Troubleshooting Containerized Applications

Troubleshooting S2i builds and Deployments

After completing this section, you will be:

- Troubleshoot an application build and deployment steps on OpenShift
- Analyze OpenShift logs to identify problems during the build and deploy process

Introduction to the S2I Process

- S2I process is simple way to automatically creates images based on
  a) Programming language
  b) And application source code
- Problems can arise during S2i image creation process
- Important to understand basic workflow for most programming languages supported by OpenShift
- S2i image creation process consists of two major steps:
  a) Build step
  b) Deployment step

S2i image creation process

1. **Build Step**
   Responsible for compiling source code
   Download library dependencies
   Packaging the application as container image
   Push built image to OpenShift internal registry
   Driven by BuildConfig (bc)

2. **Deployment Step**
   Responsible to start a pod and make it available
   Driven by Deployment or DeploymentConfg (dc)

The steps and pods

1. **Build Step**

   Creates build pod named **<application-name>-build-<number>-<string>**

   Execute build steps and saves log

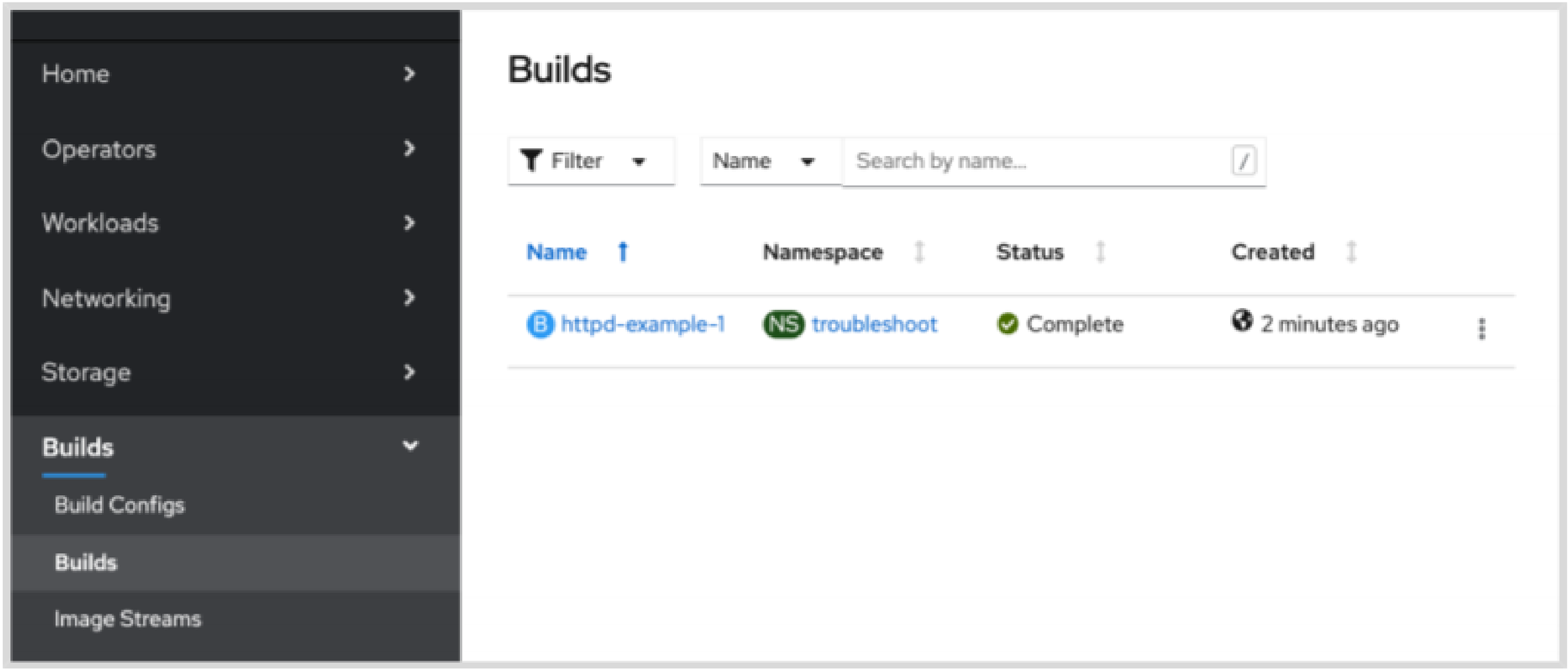
2. **Deployment Step**

   Create separate pod named **<application-name>-<string>**

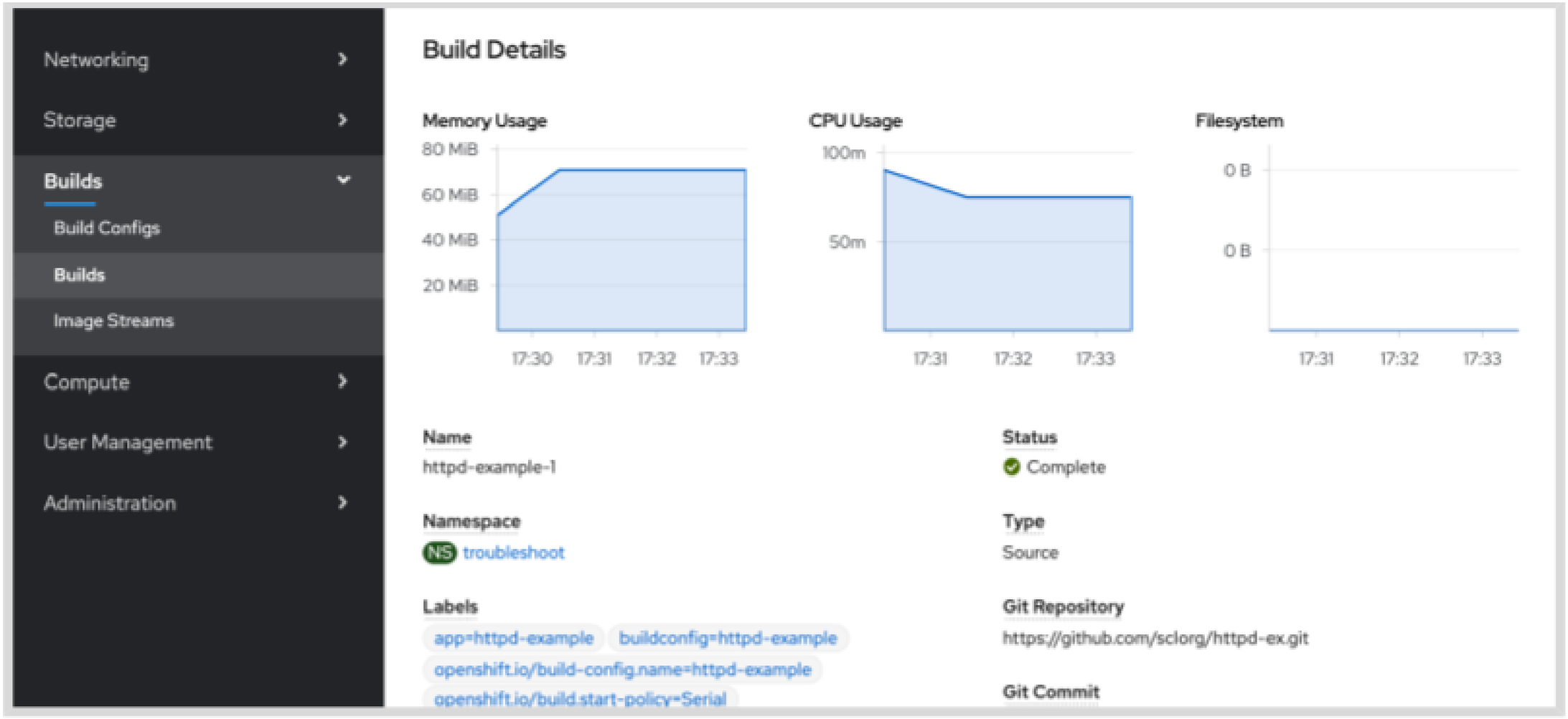
   Execute deployment steps and saves log

NOTE: Deployment step aborts if build fails

Access details of each step in web console



Build instances of a project



Using commands review logs, fix it

- Display build config logs

```
$ oc logs [-f] bc/<application-name>
```

- After found and fix issues, request a new build

```
$ oc start-build <application-name> -F
```

- OpenShift will spawns new pod with above new build process and terminating old pods
- Display deployment logs

```
$ oc logs dc/<application-name>
```

Describing Common Problems

- Permission issues
- Invalid Parameters
- Volume Mount Errors
- Obsolete Images

Troubleshooting Permission issues

- Due to wrong permission, or incorrect environment permissions set by admin
- S2i image enforce non-root access to file system and external resources
- SELinux policies restrict access to file system, network ports and process resources
- Some containers require specific user ID

  But S2i designed to run containers using random user by default OpenShift security policy

Troubleshooting Permission issues

- Take following Dockerfile example with USER instruction

```
FROM ubi7/ubi:7.7
...contents omitted...
RUN chown -R nexus:nexus ${NEXUS_HOME}

USER nexus
WORKDIR ${NEXUS_HOME}

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

Troubleshooting Permission issues

- Attempt to use image generated by previous Dockerfile lead to following error

```
$ 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
...lStateException: Nexus work directory already in use: /opt/nexus/sonatype-work
...output omitted...
```

Troubleshooting Permission issues

- To solve this issue, relax security by mapping default service account to anyuid security context constraints

```
$ oc adm policy add-scc-to-user anyuid –z default
```

- Ensure file system used by container is configured according:
  a) Use chown command to update folder ownership
  b) Use semanage fcontext –at container_file_t <folder> to configure SELinux policy
  c) Use restorecon –R <folder> to refresh SELinux policy

Troubleshooting Invalid Parameters

- Wrongly configured login credentials
- Logs from application pod provide a clear ideas of these problems
- Best practice store configuration parameters in configmaps or secret
- Using ConfigMaps ensure data injected to different container has same environment variables and values

Troubleshooting Invalid Parameters

- Pod resource definition with configured ConfigMap

```
apiVersion: v1
kind: Pod
...output omitted...
spec:
  containers:
    - name: test-container
...output omitted...
  env:
        - name: ENV_1 (1)
          valueFrom:
            configMapKeyRef:
              name: configMap_name1
              key: configMap_key_1
...output omitted...
  envFrom:
        - configMapRef:
            name: configMap_name_2 (2)
...output omitted...
```

Troubleshooting Volume Mount Errors

Common mistake

- Persistent volume configured with local file system.
- Pod might not able to allocate a PVC

To resolve the issue

- Delete PVC and PV

```
$ oc delete pv <pv name>
$ oc create –f <pv_resource_file>
```

- Recreate PV pointing to remote file system : NFS, or cloud storage such as AWS S3, Google PD

Troubleshooting Obsolete Images

- If new image is pushed to registry with same name and tag, image stream may not get updated

```
$ oc import-image <image_name> --from <url> --confirm
```

To resolve this:

- Remove image from each node via

```
$ podman rmi –f <image_id>
```

- Configure automated way to remove obsolete images and other resources

```
$ oc adm prune image
```

Guided Exercise: Troubleshooting an OpenShift Build

In this exercise:

- You will troubleshoot an Openshift build and deployment process
- You should be able to identify and solve the problems raised during build and deployment process of a Node.js application

Troubleshooting Containerized Applications

After completing this section, you should be able to:

- Implement techniques for troubleshooting and debugging containerized applications.
- Use the port-forwarding feature of the OpenShift client tool.
- View container logs.
- View OpenShift cluster events.

Forwarding Ports

- Establish temporary connections into application for troubleshooting
- Case use:
  - ❑ Admin console for database or messaging service
  - ❑ Like SSH access to container to restart terminated services
- Map to current workstation network port
- Doesn’t expose other container’s configuration

- Podman provides port forwarding

```
$ sudo podman run –name db –p 30306:3306 mysql
```

Forwarding Ports - OpenShift

- Forward a local port to a pod port
- Port-forward mapping exists only in the workstation where the oc cmd is run
- Mapping forwards connections to a single pod
- Service maps port for all network users, and load-balances connections to multiple pods

- Forward port 30306 from developer machine to port 3306 on db pod, where a MYSQL server accept network connections

```
$ oc port-forward db 30306 3306
```

- When running this command, be sure to leave the terminal window running. Closing the window or cancelling the process stops the port mapping immediately

Enabling Remote Debugging with Port Forwarding

- Enable developer to debug application directly, looking at line of codes being executed in real time
- Case use:
  a) Jboss Developer Studio to utilize Java Debug Wire Protocol(JDWP) to communicate between debugger and Java Virtual Machine.
  b) Other Integrated Development Environments (IDEs) provides capability to remotely debug an application

Enabling remote debugging on WildFly and Jboss EAP

1. Application server startup – enabling remote debugging

```
JAVA_OPTS="$JAVA_OPTS \
> -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"
```

2. On developer local workstation, execute port-forward command

```
$ oc port-forward jappserver 8787:8787
```

3. Users set breakpoints in their IDE of choice and start debugging their codes

Accessing Container Logs

- View logs in running containers and pods to facilitate troubleshooting
- Neither is aware of application specific logs
- Not all applications configured to send all logging output to standard output
- Container's standard output and standard is saved onto disk as part of ephemeral storage.
- Using following podman or oc commands to reveal logs even after container was stopped but not removed

```
$ sudo podman logs <containername>

$ oc logs <podname> [-c <containerName>]
```

- -c is optional if there is only one container, oc default to only running container

OpenShift Events

- Cluster or Project level logging
- Events signal significant actions such as starting a container or destroying a pod

```
$ oc get event [-A]
```

- Sort event by timestamp

```
$ oc get events –sort-by=‘{.lastTimestamp}’
```

```
LAST SEEN   TYPE     REASON              OBJECT                                  MESSAGE
49s         Normal   Scheduled           pod/rails-postgresql-example-1-build    Successfully assigned my
-build to crc-m89r2-master-0
48s         Normal   Scheduled           pod/postgresql-1-deploy                 Successfully assigned my
-m89r2-master-0
30s         Normal   Scheduled           pod/postgresql-1-lhwhn                  Successfully assigned my
m89r2-master-0
96s         Normal   CreatedSCCRanges    namespace/my-demo1                      created SCC ranges
49s         Normal   DeploymentCreated   deploymentconfig/postgresql             Created new replication
```

Retrieve events relate to an application pod

- Alternative to cluster events, only reveal events related to specific pod

```
$ oc get pods

$ oc describe pod mysql
```

```
Events:
  Type    Reason          Age    From               Message
  ----    ------          ----   ----               -------
  Normal  Scheduled       3d23h  default-scheduler  Successfully assigned my-demo1/ra
  Normal  AddedInterface  3d23h  multus             Add eth0 [10.217.0.90/23]
  Normal  Pulled          3d23h  kubelet            Container image "image-registry.o
gresql-example@sha256:fdac838560e09569bdfe0c168b8123180640a6fe297125b8413fa6feac54372
  Normal  Created         3d23h  kubelet            Created container rails-postgresq
  Normal  Started         3d23h  kubelet            Started container rails-postgresq
```

- Or all events related to deployment

```
$ oc describe { dc | deployment }
```

Accessing Running Containers

- Some troubleshooting scenarios require a more drastic skill
- Like logging directly into specific pod
- Both Podman and OpenShift provide *exec* subcommand

To execute a single interactive command or start a shell, add the **-it** options. The following example starts a Bash shell for the **myhttpdpod** pod:

```
$ oc exec -it myhttpdpod /bin/bash
```

You can use this command to access application logs saved to disk (as part of the container ephemeral storage). For example, to display the Apache error log from a container, run the following command:

```
$ sudo podman exec apache-container cat /var/log/httpd/error_log
```

Overriding Container Binaries

- Most container images doesn't include some basic troubleshooting commands: curl, telnet, netcat, ip, netstat, traceroute, top and etc
- Reason: keep image slim, focus and lightweight
- Resolve this: mount host binaries folders as volume inside container

```
$ sudo podman run –it –v /bin:/bin <image> /bin/bash
```

Note

To access these commands in OpenShift, you need to change the pod resource definition in order to define **`volumeMounts`** and **`volumeClaims`** objects. You also need to create a **`hostPath`** persistent volume.

Alternatively: Build new image with the tools

- Perform yum or apt install on top of base image

```
FROM ubi7/ubi:7.7

RUN yum install -y \
      less \
      dig \
      ping \
      iputils && \
    yum clean all
```

Transferring Files To and Out of Containers

- By mounting volume to container

```
$ sudo podman run –v /conf:/etc/httpd/conf –d do180/apache
```

- Use podman cp to copy files into todoapi container

```
$ sudo podman cp standalone.conf   todoapi:/opt/jboss/standalone/conf/standalone.conf
```

- Use podman cp to copy out files from todoapi container

```
$ sudo podman cp todoapi:/opt/jboss/standalone/conf/standalone.conf  /tmp
```

Transferring Files To and Out of Containers

- Use podman exec and piped output

```
$ sudo podman exec -i mydb  mysql –uroot –proot < /path/on/host/db.sql
```

- Using same concept, retrieve data from running container and save it in host machine

```
$ sudo podman exec -it <containerName> sh \
> -c 'exec mysqldump -h"$MYSQL_PORT_3306_TCP_ADDR" \
> -P"$MYSQL_PORT_3306_TCP_PORT" \
> -uroot -p"$MYSQL_ENV_MYSQL_ROOT_PASSWORD" items'
> db_dump.sql
```

- Use oc rsync command: similar to podman cp under OpenShift pods

Guided Exercise: Configuring Apache Container Logs for Debugging

In this exercise:

- You will configure an Apache httpd container to send logs to standard output
- You will review Podman logs and events

Chapter Summary

In this chapter, you learned:

Access logs for events, warnings, and errors

Use oc logs , oc describe command

Use podman logs command

Use port-forward connect directly to a port n application pod

Using oc get event command