

# Lesson : Manage Non-shared Storage with Stateful Sets

Deploy applications that scale without sharing storage.

# Application Clustering

- Clustering applications requires persistent storage
  - MySQL
  - PostGreSQL
  - Cassandra
- Multiple applications requires access to database system
- Need strong or eventual consistency read/write
- Common setup: Shared storage via NAS

# Network Attached Storage (NAS)

- File-based storage
- Common protocol: NFS and CIFS (SMB)
- Good for data backup, archiving
- Has reliability, fault-tolerant, security, standard protocol, lock handler and caching
- No need for proprietary hardware / software
- Common case uses:
  - Web content
  - File share services
  - FTP storage
  - Backup archives
  - Video streaming, audio streaming
- Main disadvantage: not fit for structural content

# Storage Area Network (SAN)

- Block-level storage
- Mostly required proprietary hardware or connections
- Protocols:
  - FC, FCoE, iSCSI, NVMeoFC, NVMeoTCP
- Present raw device / LUN to hosts
- LUNs sharing; application handle locks and release
- Data protection: RAID configuration
- Common use:
  - SQL Databases (single node access), VM (multimode access)
  - High-performance data access, Server-side processing applications
  - Clustering applications and many more

# Stateless vs Statefull

## Stateless

- Request sent to server and response relayed back without storing any information
- Or Data loss upon user exit/logouts
- Less complex setup
- Typically, slower than Stateful. Because every transaction; recreate, repeat, recalculate

## Statefull

- Store data persistently
- Data can be used/tracked by servers, clients, other applications
- Simplifies data recovery (point in time)
- Requires design to complex setup
- Faster processing in subsequent transactions

# Stateful Sets

- Stateful application simplifies recovery from failures
- **Consistent identities:** Single stable DNS, hostname, storage
- Stateful set guarantees given network identity maps to same storage identity
  - Consistent permission to volumes, network resources
  - Facilitates graceful scaling operations and rolling updates
  - Pods added and removed in predictable order
- Best options for applications (databases) require **consistent identities** and **non-shared persistent storage**

# Differences

- Deployments provides **stateless** representation of set of pods
- StatefulSets provides **stateful** representation of set of pods
- ReplicaSets help **manage traffic** by scaling and synchronize configuration

# Example application

## Database requirements:

mysql-0 : First pod, primary role (read-write) → sit on higher end node

mysql-1 : Read-only replica → sits on lower end node

mysql-2 : Read-only replica → sits on lower end node

- Applications / user request connect to primary role for read-write access
- Event of failure, next replica is promoted to primary role, continue access by application
- When first replica back online, failback can be automated
- If scale down occurs, only read-only replicas removed first

# Create StatefulSets using YAML manifest file

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: dbserver ①
spec:
  selector:
    matchLabels:
      app: database ②
  replicas: 3 ③
  template:
    metadata:
      labels:
        app: database ④
    spec:
      containers:
        - env: ⑤
          - name: MYSQL_USER
            valueFrom:
              secretKeyRef:
                key: user
                name: sakila-cred
        image: registry.ocp4.example.com:8443/redhattraining/mysql-app:v1 ⑥
        name: database ⑦
        ports: ⑧
          - containerPort: 3306
            name: database
        volumeMounts: ⑨
          - mountPath: /var/lib/mysql
            name: data
terminationGracePeriodSeconds: 10
```

1. Name of the stateful set.
- 2,4 Application labels.
- 3 Number of replicas.
- 5 Environment variables (can use secret object).
- 6 Image source.
- 7,8 Container name and ports.
- 9 Mount path information for the persistent volumes for each replica. Each persistent volume has the same configuration.

# Create StatefulSets using YAML manifest file

Create the stateful set by using the `create` command:

```
[user@host ~]$ oc create -f statefulset-dbserver.yml
```

Verify the creation of the stateful set named dbserver:

```
[user@host ~]$ kubectl get statefulset
NAME      READY   AGE
dbserver  3/3    6s
```

Verify the status of the instances:

```
[user@host ~]$ oc get pods
NAME      READY   STATUS    RESTARTS   AGE
dbserver-0 1/1     Running   0          85s
dbserver-1 1/1     Running   0          82s
dbserver-2 1/1     Running   0          79s
```

# Create StatefulSets using YAML manifest file

Verify the status of the persistent volumes:

```
[user@host ~]$ kubectl get pvc
NAME           STATUS    VOLUME          CAPACITY   ACCESS MODES
STORAGECLASS ...
data-dbserver-0  Bound    pvc-c28f61ee-...  1Gi        RWO         nfs-
storage ...
data-dbserver-1  Bound    pvc-ddbe6af1-...  1Gi        RWO         nfs-
storage ...
data-dbserver-2  Bound    pvc-8302924a-...  1Gi        RWO         nfs-
storage ...
```

# Create StatefulSets using YAML manifest file

Notice that three PVCs were created. Confirm that persistent volumes are attached to each instance:

```
[user@host ~]$ oc describe pod dbserver-0
...output omitted...
Volumes:
  data:
    Type:      PersistentVolumeClaim (a reference to
               the same namespace)
    ClaimName: data-dbserver-0
...output omitted...
```

```
[user@host ~]$ oc describe pod dbserver-1
...output omitted...
Volumes:
  data:
    Type:      PersistentVolumeClaim (a reference to
               the same namespace)
    ClaimName: data-dbserver-1
```

```
[user@host ~]$ oc describe pod dbserver-2
...output omitted...
Volumes:
  data:
    Type:      PersistentVolumeClaim (a reference to
               the same namespace)
    ClaimName: data-dbserver-2
```

# StatefulSet lifecycles

You can update the number of replicas of the stateful set by using the `scale` command:

```
[user@host ~]$ oc scale statefulset/dbserver --replicas 1
NAME      READY  STATUS    RESTARTS ...
dbserver-0 1/1    Running   0          ...
```

You can create a stateful set from the web console by clicking the [Workloads > StatefulSets](#) menu. Click [Create StatefulSet](#) and customize the YAML manifest.

# StatefulSet lifecycles

To delete the stateful set, use the `delete statefulset` command:

```
[user@host ~]$ kubectl delete statefulset dbserver
statefulset.apps "dbserver" deleted
```

Notice that the PVCs are not deleted after the execution of the `oc delete statefulset` command:

```
[user@host ~]$ oc get pvc
NAME           STATUS    VOLUME          CAPACITY   ACCESS MODES
STORAGECLASS ...
data-dbserver-0 Bound    pvc-c28f61ee-...  1Gi        RWO          nfs-
storage ...
data-dbserver-1 Bound    pvc-ddbe6af1-...  1Gi        RWO          nfs-
storage ...
data-dbserver-2 Bound    pvc-8302924a-...  1Gi        RWO          nfs-
storage ...
```

## Guided Exercise: Manage Non-shared Storage with Stateful Sets

You should be able to:

- Deploy a web server with persistent storage.
- Add data to the persistent storage.
- Scale the web server deployment and observe the data that is shared with the replicas.
- Create a database server with a stateful set by using a YAML manifest file.
- Verify that each instance from the stateful set has a persistent volume claim.

## Lab: Manage Storage for Application Configuration and Data

You should be able to:

- Deploy a database server.
- Deploy a web application.
- Create a secret that contains the database server credentials.
- Create a configuration map that contains an SQL file.
- Add and remove a volume on the database server and the web application.
- Expose the database server and the web application.
- Scale up the web application.
- Mount the configuration map as a volume.

# Quiz 1

Which method for creating container images is recommended by the containers community?

- a) Run commands inside a basic OS container, commit the container, and save or export it as a new container image.
- b) Run commands from a Dockerfile and push the generated container image to an image registry.
- c) Create the container image layers manually from tar files.
- d) Run the podman build command to process a container image description in YAML format.

# Quiz 1

Which method for creating container images is recommended by the containers community?

- a) Run commands inside a basic OS container, commit the container, and save or export it as a new container image.
- b) Run commands from a Dockerfile and push the generated container image to an image registry.
- c) Create the container image layers manually from tar files.
- d) Run the podman build command to process a container image description in YAML format.

# Chapter Summary

In this chapter, you learned:



Dockerfile contains instructions that specify how to construct a container image.



The Source-to-Image (S2I) process provides an alternative to Dockerfiles. S2I implements a standardized container image build process for common technologies from application source code. This allows developers to focus on application development and not Dockerfile development.



Container images provided by Red Hat Container Catalog or Quay.io are a good starting point for creating custom images for a specific language or technology.



Building an image from a Dockerfile using a three-step process