Table of Contents

Controllers Lab

In this lab, you explore more complex deployment types. You set up a blue-green deployment to examine deploying a new version of the application without impacting an existing, running application. Then you configure health checks for the application.

Next, you configure sidecar containers in some of your pods to protect the application behind an `oauth-proxy`.

Finally, you create a StatefulSet to experience how to set up an application that has state requirements.

Goals

- Explore blue-green deployment
- Configure health checks
- Configure sidecar containers
- Explore StatefulSets

1. Provision Lab Environment

If you previously set up an environment for this class, you can skip this section and go directly to the Explore Blue-Green Deployment section.

In this section, you provision the lab environment to provide access to all of the components required to perform the labs, including access to the OPENTLC OpenShift Container Platform portal (https://master.na37.openshift.opentlc.com/). The lab environment is a shared cloud-based environment, so that you can access it over the Internet from anywhere. However, do not expect performance to match a dedicated environment.

1. Go to the OPENTLC lab portal (https://labs.opentlc.com/) and use your OPENTLC credentials to log in.

   If you do not remember your password, go to the OPENTLC Account Management page (https://www.opentlc.com/account/) to reset your password.

2. Navigate to **Services** → **Catalogs** → **All Services** → **OPENTLC OpenShift Labs**.
3. On the left, select **OPENTLC OpenShift 3.7 Shared Access**.
4. On the right, click **Order**.
5. On the bottom right, click **Submit**.

   Do not select **App Control** → **Start** after ordering the lab. You are simply requesting access to the shared environment so there is nothing to start.

   - After a few minutes, expect to receive an email with instructions on how to connect to the environment.
6. While you are waiting for the environment to build, read the email carefully and specifically note the following:
   - The email includes a URL for the OpenShift master similar to `https://master.na37.openshift.opentlc.com`.
     - This URL varies based on the region where you are located—in this example, the region is `na37`. Whenever you see "${region}" in a URL from now on, make sure to replace it with the region that your environment was provisioned in.

1.1. Deploy Client

In the labs for this course, you use the command line for some of the tasks. The labs assume that you are using the provided client VM. You may use your own laptop or workstation for this if you are able to install the required software. This is recommended only for experienced users, because it may cause technical issues due to differences between local and virtual environments.

In this section, you deploy the client VM for this course.

1. Go to the OPENTLC lab portal (https://labs.opentlc.com/) and use your OPENTLC credentials to log in.
2. Navigate to **Services** → **Catalogs** → **All Services** → **OPENTLC OpenShift Labs**.
3. On the left, select **OpenShift 3.7 Client VM**.
4. On the right, click **Order**.
5. On the bottom right, click **Submit**.

   Do not select **App Control** → **Start** after ordering the client VM. This is not necessary in this environment. Selecting **Start** may corrupt the lab environment or cause other complications.

   - After a few minutes, expect to receive an email with instructions on how to connect to the environment.
6. Read the email carefully and specifically note the following:
   - The email includes a hostname for your Client VM.
   - Remember this hostname. It will look something like (where `xxxx` is a unique GUID for your Client VM):

```
ocplab-xxxx-oslab.opentlc.com
```

1.2. Start Client After Shut Down

To conserve resources, the client VM shuts down automatically after eight hours. In this section, you restart the client VM for this course after it has shut down automatically.

1. Go to the OPENTLC lab portal (https://labs.opentlc.com/) and use your OPENTLC credentials to log in.
2. Navigate to **Services** → **My Services** (this should be the screen shown right after logging in).
3. In the list of your services, select your client VM.
4. Select **App Control** → **Start** to start your client VM.
5. Select **Yes** at the **Are you sure?** prompt.
6. On the bottom right, click **Submit**.

After a few minutes, expect to receive an email letting you know that the Client VM has been started.

1.3. Share Public Key with OPENTLC

To access any of your lab systems via SSH, you must use your personal OPENTLC SSO username and public SSH key.

If you have not already done so, you must provide a public SSH key to the OPENTLC authentication system:

1. Go to the OPENTLC Account Management page (https://www.opentlc.com/account/).
2. Click **Update SSH Key** and log in using your OPENTLC credentials.
3. Paste your public key in that location.

For more information on generating SSH keys, see Setting Up an SSH Key Pair (https://www.opentlc.com/ssh.html).

1.4. Test Server Connections

The `ocplab` host—also called the client VM—serves as an access point into the environment and is not part of the OpenShift environment.

1. Connect to your client VM:

```
ssh -i ~/.ssh/yourprivatekey.key opentlc-
username@ocplab-$GUID.oslab.opentlc.com
```

- Remember to replace `$GUID` with your unique GUID from the welcome e-mail.
- Red Hat recommends that you create an environment variable so that you do not have to type it each time.

Example

```
Laptop$ export GUID=c3po
Laptop$ ssh -i ~/.ssh/mykey.key wkulhane-
redhat.com@ocplab-$GUID.oslab.opentlc.com
(root password is: r3dh4t1!)
```

1.5. Connect to OpenShift Cluster

Once you are connected to your client VM, you can log in to the OpenShift cluster.

1. Use the `oc login` command to log in to the cluster, making sure to replace the URL with your cluster's URL:

   ```
   oc login -u <Your OPENTLC UserID> -p <Your OPENTLC Password>
   https://master.na37.openshift.opentlc.com
   ```

   If you see a warning about an insecure certificate, type `y` to connect insecurely.

   Sample Output

   ```
   Login successful.

   You don't have any projects. You can try to create a new project, by running

       oc new-project <projectname>
   ```

2. Explore Blue-Green Deployment

Using blue-green deployment, you serve one application at a time and switch from one application to the other. Blue-green deployment is especially useful when you are deploying a new version of an application that requires a `Recreate` deployment strategy and cannot experience any downtime. It is also useful when you want to test the application in a production environment before actually routing live traffic to it.

In this lab, you use a simple PHP application to explore blue-green deployments. You configure the application via an environment variable to display pictures of cats, cities, or pets (cats and other animals).

You simulate blue-green deployments by deploying one version of the application displaying cats. Then you spin up a new version of the application using a different selector environment variable to show cities. Once the new instance is running, you move traffic over to it.

You use the `cotd` application located at https://github.com/wkulhanek/cotd.git (https://github.com/wkulhanek/cotd.git). You can set up an application like this:

```
oc new-app --name='blue' --labels=name="blue"
php~https://github.com/wkulhanek/cotd.git --env=SELECTOR=cats
```

`SELECTOR` can be either `cats`, `cities`, or `pets`. However, using `pets` is discouraged because `pets` includes cats among other animals and makes it difficult to distinguish between the `cats` and `pets` versions of the application.

2.1. Set Up Blue-Green Deployment

1. Create a new project named `xyz-deployments` with display name `XYZ Deployments`, replacing `xyz` with your initials.

   ```
   oc new-project xyz-deployments --display-name "XYZ Deployments"
   ```

2. Set up one `cotd` application serving cats (the blue application).

   ```
   oc new-app --name='blue' --labels=name=blue
   php~https://github.com/wkulhanek/cotd.git --env=SELECTOR=cats
   ```

3. Expose the service that was created as route `bluegreen`.

   ```
   oc expose svc/blue --name=bluegreen
   ```

4. In a second terminal window, start a `curl` loop to your application to verify that it is working and serving up cat pictures:

   ```
   while true; do curl -s $(oc get route bluegreen --template='{{ .spec.host
   }}')/item.php | grep "data/images" | awk '{print $5}'; sleep 1; done
   ```

2.2. Execute Blue-Green Deployment

1. Set up another `cotd` application serving cities (the green application).

   ```
   oc new-app --name='green' --labels=name=green
   php~https://github.com/wkulhanek/cotd.git --env=SELECTOR=cities
   ```

2. Once the application is running, move traffic over to the new application instance by updating the `bluegreen` route (watch the `curl` terminal window to see it change).

   - Use the `oc patch` command to update the `bluegreen` route to point to the `green` service.

     ```
     oc patch route/bluegreen -p '{"spec":{"to":{"name":"green"}}}'
     ```

3. Observe in your second terminal window that the application now serves pictures of cities.

4. Execute another blue-green deployment by modifying the blue application to show pets.

- You need to update the `SELECTOR` environment variable of your blue application.
  - This is a configuration change, so wait until the application has redeployed.

    ```
    oc set env dc/blue SELECTOR=pets
    ```

5. This time, add both services to the route and set the weights of the services accordingly (like in an A/B deployment).
   - Initially, set the currently active green application to a weight of `100` and the currently inactive blue application to a weight of `0`.

     ```
     oc set route-backends bluegreen green=100 blue=0
     ```

6. Keep `curl` running and observe that the previous change did not change the data that the application serves—it is still serving images of cities.
7. Now execute the blue-green switch by simply adjusting the weights of the `bluegreen` route to set the green application to `0` and the blue application to `100`.

   ```
   oc set route-backends bluegreen green=0 blue=100
   ```

8. Observe that the second terminal window now shows the application serving images of pets.

3. Explore Health Checks

The `cotd` application does not include any health checks, so you need to add health checks to both the `blue` and the `green` applications.

1. Think about the following:
   - How do you determine that the application is ready?
   - How do you determine that the application is still alive?
   - How long should the initial timeout be? Does there need to be one?
   - How often should the probe be checked? Do you need to set this up?
2. For the `blue` and `green` applications, set up both a readiness probe and a liveness probe.

   ```
   oc set probe dc/green --readiness --get-url=http://:8080/item.php --initial-
   delay-seconds=2
   oc set probe dc/blue --readiness --get-url=http://:8080/item.php --initial-
   delay-seconds=2

   oc set probe dc/green --liveness --get-url=http://:8080/item.php --initial-
   delay-seconds=2
   oc set probe dc/blue --liveness --get-url=http://:8080/item.php --initial-
   delay-seconds=2
   ```

4. Configure Sidecar Containers

Sometimes you need to configure a sidecar container in a pod. Examples include:

- Containers that write the logs to a file. A sidecar container can read that log file and publish it to **`stdout`** so that the OpenShift logging infrastructure can pick up the logs.
- Containers that do not have their own authentication mechanism, but must not be accessible to everyone. You can use an **`oauth-proxy`** container to hide this container behind an authentication screen.

In this exercise, you use a simple container that writes the date and time to a log file every five seconds. The log file location is **`/tmp/datelog.txt`**. Then you add a second container to this pod that reads the file and outputs it to **`stdout`**.

4.1. Set Up Project

1. Create a new project named **`xyz-logging`** with display name **`XYZ Logging`**, replacing **`xyz`** with your initials.

```
oc new-project xyz-logging --display-name "XYZ Logging"
```

4.2. Deploy Container

In this section, you deploy a container that logs to a file.

- You use a simple image that writes the current date into a file every five seconds. This image is built from the following **`Dockerfile`**:

```
FROM docker.io/centos:7
COPY ./root /
ENTRYPOINT ["/usr/bin/writelog"]
```

- **`/usr/bin/writelog`** has the following content:

```
#!/bin/bash
while [ true ]; do
  date >>/tmp/datelog.txt
  sleep 5
done
```

1. Create an application using the **`docker.io/wkulhanek/logtofile:latest`** image.

```
oc new-app --docker-image=docker.io/wkulhanek/logtofile:latest
```

2. When the application is running, check the logs of the running pod and confirm that there are no logs available.

```
oc logs -f logtofile-1-rcw78
```

4.3. Deploy Logging Sidecar Container

The easiest way to create a logging sidecar is to use the **`docker.io/busybox:latest`** container. This container can take a few arguments that it then executes.

1. Set your arguments to the container as follows:
   - **`/bin/sh`**
   - **`-c`**
   - **`......`** any command you choose, to read the log file and write it to **`stdout`**.
2. Add this container definition to your **`logtofile`** deployment configuration in the **`containers:`** section:

   ```
   oc edit dc logtofile
   ```

   ```
   - name: logging-sidecar
     image: busybox
     args: [/bin/sh, -c, 'sleep 5 && tail -n+1 -f /tmp/datelog.txt']
   ```

3. Note that the logging sidecar container fails, indicating that it cannot open the **`/tmp/datelog.txt`** file.
   - This is because every container has its own file system. In order to have the logging sidecar container read the log file that the other container writes, both containers need a volume mount that points to a shared volume. Remember that volumes are per pod and not per container—and containers can share a volume that has been defined in a pod.
4. Make sure to add a volume mount pointing to **`/tmp`** to both containers and create an **`EmptyDir`** volume that links the file system for both containers.
   - Expect your final deployment configuration pod template to look similar to this:

```
template:
  [...]
  spec:
    containers:
    - args:
      - /bin/sh
      - -c
      - sleep 5 && tail -n+1 -f /tmp/datelog.txt
      image: docker.io/busybox:latest
      imagePullPolicy: Always
      name: logging-sidecar
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /tmp
        name: tmp
    - image:
docker.io/wkulhanek/logtofile@sha256:3b696d63235007e9b018ec2f20f3f6c5553842e
cdbcb7065966cf9b9cb72a7c0
      imagePullPolicy: Always
      name: logtofile
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /tmp
        name: tmp
    [...]
    volumes:
    - emptyDir: {}
      name: tmp
```

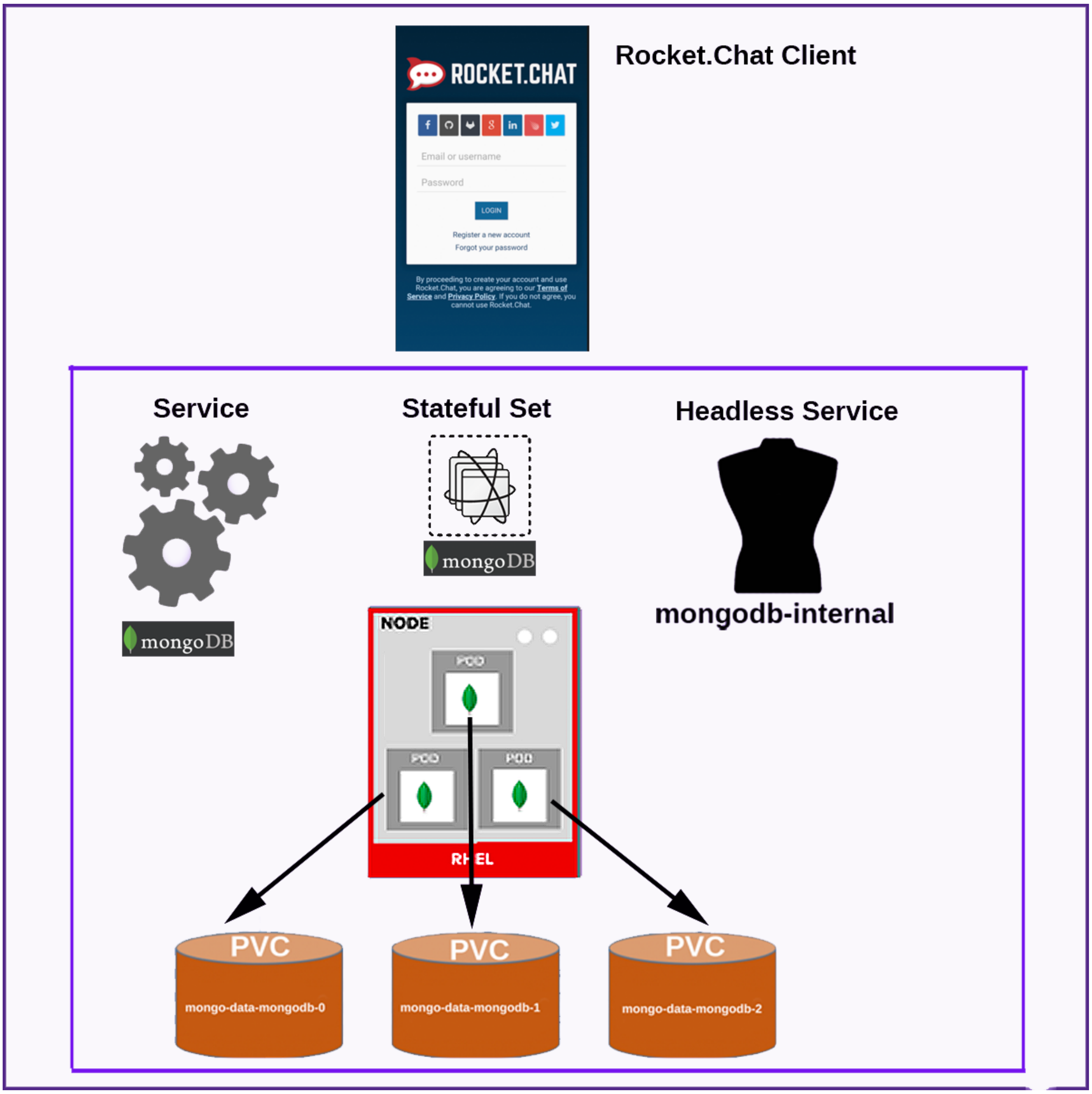
5. In the web console, examine the pod, look at the pod logs, and open the log archive in Kibana.
   - Expect to see in Kibana that the logs from both containers are grouped under a pod using a search query like `kubernetes.pod_name:"logtofile-5-pqs7q" AND kubernetes.namespace_name:"xyz-logging"`.

5. Explore StatefulSets

Some applications require more capabilities than a deployment (configuration), replica set, or daemon set can provide. All of these objects are really useful for stateless applications. But for applications that require some predictability, these objects do not offer enough. This is where a StatefulSet comes into play.

In this exercise, you set up a StatefulSet for MongoDB that consists of three copies of the MongoDB database that replicate the data amongst themselves. This requires three pods and three persistent volume claims. It also requires a headless service for the pods of the set to communicate, as well as a regular service for clients to connect to the database.

Once MongoDB is running, you deploy the Rocket.Chat server to illustrate how to use the database. This diagram depicts the back-end architecture:



There currently is no information about StatefulSets in the OpenShift documentation. Instead, consult the Kubernetes v1.7 documentation (https://v1-7.docs.kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set).

5.1. Set Up Project

1. Create a new project named `xyz-rocket` with display name `XYZ Rocket Chat`, replacing `xyz` with your initials.

```
oc new-project xyz-rocket --display-name "XYZ Rocket Chat"
```

5.2. Set Up Services

In this section, you create two services.

1. Create the internal headless service that the pods in the StatefulSet use to communicate with each other.
   - Set the name of the service to `mongodb-internal`.
   - Set the `ClusterIP` to `none` in order to make it headless.
   - It must have the annotation `service.alpha.kubernetes.io/tolerate-unready-endpoints: "true"` for MongoDB to properly come up.
   - The port to connect to is `27017`, the standard MongoDB port.
   - It needs a selector, `name: "mongodb"`, which is used to determine which pods to route traffic to.
   - Finally, create a YAML/JSON file with the service definition, and use `oc create -f <servicedefinition>.yaml` to create the service.

```
echo 'kind: Service
apiVersion: v1
metadata:
  name: "mongodb-internal"
  labels:
    name: "mongodb"
  annotations:
    service.alpha.kubernetes.io/tolerate-unready-endpoints: "true"
spec:
  clusterIP: None
  ports:
    - name: mongodb
      port: 27017
  selector:
    name: "mongodb"' | oc create -f -
```

2. Create the regular service that database clients use to connect to the database.
   - Set the name of the service to `mongodb`.
   - The port to connect to is `27017`, the standard MongoDB port.
   - It needs a selector, `name: "mongodb"`, which is used to determine which pods to route traffic to.
   - Finally, create a YAML/JSON file with the service definition, and use `oc create -f <servicedefinition>.yaml` to create the service.

```
echo 'kind: Service
apiVersion: v1
metadata:
  name: "mongodb"
  labels:
    name: "mongodb"
spec:
  ports:
    - name: mongodb
      port: 27017
  selector:
    name: "mongodb"' | oc create -f -
```

5.3. Create StatefulSet for MongoDB Database

1. Create the StatefulSet for the MongoDB database.
   - You need to create a YAML/JSON file with the StatefulSet definition and use `oc create -f <statefulset>.yaml` to create the StatefulSet.
   - Make sure to use `apiVersion: apps/v1beta1` for OpenShift 3.7.
   - You need three replicas (pods).
   - The pods need the label `name=mongodb` for your services to find them.
   - You can use the MongoDB container image from Red Hat Software Collections: `registry.access.redhat.com/rhscl/mongodb-32-rhel7:3.2`.
     - This container image can be run both standalone (the default) and as a MongoDB replica set.
     - To run this container with replication enabled, add a startup argument: `run-mongodb-replication`.
     - This container listens on port `27017`.
     - It needs a volume mount defined for the `/var/lib/mongodb/data` path.
     - The image expects configuration as environment variables—you can either specify the variables directly (easier) or add all required values into a secret and reference the appropriate fields in the secret. Feel free to use values other than the suggested ones—just make sure to use the same values when connecting to the database.
       - `MONGODB_DATABASE` = `mongodb`
       - `MONGODB_USER` = `mongodb_user`
       - `MONGODB_PASSWORD` = `mongodb_password`
       - `MONDODB_ADMIN_PASSWORD` = `mongodb_admin_password`
       - `MONGODB_REPLICA_NAME` = `rs0` (Do not change this.)
       - `MONGODB_KEYFILE_VALUE` = `12345678901234567890` (Randomly generated from a secret would be better.)
       - `MONGODB_SERVICE_NAME` = `mongodb-internal` (Your headless service name.)
     - The container needs a readiness probe to tell OpenShift when it is successfully started and it is safe to start the next pod.
     - The startup script writes a `/tmp/initialized` file when the database is running.
     - You can use the `stat /tmp/initialized` command for the probe.

- The pods need a `volumeClaimTemplate` to define the PVCs to attach to the individual pods.
  - Remember that in a `StatefulSet` the same PVC gets attached to the same pod every single time—therefore all PVCs need to be identical and created via a volume claim template.
  - The name of the `volumeClaimTemplate` needs to match the name of the `volumeMount` of your pod definition.
  - Set `accessModes` to `ReadWriteOnce` because each PVC can be attached to exactly one pod—otherwise database corruption occurs.

```
echo 'kind: StatefulSet
apiVersion: apps/v1beta1
metadata:
  name: "mongodb"
spec:
  serviceName: "mongodb-internal"
  replicas: 3
  template:
    metadata:
      labels:
        name: "mongodb"
    spec:
      containers:
        - name: mongo-container
          image: "registry.access.redhat.com/rhscl/mongodb-32-rhel7:3.2"
          ports:
            - containerPort: 27017
          args:
            - "run-mongod-replication"
          volumeMounts:
            - name: mongo-data
              mountPath: "/var/lib/mongodb/data"
          env:
            - name: MONGODB_DATABASE
              value: "mongodb"
            - name: MONGODB_USER
              value: "mongodb_user"
            - name: MONGODB_PASSWORD
              value: "mongodb_password"
            - name: MONGODB_ADMIN_PASSWORD
              value: "mongodb_admin_password"
            - name: MONGODB_REPLICA_NAME
              value: "rs0"
            - name: MONGODB_KEYFILE_VALUE
              value: "12345678901234567890"
            - name: MONGODB_SERVICE_NAME
              value: "mongodb-internal"
          readinessProbe:
            exec:
              command:
                - stat
                - /tmp/initialized
  volumeClaimTemplates:
    - metadata:
        name: mongo-data
        labels:
          name: "mongodb"
      spec:
        accessModes: [ ReadWriteOnce ]
```

```
          resources:
            requests:
              storage: "4Gi"' | oc create -f -
```

2. Once you create the StatefulSet, watch the pods as they come up. It may take a few minutes for each pod to switch from `ContainerCreating` to `Running`.
3. When a pod is running, check the pod logs to make sure everything looks correct.
4. On the first pod, (`mongodb-0`), expect to see entries like the following, indicating that this is the first pod in the set of replicas:

```
[...]
018-02-14T19:38:31.787+0000 I CONTROL  [initandlisten] options: { config:
"/etc/mongod.conf", net: { port: 27017 }, replication: { oplogSizeMB: 64,
replSet: "rs0" }, security: { keyFile: "/var/lib/mongodb/keyfile" }, s
torage: { dbPath: "/var/lib/mongodb/data", wiredTiger: { engineConfig: {
cacheSizeGB: 1 } } }, systemLog: { quiet: true } }
2018-02-14T19:38:31.816+0000 I STORAGE  [initandlisten] wiredtiger_open
config: create,cache_size=1G,session_max=20000,eviction=
(threads_max=4),config_base=false,statistics=(fast),log=
(enabled=true,archive=true,path=jou
rnal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=
(wait=60,log_size=2GB),statistics_log=(wait=0),
[...]
=> [Wed Feb 14 19:38:32] Initiating MongoDB replica using: {_id: 'rs0',
members: [{_id: 0, host: 'mongodb-0.mongodb-internal.xyz-
mongodb.svc.cluster.local'}]}
[...]
2018-02-14T19:38:33.219+0000 I REPL     [ReplicationExecutor] New replica set
config in use: { _id: "rs0", version: 1, protocolVersion: 1, members: [ { _id:
0, host: "mongodb-0.mongodb-internal.xyz-mongodb.svc.cluster.l
ocal:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority:
1.0, tags: {}, slaveDelay: 0, votes: 1 } ], settings: { chainingAllowed: true,
heartbeatIntervalMillis: 2000, heartbeatTimeoutSecs: 10, elect
ionTimeoutMillis: 10000, getLastErrorModes: {}, getLastErrorDefaults: { w: 1,
wtimeout: 0 }, replicaSetId: ObjectId('5a849039c701080051c7f3d4') } }
2018-02-14T19:38:33.219+0000 I REPL     [ReplicationExecutor] This node is
mongodb-0.mongodb-internal.xyz-mongodb.svc.cluster.local:27017 in the config
[...]
Successfully added user: { "user" : "mongo_user", "roles" : [ "readWrite" ] }
```

5. On the second (`mongodb-1`) and third (`mongodb-2`) pods, expect to see log entries like the following, indicating that they are joining the replica set:

```
[...]
=> [Wed Feb 14 19:41:00] Adding mongodb-1.mongodb-internal.xyz-
mongodb.svc.cluster.local to replica set ...
2018-02-14T19:41:00.272+0000 I NETWORK  [thread1] Starting new replica set
monitor for rs0/10.1.2.204:27017,10.1.8.115:27017
2018-02-14T19:41:00.272+0000 I NETWORK  [ReplicaSetMonitorWatcher] starting
2018-02-14T19:41:00.274+0000 I NETWORK  [thread1] changing hosts to
rs0/mongodb-0.mongodb-internal.xyz-mongodb.svc.cluster.local:27017 from
rs0/10.1.2.204:27017,10.1.8.115:27017
Cannot use 'commands' readMode, degrading to 'legacy' mode
2018-02-14T19:41:00.324+0000 I ASIO     [NetworkInterfaceASIO-Replication-0]
Connecting to mongodb-0.mongodb-internal.xyz-mongodb.svc.cluster.local:27017
=> [Wed Feb 14 19:41:00] Waiting for PRIMARY/SECONDARY status ...
2018-02-14T19:41:00.362+0000 I ASIO     [NetworkInterfaceASIO-Replication-0]
Successfully connected to mongodb-0.mongodb-internal.xyz-
mongodb.svc.cluster.local:27017
2018-02-14T19:41:00.477+0000 I REPL     [replExecDBWorker-2] Starting
replication applier threads
2018-02-14T19:41:00.477+0000 W REPL     [rsSync] did not receive a valid
config yet
2018-02-14T19:41:00.477+0000 I REPL     [ReplicationExecutor] New replica set
config in use: { _id: "rs0", version: 2, protocolVersion: 1, members: [ { _id:
0, host: "mongodb-0.mongodb-internal.xyz-mongodb.svc.cluster.l
ocal:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority:
1.0, tags: {}, slaveDelay: 0, votes: 1 }, { _id: 1, host: "mongodb-1.mongodb-
internal.xyz-mongodb.svc.cluster.local:27017", arbiterOnly: fals
e, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0,
votes: 1 } ], settings: { chainingAllowed: true, heartbeatIntervalMillis:
2000, heartbeatTimeoutSecs: 10, electionTimeoutMillis: 10000, getLa
stErrorModes: {}, getLastErrorDefaults: { w: 1, wtimeout: 0 }, replicaSetId:
ObjectId('5a849039c701080051c7f3d4') } }
2018-02-14T19:41:00.477+0000 I REPL     [ReplicationExecutor] This node is
mongodb-1.mongodb-internal.xyz-mongodb.svc.cluster.local:27017 in the config
2018-02-14T19:41:00.477+0000 I REPL     [ReplicationExecutor] transition to
STARTUP2
2018-02-14T19:41:00.478+0000 I REPL     [ReplicationExecutor] Member mongodb-
0.mongodb-internal.xyz-mongodb.svc.cluster.local:27017 is now in state PRIMARY
[...]
```

6. Once all three pods are up, examine the created PVCs:

```
oc get pvc
```

7. Scale the StatefulSet up to five replicas.

```
oc scale statefulset mongodb --replicas=5
```

8. Once the StatefulSet has created the five replicas, scale it back down to three replicas.

```
oc scale statefulset mongodb --replicas=3
```

9. Once the StatefulSet is back down to three replicas, examine the PVCs again.
   - Note that there are still five PVCs available. This is because OpenShift keeps these PVCs around in case you want to scale back up—and the exact same PVC attaches to each pod again.

Your MongoDB application is now ready for use.

5.4. Deploy Rocket.Chat as MongoDB Client

In order to test that the database works, you deploy the stock Rocket.Chat container connecting it to the database.

The only difference between connecting to a single pod MongoDB database and your StatefulSet is that the client needs to know that it is connecting to a replicated database. This is done by adding **`?replicaSet=<replica_set_name>`** to the end of the connection URL.

1. Deploy Rocket.Chat as a database client to the MongoDB database, making sure to match the user ID, password, and database name to your specific values:

```
oc new-app docker.io/rocketchat/rocket.chat:0.61.0 -e
MONGO_URL="mongodb://mongodb_user:mongodb_password@mongodb:27017/mongodb?
replicaSet=rs0"

oc expose svc/rocketchat
```

2. Follow the logs to make sure the application deployed successfully.
3. When the application is up and running, connect to the route and start using Rocket.Chat.

6. Clean Up Environment

1. Delete the **`xyz-deployment`**, **`xyz-logging`**, and **`xyz-rocket`** projects to free up resources:

```
oc delete project xyz-deployment
oc delete project xyz-logging
oc delete project xyz-rocket
```