JDWP) to communicate between its debugger and the Java Virtual Machine. When enabled, developers can step through each line of code as it is being executed in real time.

Accessing Container Logs

To retrieve the output of a running container, use the following `podman` command:

```
[user@host ~]$ podman logs containerName
```

In OpenShift, the following command returns the output for a container within a pod:

```
[user@host ~]$ oc logs podName -c containerName
```

OpenShift Events

- Some developers consider Podman and OpenShift logs to be too low-level, making troubleshooting difficult. Fortunately, OpenShift provides a high-level logging and auditing facility called events.

```
[user@host ~]$ oc get events
```

- The approach is provided by the describe subcommand.

```
[user@host ~]$ oc describe pod mysql
...output omitted...
Events:
  FirstSeen   LastSeen    Count From          Reason          Message
  Wed, 10 ... Wed, 10 ... 1     {scheduler } scheduled       Successfully as...
...output omitted...
```

Accessing Running Containers

- Podman and OpenShift provide the exec subcommand, allowing the creation of new processes inside a running container, with the standard output and input of these processes redirected to the user terminal.

```
[user@host ~]$ podman exec options container command arguments
```

```
[user@host ~]$ oc exec options pod -c container -- command arguments
```

- Examples:

```
[user@host ~]$ podman exec apache-container cat /var/log/httpd/error_log
```

```
[user@host ~]$ oc exec -it myhttpdpod /bin/bash
```

Guided Exercise: Configuring Apache Container Logs for Debugging

- **[student@workstation ~]$ lab troubleshoot-container start**
- https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch08s04

Lab: Troubleshooting Containerized Applications

- **[student@workstation ~]$ lab troubleshoot-review start**
- https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch08s05