Table of Contents

* [Controllers Lab](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_controllers_lab)
* [1. Provision Lab Environment](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_provision_lab_environment)
  + [1.1. Deploy Client VM](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_deploy_client_vm)
  + [1.2. Start Client VM After Shut Down](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_start_client_vm_after_shut_down)
  + [1.3. Share Public Key with OPENTLC](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_share_public_key_with_opentlc)
  + [1.4. Test Server Connections](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_test_server_connections)
  + [1.5. Connect to OpenShift Cluster](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_connect_to_openshift_cluster)
* [2. Explore Blue-Green Deployment](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#labexercises)
  + [2.1. Set Up Blue-Green Deployment](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_set_up_blue_green_deployment)
  + [2.2. Execute Blue-Green Deployment](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_execute_blue_green_deployment)
* [3. Explore Health Checks](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_explore_health_checks)
* [4. Configure Sidecar Containers](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_configure_sidecar_containers)
  + [4.1. Set Up Project](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_set_up_project)
  + [4.2. Deploy Container](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_deploy_container)
  + [4.3. Deploy Logging Sidecar Container](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_deploy_logging_sidecar_container)
* [5. Explore StatefulSets](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_explore_statefulsets)
  + [5.1. Set Up Project](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_set_up_project_2)
  + [5.2. Set Up Services](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_set_up_services)
  + [5.3. Create StatefulSet for MongoDB Database](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_create_statefulset_for_mongodb_database)
  + [5.4. Deploy Rocket.Chat as MongoDB Client](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_deploy_rocket_chat_as_mongodb_client)
* [6. Clean Up Environment](https://www.opentlc.com/labs/ocp_advanced_development/03_1_Controllers_Solution_Lab.html#_clean_up_environment)

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

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>. 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"

1. 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

1. Expose the service that was created as route **bluegreen**.

oc expose svc/blue --name=bluegreen

1. In a second terminal window, start a **curl** loop to your application to verify that it is working and serving up cat pictures (you will see that the URL of the image is similar to **data/images/cats/wellington.jpg**. It is the **cats** part that you need to verify to see that the images are indeed images of cats):

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

1. 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"}}}'

1. Observe in your second terminal window that the application now serves pictures of cities. Again you will see that the URL of the image is similar to **data/images/cities/wellington.jpg**. It is the **cities** part that you need to verify to see that the images are indeed images of cities.
2. 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

1. 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

1. 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.
2. 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

1. 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.
3. oc set probe dc/green --readiness --get-url=http://:8080/item.php --initial-delay-seconds=2
4. oc set probe dc/blue --readiness --get-url=http://:8080/item.php --initial-delay-seconds=2
5. 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

1. 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']

1. 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.
2. 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@sha256:3b696d63235007e9b018ec2f20f3f6c5553842ecdbcb7065966cf9b9cb72a7c0
   * imagePullPolicy: Always
   * name: logtofile
   * resources: {}
   * terminationMessagePath: /dev/termination-log
   * terminationMessagePolicy: File
   * volumeMounts:
   * - mountPath: /tmp
   * name: tmp
   * [...]
   * volumes:
   * - emptyDir: {}

name: tmp

1. 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 v1.9 documentation](https://v1-9.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 -

1. 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/v1** for OpenShift 3.9.
   * Make sure your **spec.selector.matchLabels** matches your **spec.template.metadata.labels** field.
   * Make sure your **spec.serviceName** matches the name of your headless service.
   * 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-34-rhel7:latest**.
     + 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-mongod-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**
       - **MONGODB\_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/v1
     + metadata:
     + name: "mongodb"
     + spec:
     + serviceName: "mongodb-internal"
     + replicas: 3
     + selector:
     + matchLabels:
     + name: mongodb
     + template:
     + metadata:
     + labels:
     + name: "mongodb"
     + spec:
     + containers:
     + - name: mongo-container
     + image: "registry.access.redhat.com/rhscl/mongodb-34-rhel7:latest"
     + 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 -

1. 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**.
2. When a pod is running, check the pod logs to make sure everything looks correct.
3. 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:
4. [...]
5. 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
6. torage: { dbPath: "/var/lib/mongodb/data", wiredTiger: { engineConfig: { cacheSizeGB: 1 } } }, systemLog: { quiet: true } }
7. 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
8. rnal,compressor=snappy),file\_manager=(close\_idle\_time=100000),checkpoint=(wait=60,log\_size=2GB),statistics\_log=(wait=0),
9. [...]
10. => [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'}]}
11. [...]
12. 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
13. ocal:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 } ], settings: { chainingAllowed: true, heartbeatIntervalMillis: 2000, heartbeatTimeoutSecs: 10, elect
14. ionTimeoutMillis: 10000, getLastErrorModes: {}, getLastErrorDefaults: { w: 1, wtimeout: 0 }, replicaSetId: ObjectId('5a849039c701080051c7f3d4') } }
15. 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
16. [...]

Successfully added user: { "user" : "mongo\_user", "roles" : [ "readWrite" ] }

1. 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:
2. [...]
3. => [Wed Feb 14 19:41:00] Adding mongodb-1.mongodb-internal.xyz-mongodb.svc.cluster.local to replica set ...
4. 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
5. 2018-02-14T19:41:00.272+0000 I NETWORK [ReplicaSetMonitorWatcher] starting
6. 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
7. Cannot use 'commands' readMode, degrading to 'legacy' mode
8. 2018-02-14T19:41:00.324+0000 I ASIO [NetworkInterfaceASIO-Replication-0] Connecting to mongodb-0.mongodb-internal.xyz-mongodb.svc.cluster.local:27017
9. => [Wed Feb 14 19:41:00] Waiting for PRIMARY/SECONDARY status ...
10. 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
11. 2018-02-14T19:41:00.477+0000 I REPL [replExecDBWorker-2] Starting replication applier threads
12. 2018-02-14T19:41:00.477+0000 W REPL [rsSync] did not receive a valid config yet
13. 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
14. 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
15. e, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 } ], settings: { chainingAllowed: true, heartbeatIntervalMillis: 2000, heartbeatTimeoutSecs: 10, electionTimeoutMillis: 10000, getLa
16. stErrorModes: {}, getLastErrorDefaults: { w: 1, wtimeout: 0 }, replicaSetId: ObjectId('5a849039c701080051c7f3d4') } }
17. 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
18. 2018-02-14T19:41:00.477+0000 I REPL [ReplicationExecutor] transition to STARTUP2
19. 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

[...]

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

oc get pvc

1. Scale the StatefulSet up to five replicas.

oc scale statefulset mongodb --replicas=5

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

oc scale statefulset mongodb --replicas=3

1. 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:
2. oc new-app docker.io/rocketchat/rocket.chat:0.63.3 -e MONGO\_URL="mongodb://mongodb\_user:mongodb\_password@mongodb:27017/mongodb?replicaSet=rs0"

oc expose svc/rocketchat

1. Follow the logs to make sure the application deployed successfully.
2. 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:
2. oc delete project xyz-deployment
3. oc delete project xyz-logging

oc delete project xyz-rocket

Build Version: 98391d492c70bf60a6900485d3134768c5a5bfd7 : Last updated 2018-07-25 03:37:02 EDT