# Table of Contents



## OpenShift 101 - Developer Operations

# OpenShift Developer Workshop Labs

This application supports the following workshops at the BCDevOps / BC Developers Exchange / BC Gov CSI Lab:

- OpenShift 101 - Developer Operations
- OpenShift 201 - Pipeline your DevOps

# Lab Setup

Prior to working on labs, the Platform Services team will have created 4 projects:

- tooling
- dev
- test
- production

One member from your group has been given administrative rights to the projects and is responsible for adding all other members to each project.

---

Assign yourself a unique name to be used during your application deployments. **Do not copy the application names directly from the lab guides** as you are in a shared environment with the rest of your team.

For example, if the lab says to create an application named `app1-[username]`, I would create an app named `app1-stewartshea`.

# Lab Requirements

These labs will require access to the pathfinder production environment:

- Pathfinder Web Console

The environment can be accessed from a web browser such as Chrome or Firefox. For users that also leverage the CLI, the `oc` binary is available for download here. The current version of OpenShift is 3.10 and should be installed into your PATH.

In addition, developers require GIT locally installed and must have a 2FA device to support access to GitHub.

- To login with the oc utility:
  - Login to the Pathfinder Web Console with your GitHub ID
  - Navigate to the top right corner, select the drop-down from your username, and select `Copy Login Command`
  - Paste the copied command into your terminal of choice

```
oc login https://console.pathfinder.gov.bc.ca:8443 --token=[token]
```

# General OpenShift Components and Tasks

In these labs you will explore builds and deployments within the context of the Pathfinder OpenShift Platform. These labs will focus on a single 2-tier application based on the popular open source chat utility, Rocket.Chat.

The content in the labs will get `slimmer` as labs progress. The goal here is that previous learnings will help you familiarize yourself with the Web Console or CLI utilities.

# Network Security Policy

By default, the new OpenShift namespace has been configured with zero network access.

- Attempt a general build

```
oc new-build https://github.com/ArctiqTeam/random-beer-giphy
```

- Notice that the build will fail
- Create the 3 basic NetworkSecurityPolicy objects
  - Obtain the sample policy

    ```
    wget https://raw.githubusercontent.com/BCDevOps/platform-services/master/security/aporeto/docs/sample/
    quickstart-nsp.yaml
    ```

  - Modify the policy with the appropriate namespace

    ```
    ---
    apiVersion: secops.pathfinder.gov.bc.ca/v1alpha1
    kind: NetworkSecurityPolicy
    metadata:
      name: egress-internet
    spec:
      description: |
        allow the [INSERT NAMESPACE HERE] namespace to talk to the internet.
      source:
        - - $namespace=[INSERT NAMESPACE HERE]
      destination:
        - - ext:network=any
    ---
    apiVersion: secops.pathfinder.gov.bc.ca/v1alpha1
    kind: NetworkSecurityPolicy
    metadata:
      name: intra-namespace-comms
    spec:
      description: |
        allow the [INSERT NAMESPACE HERE] namespace to talk to itself
      source:
        - - $namespace=[INSERT NAMESPACE HERE]
      destination:
        - - $namespace=[INSERT NAMESPACE HERE]
    ---
    apiVersion: secops.pathfinder.gov.bc.ca/v1alpha1
    kind: NetworkSecurityPolicy
    metadata:
      name: int-cluster-k8s-api-comms
    spec:
      description: |
        allow [INSERT NAMESPACE HERE] pods to talk to the k8s api
      destination:
      - - int:network=internal-cluster-api-endpoint
      source:
      - - $namespace=[INSERT NAMESPACE HERE]
    ```

  - Apply the policy again

    ```
    oc apply -f quickstart-nsp.yaml
    ```

- Attempt the build again

```
oc start-build random-beer-giphy
```

- Remove the previous test build

```
oc delete build random-beer-giphy
```

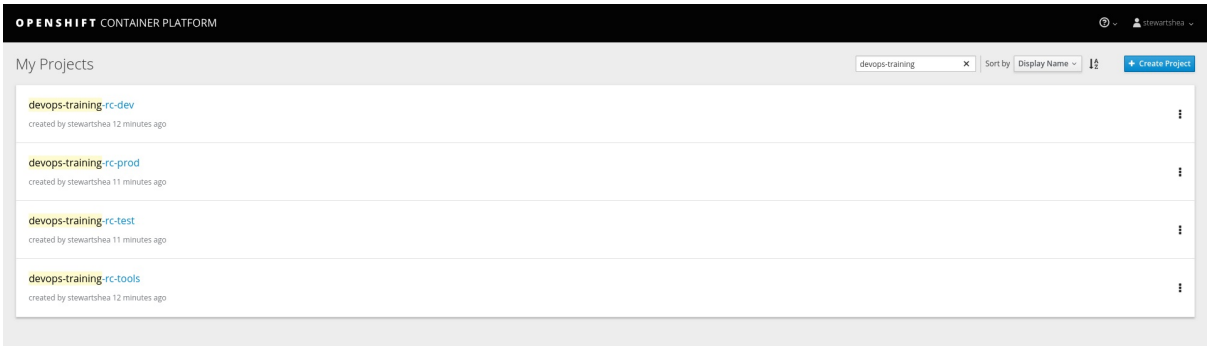
# Builds

In this lab, you will create a simple Docker based build for the Rocket Chat application.

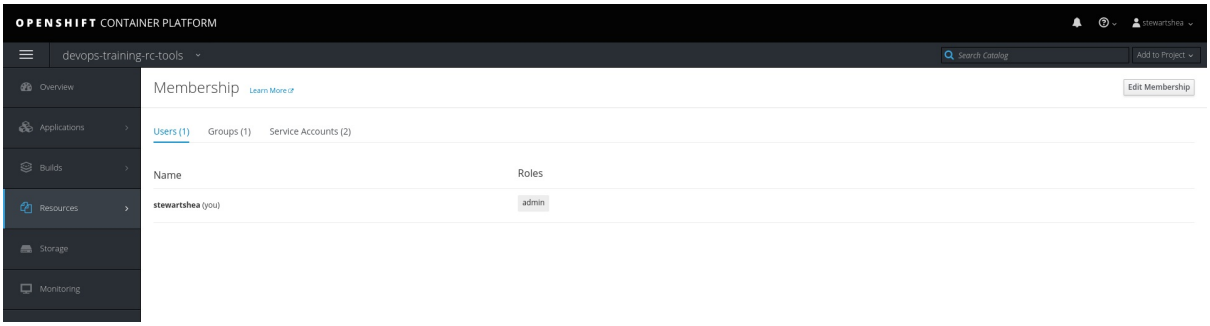
## Team Permissions

Once all projects have been created by the Platform Services team, the team admin must navigate to each project and assign your users the appropriate permissions.

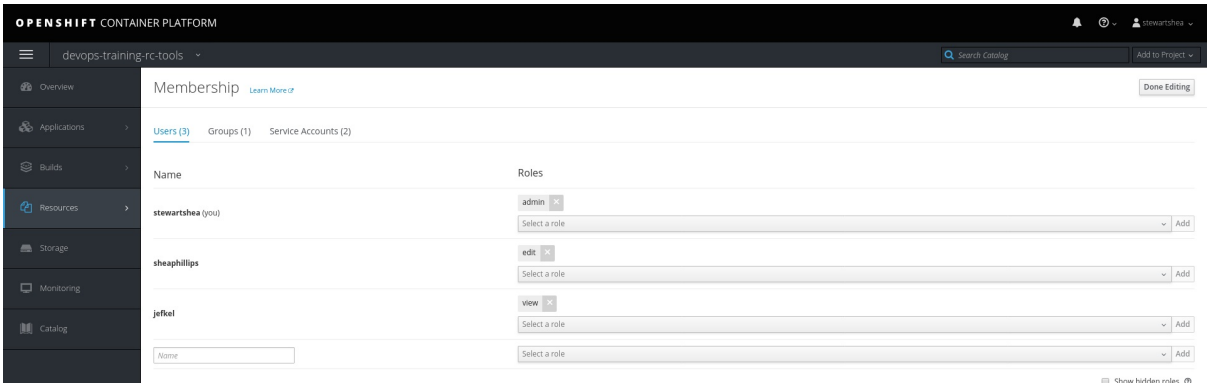
As a team, find each project and add the rest of the team members. Feel free to experiment with the default roles.




- Once in the project, navigate to `Resources -> Membership -> Users`
- Select `Edit Membership`




- Add each user based on their GitHub id




- Select `Done Editing`

This can also be done on the CLI with the `oc` utility:

```
oc policy add-role-to-user [role] [username]
```

## The Tools Project

The tools project is what will hold various support tools for the application. In this case, Jenkins will run here (later), and all builds will run in this project.

## Creating a Docker-Based Build

The Rocket.Chat application build will be based off a minimal Dockerfile in a public repository. Leveraging the commandline, you can use the `oc new-build` command to create all of the necessary OpenShift build components.

Ensure that all team members have edit rights into the project. Once complete, each member can create their own Rocket.Chat docker build.

- To start, switch to your new project. *Ensure that you are in the tools project in this lab*

```
oc project [project-name]
```

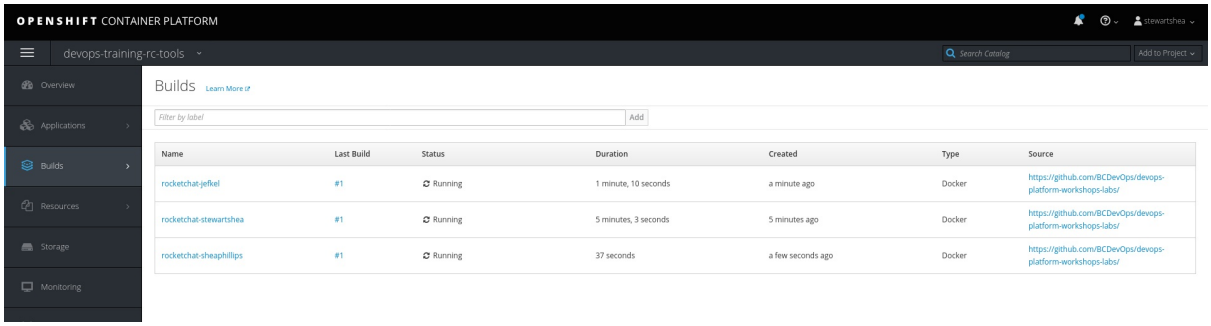
- With the `oc` cli, create the build

```
oc new-build https://github.com/BCDevOps/devops-platform-workshops-labs/ --context-dir=apps/rocketchat --name=r
ocketchat-[username]
```

- The output of the previous command should be similar to the following:

```


--> Found Docker image c71b6c4 (4 weeks old) from registry.access.redhat.com for "registry.access.redhat.com/rh
scl/nodejs-8-rhel7"

    Node.js 8
    ---------
    Node.js 8 available as container is a base platform for building and running various Node.js 8 applications
 and frameworks. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable
network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and effici
ent, perfect for data-intensive real-time applications that run across distributed devices.

    Tags: builder, nodejs, nodejs8

    * An image stream will be created as "nodejs-8-rhel7:latest" that will track the source image
    * A Docker build using source code from https://github.com/BCDevOps/devops-platform-workshops-labs/ will be
 created
      * The resulting image will be pushed to image stream "rocketchat-[username]:latest"
      * Every time "nodejs-8-rhel7:latest" changes a new build will be triggered

--> Creating resources with label build=rocketchat-[username] ...
    imagestream "nodejs-8-rhel7" created
    imagestream "rocketchat-[username]" created
    buildconfig "rocketchat-[username]" created
--> Success
    Build configuration "rocketchat-[username]" created and build triggered.
    Run 'oc logs -f bc/rocketchat-[username]' to stream the build progress.
```

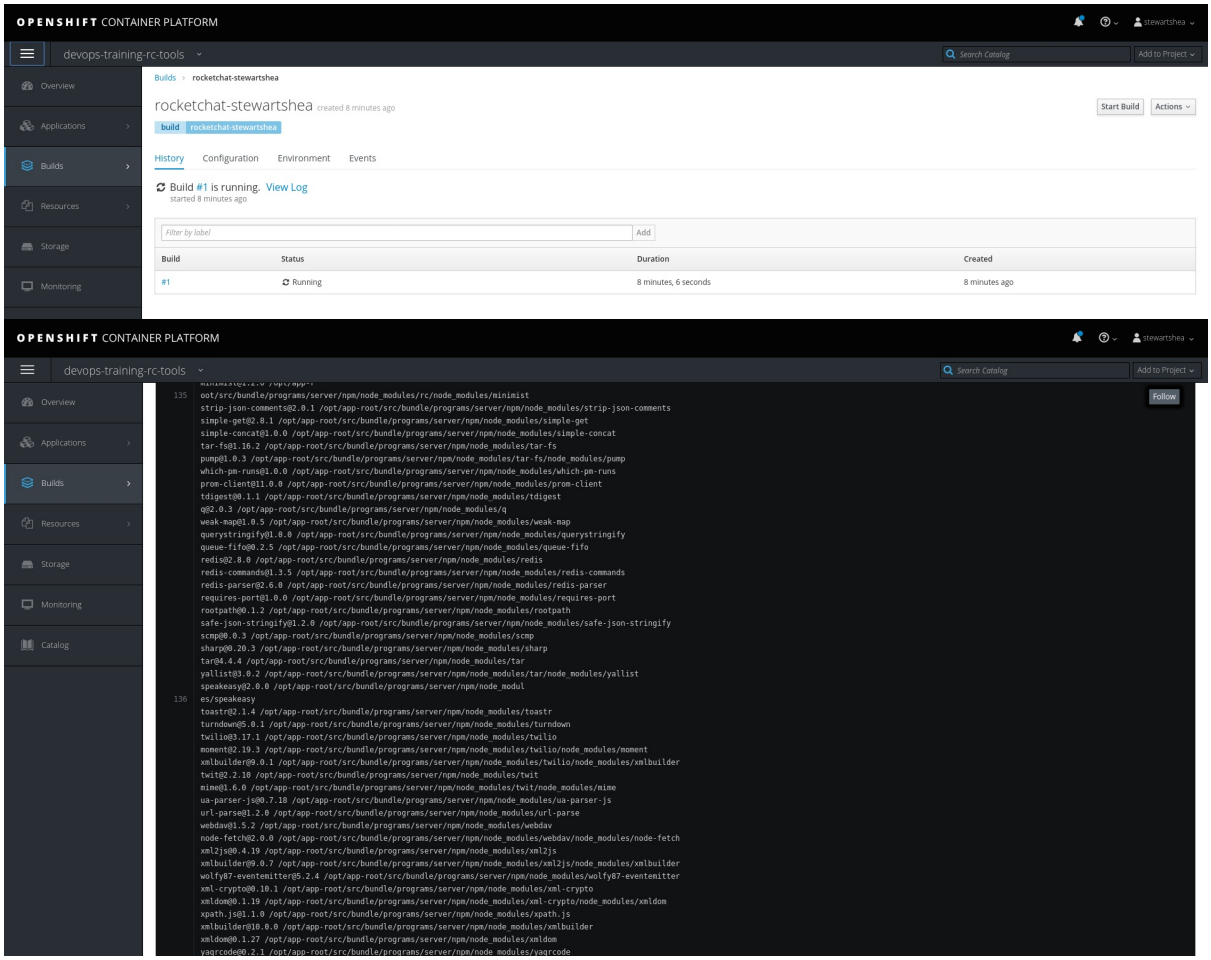
- You can now explore the Web Console to watch the build status from `Builds -> Builds` *note* you will see multiple builds from each team member




- Or this can be done on the CLI

```
oc get bc
oc status
```

- The build status can be monitored from the Web Console by selecting the `View Logs` link



# Interacting with the Registry Console

The registry console provides a web based interface to view the container images created for your projects. It can be used to obtain information on how to push/pull to the registry (from your local machine), as well as setting access rights on your project images.

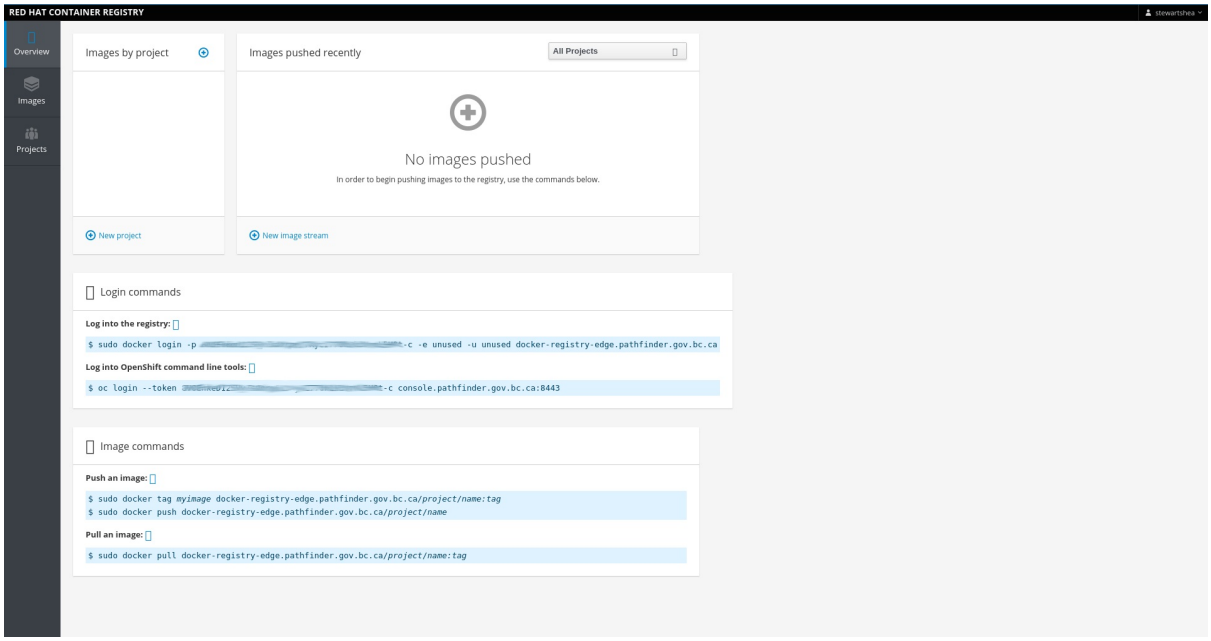
## Accessing the Registry Console

The registry console can be accessed at https://registry-console-default.pathfinder.gov.bc.ca.

- From the browser, navigate to the registry console and login if required

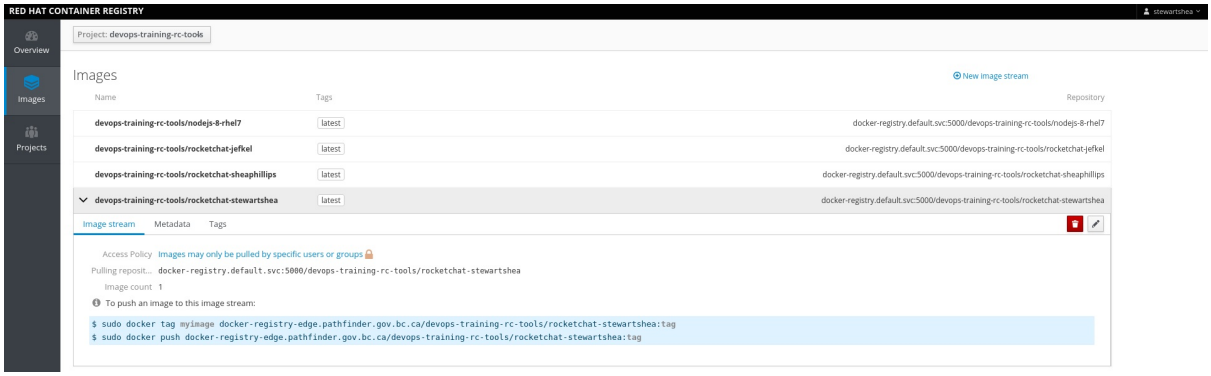
## Viewing the Registry Push/Pull Commands

Navigate to the bottom of the `Overview` tab to find your specific docker login commands. These can be used if you wish to push images into your project space.




## Viewing Image Details

Navigate to the Images tab, select the correct project, and explore the image details. Also nagivate to the Web Console and explore `Builds -> Images` to see similar details.



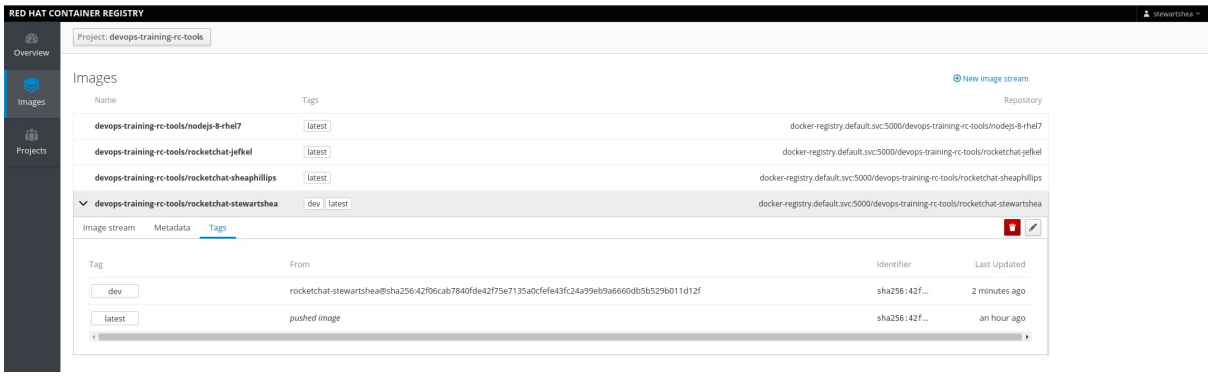
## Adding an Image Tag

In preparation for deployment to our dev environment, we will tag the latest version of our image with the tag `dev`.

- From the CLI

```
oc tag rocketchat-[username]:latest rocketchat-[username]:dev
```

- In the Registry Console, notice that the tag has now been added:

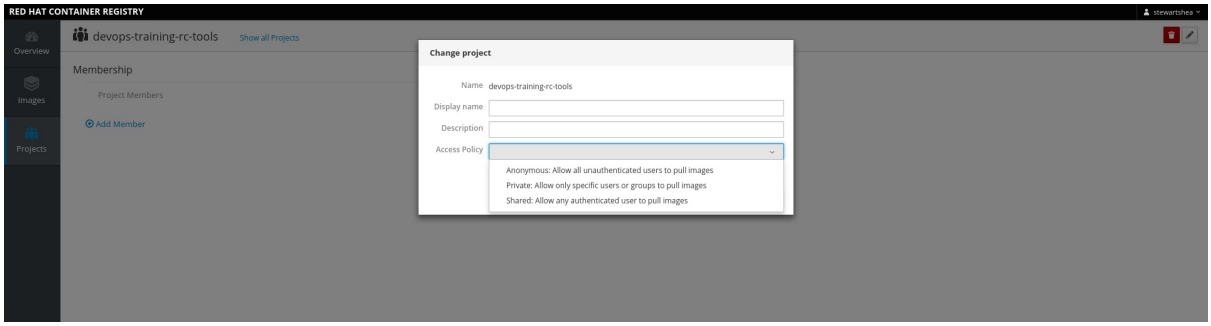



- From the CLI

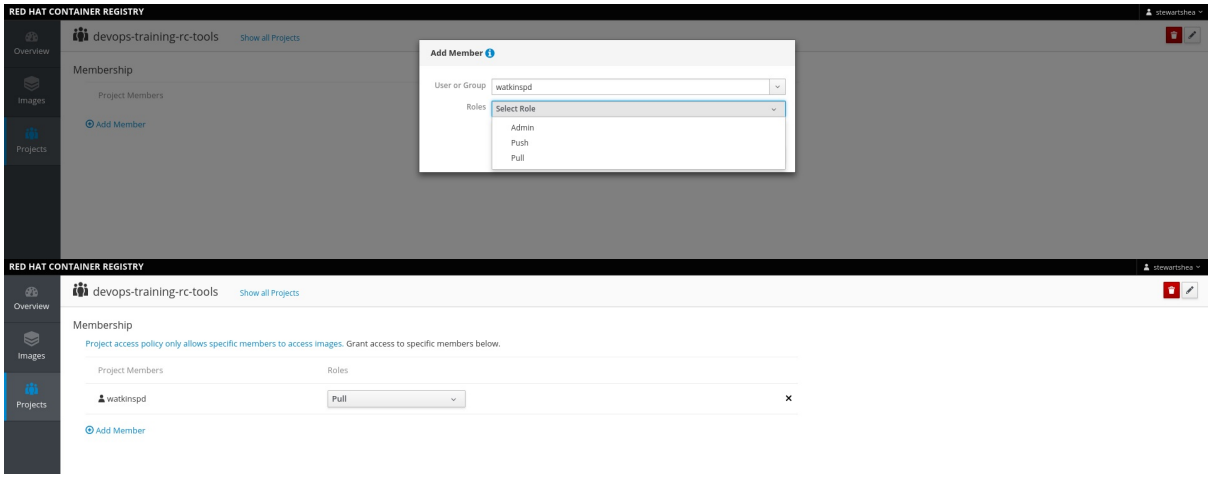
```
oc get imagestreams
```

## Changing Project Image Permissions

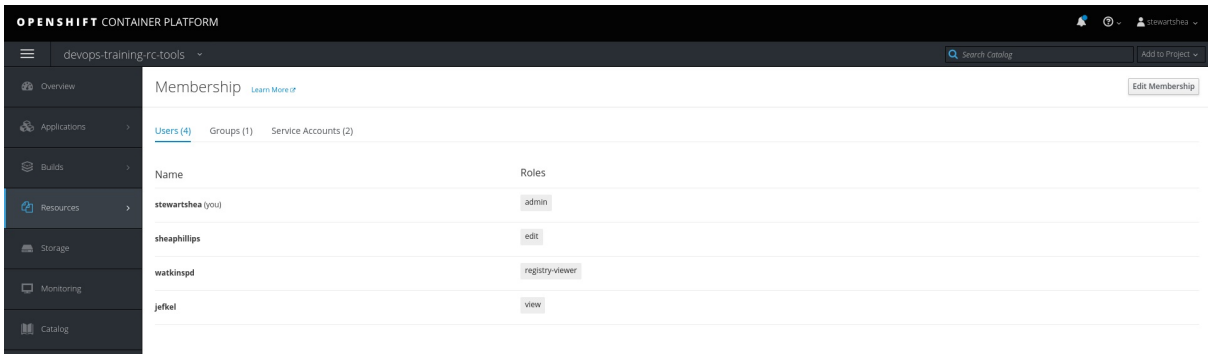
- Navigate to the Projects tab of the registry console and notice that the security for the project can be changed with the edit icon




- Or individual members / service account access can be added




- There is currently a bug if your username has upper case characters you wll see "The member name contains invalid characters. Only letters, numbers, spaces and the following symbols are allowed: , = @ . _" You can skip this step as it just for informational purposes. You can also do this from the command line `oc policy add-role-to-user system:image-puller UpperUsername`
- Notice that the change is reflected in the Web Console



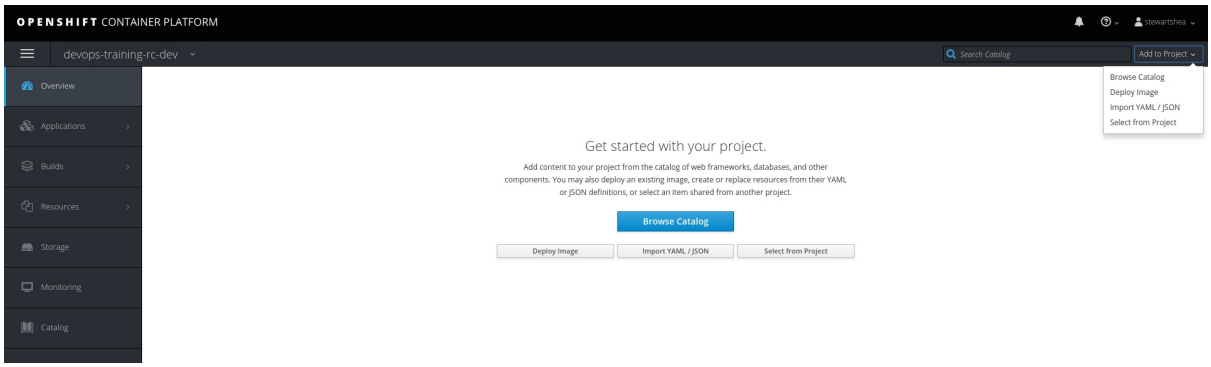
# Deployment

Since the build and deploy stages are separate, and we have a built image, we can deploy this image into our remining project spaces.

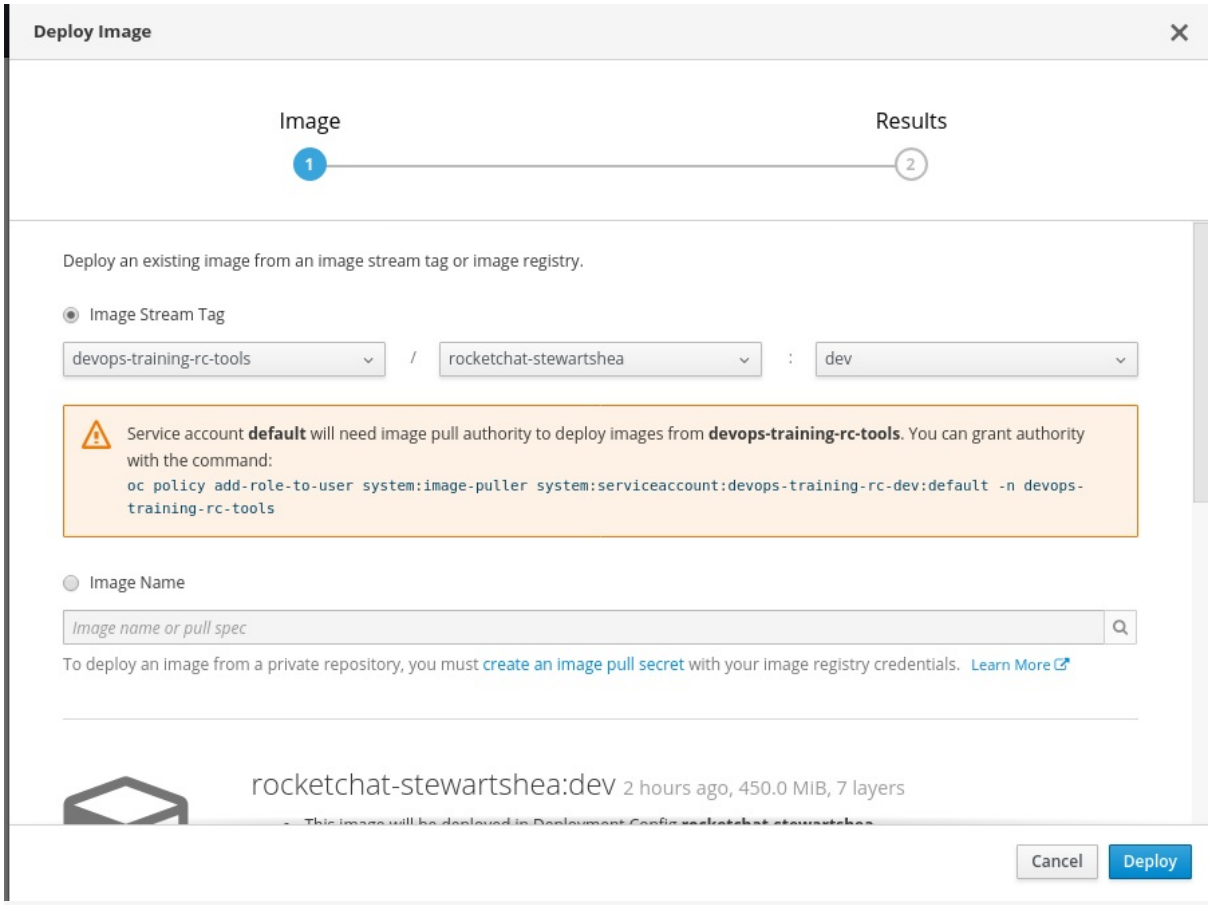
## Create an Image-Based Deployment

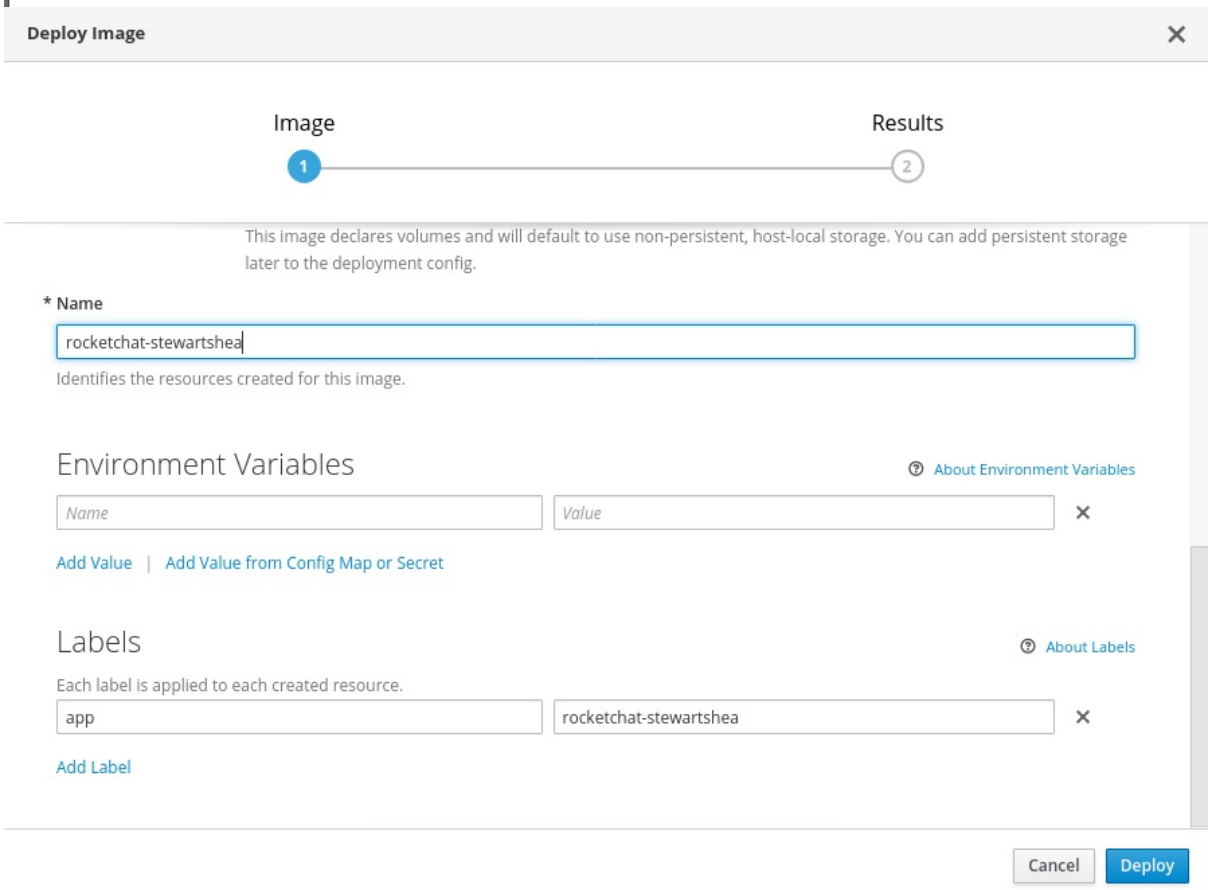
Navigate to the configured `dev` project and deploy an image.

- From the Overview tab, select `Deploy Image`, or from the top right corner from the drop down `Add to Project`




- Select the tools project, the your specific rocketchat image, and the appropriate tag




- Or do this from the CLI

```
oc new-app [devops-training-namespace]-tools/rocketchat-[username]:dev --name=rocketchat-[username]
```

- If performed with the CLI, the output should be as follows

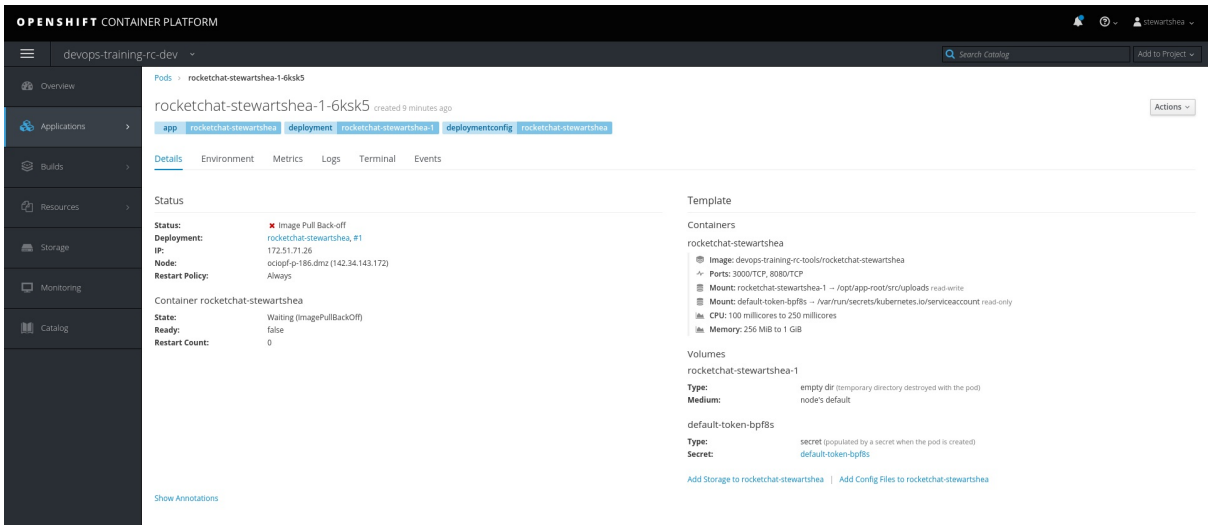
```
--> Found image b949f08 (2 hours old) in image stream "[devops-training-namespace]-tools/rocketchat-[username]"
 under tag "dev" for "[devops-training-namespace]-tools/rocketchat-[username]:dev"

    Node.js 8
    ---------
    Node.js 8 available as container is a base platform for building and running various Node.js 8 applications
 and frameworks. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable
network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and effici
ent, perfect for data-intensive real-time applications that run across distributed devices.

    Tags: builder, nodejs, nodejs8

    * This image will be deployed in deployment config "rocketchat-[username]"
    * Ports 3000/tcp, 8080/tcp will be load balanced by service "rocketchat-[username]"
      * Other containers can access this service through the hostname "rocketchat-[username]"
    * This image declares volumes and will default to use non-persistent, host-local storage.
      You can add persistent volumes later by running 'volume dc/rocketchat-[username] --add ...'

--> Creating resources ...
    imagestreamtag "rocketchat-[username]:dev" created
    deploymentconfig "rocketchat-[username]" created
    service "rocketchat-[username]" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing one or more of the co
mmands below:
     'oc expose svc/rocketchat-[username]'
    Run 'oc status' to view your app.
```

-

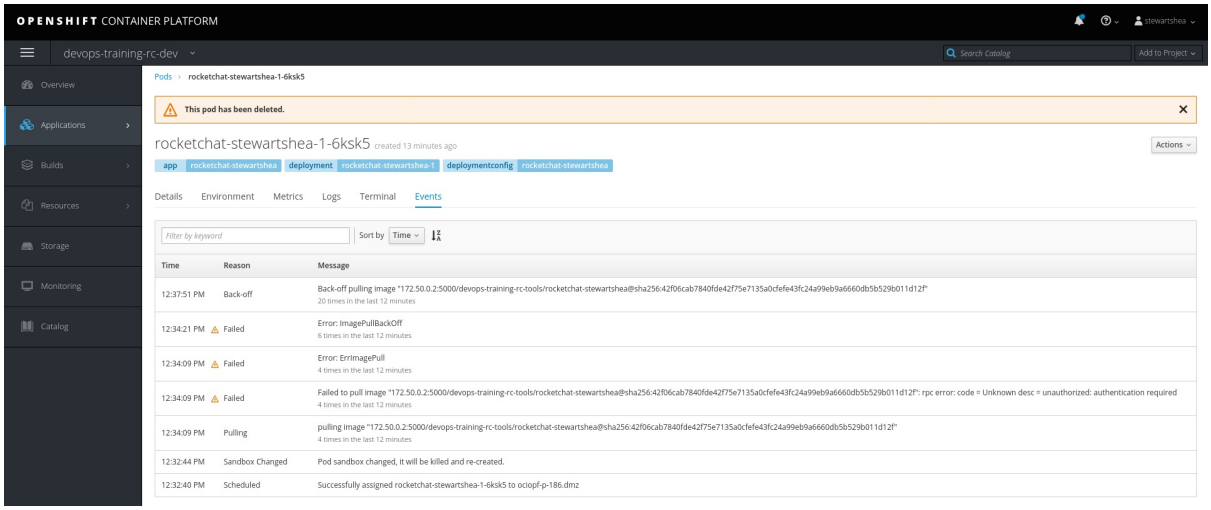
## Troubleshoot Image Pull Access

As the Web UI indicated, the `dev` project service accounts do not have the appropriate access to pull the image from the `tools` project.

- Navigate to `Applications -> Pods` and investigate further



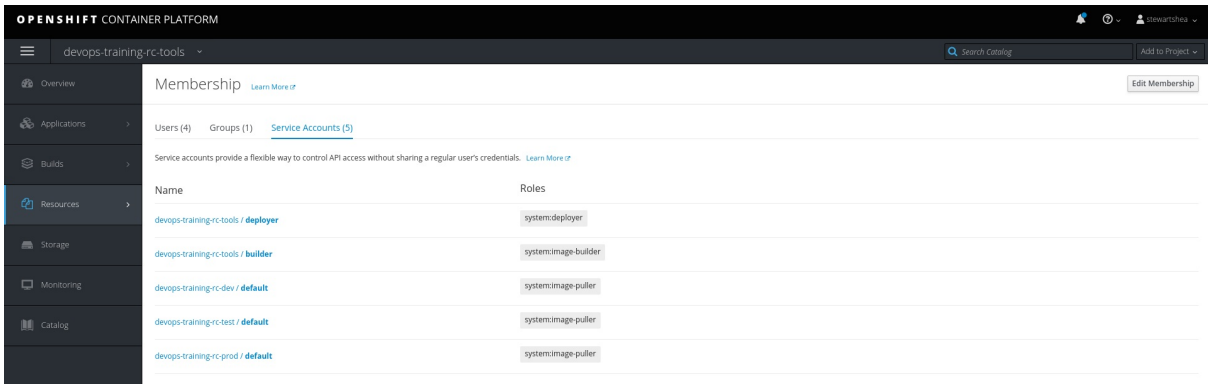

- Navigate to the pods `Events` tab for more detail




# Create Proper Access Rights Across Projects

Any team member with admin rights on the `tools` project can create the appropriate permissions for each other project.

- From the Web Console

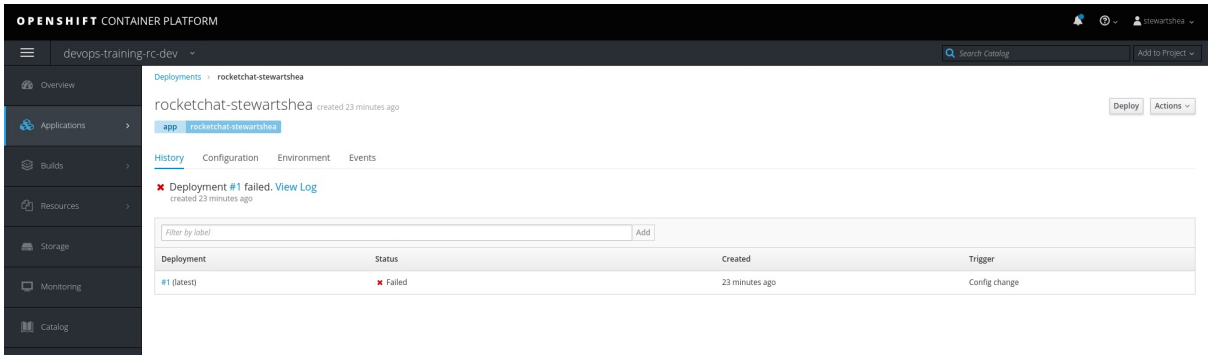



- From the CLI:

```
oc policy add-role-to-user system:image-puller system:serviceaccount:[devops-training-namespace]-dev:default -n
 [devops-training-namespace]-tools
oc policy add-role-to-user system:image-puller system:serviceaccount:[devops-training-namespace]-test:default -
n [devops-training-namespace]-tools
oc policy add-role-to-user system:image-puller system:serviceaccount:[devops-training-namespace]-prod:default -
n [devops-training-namespace]-tools
```

With the appropriate access in place, redeploy the application.

- From the Web Console



- OR from the CLI

```
oc rollout latest rocketchat-[username]
```

- Validate that the image is able to be pulled

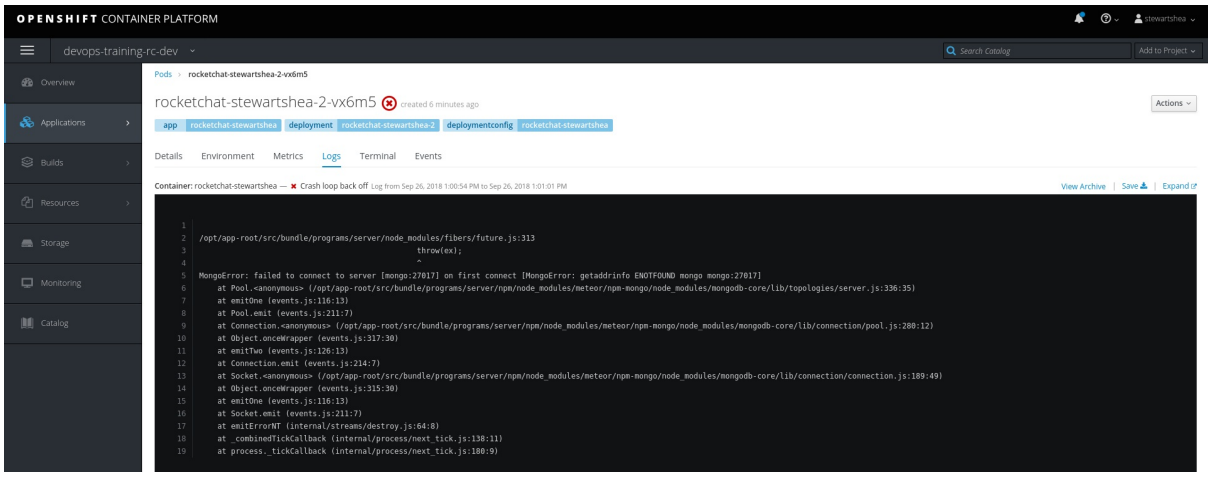
## Deploying the Database

Before going into further deployment configuration options, review the current status of the application container.

## Troubleshoot Deployment Issues

Navigate to the pod and review the logs to determine why we the container will not start.

- From the Web Console navigate to `Applications -> Pods -> rocketchat-[username]-[randomid]` and select Logs




- Or from the CLI

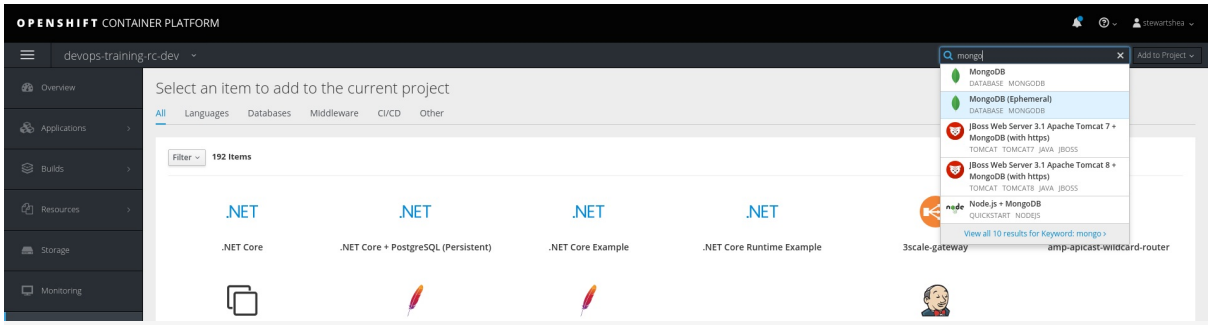
```
oc get pods  | grep rocketchat-[username]
oc logs rocketchat-[username]-[randomid]
```

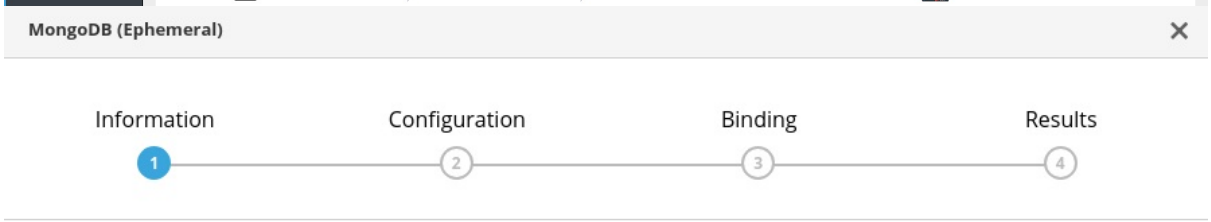
*note* you can follow the logs with `oc -f`

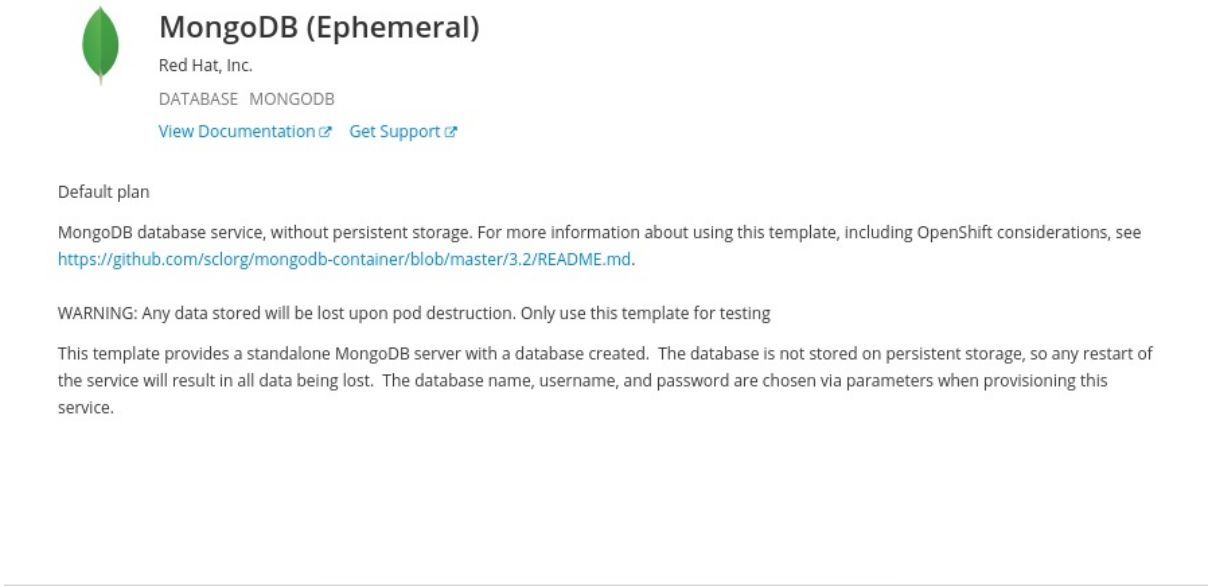
## Create Mongo Database with Ephemeral Storage

Having identified that the application is trying to connect to a mongo database, add a mongo database to the project for your application.

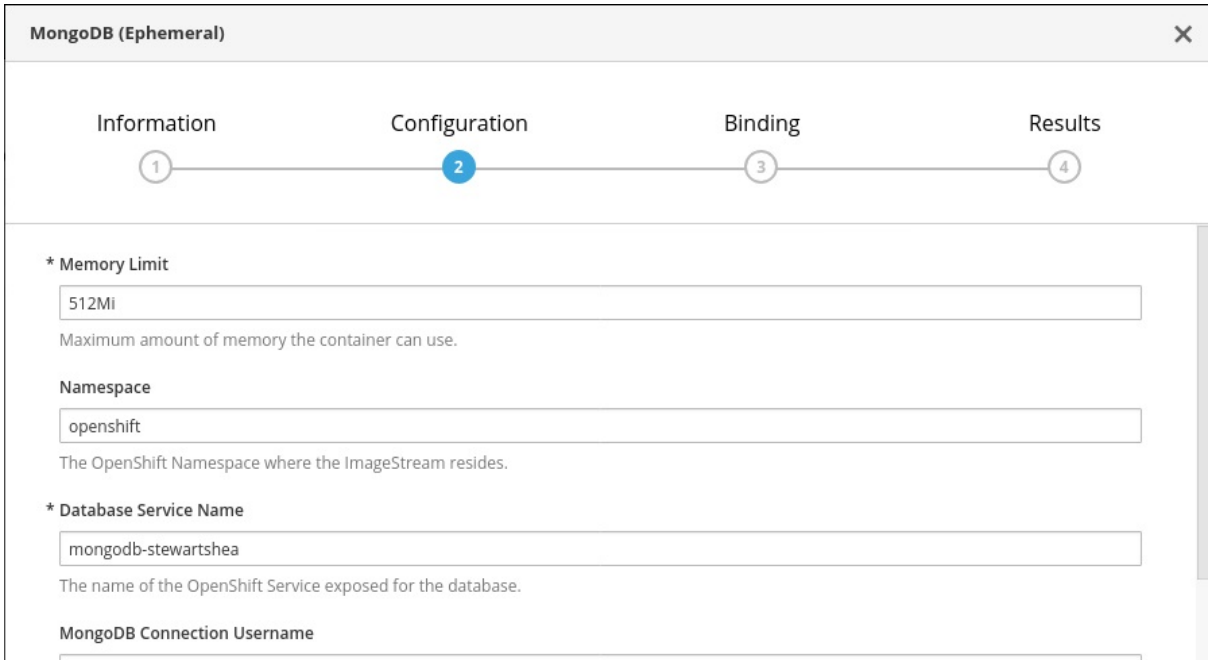
- From the Web Console
  - From the `Add to Project` dropdown, select `Browse Catalog`
  - In the search catalog area, type `mongo` and select `mongodb-ephemeral`
  - Ensure to customize the details with a service name such as `mongodb-[username]`, username/password and default database such as `rocketchat`

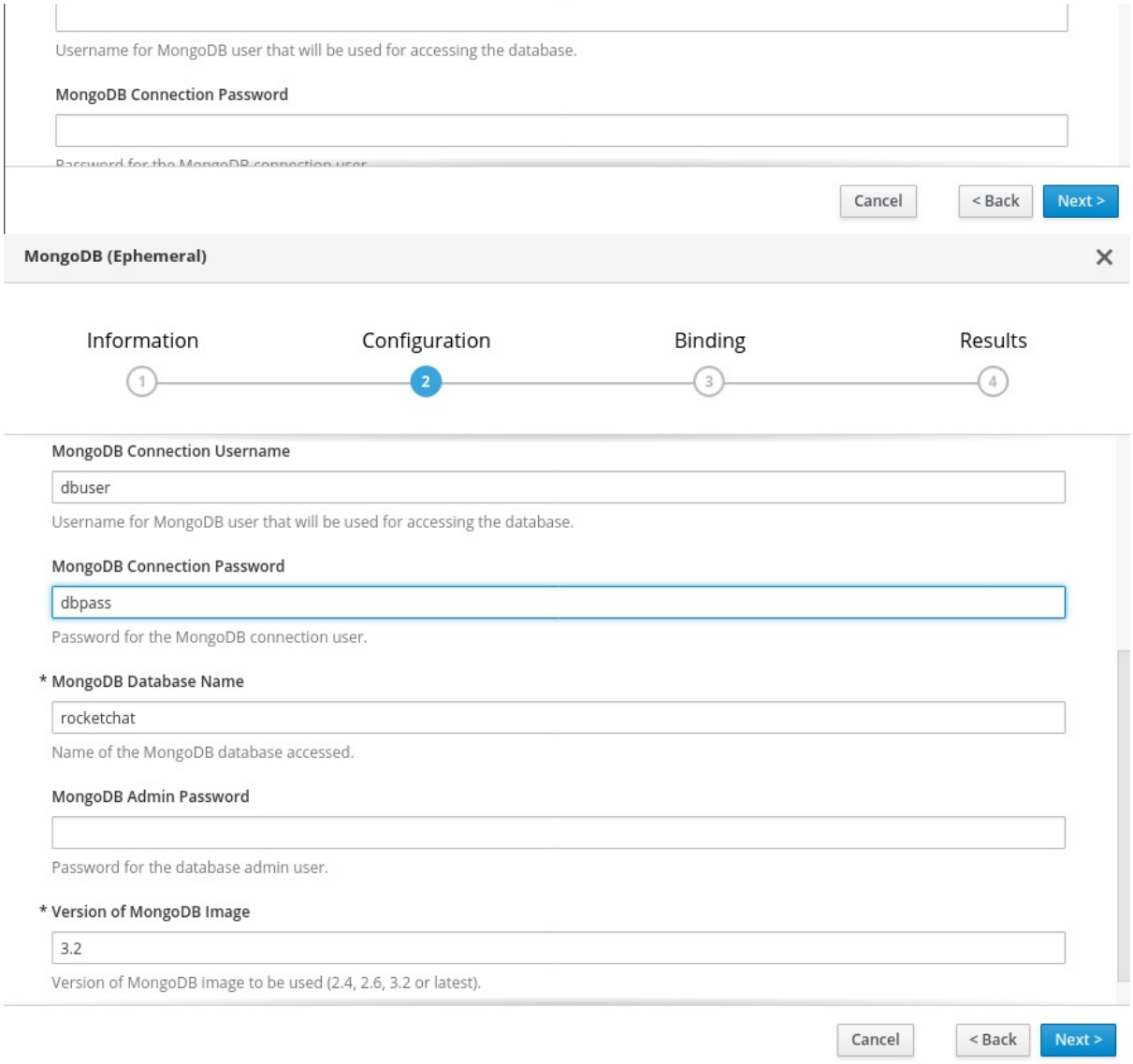

OPENSHIFT CONTAINER PLATFORM
devops-training-rc-dev
Overview
Applications
Builds
Resources
Storage
Monitoring
Select an item to add to the current project
All
Languages
Databases
Middleware
CI/CD
Other
Filter
192 Items
.NET
.NET Core
.NET Core + PostgreSQL (Persistent)
.NET Core Example
.NET Core Runtime Example
3scale-gateway
amp-apicast-wildcard-router
MongoDB
DATABASE MONGODB
MongoDB (Ephemeral)
DATABASE MONGODB
JBoss Web Server 3.1 Apache Tomcat 7 + MongoDB (with https)
JBoss Web Server 3.1 Apache Tomcat 8 + MongoDB (with https)
Node.js + MongoDB
QUICKSTART NODEJS
View all 10 results for Keyword: mongo

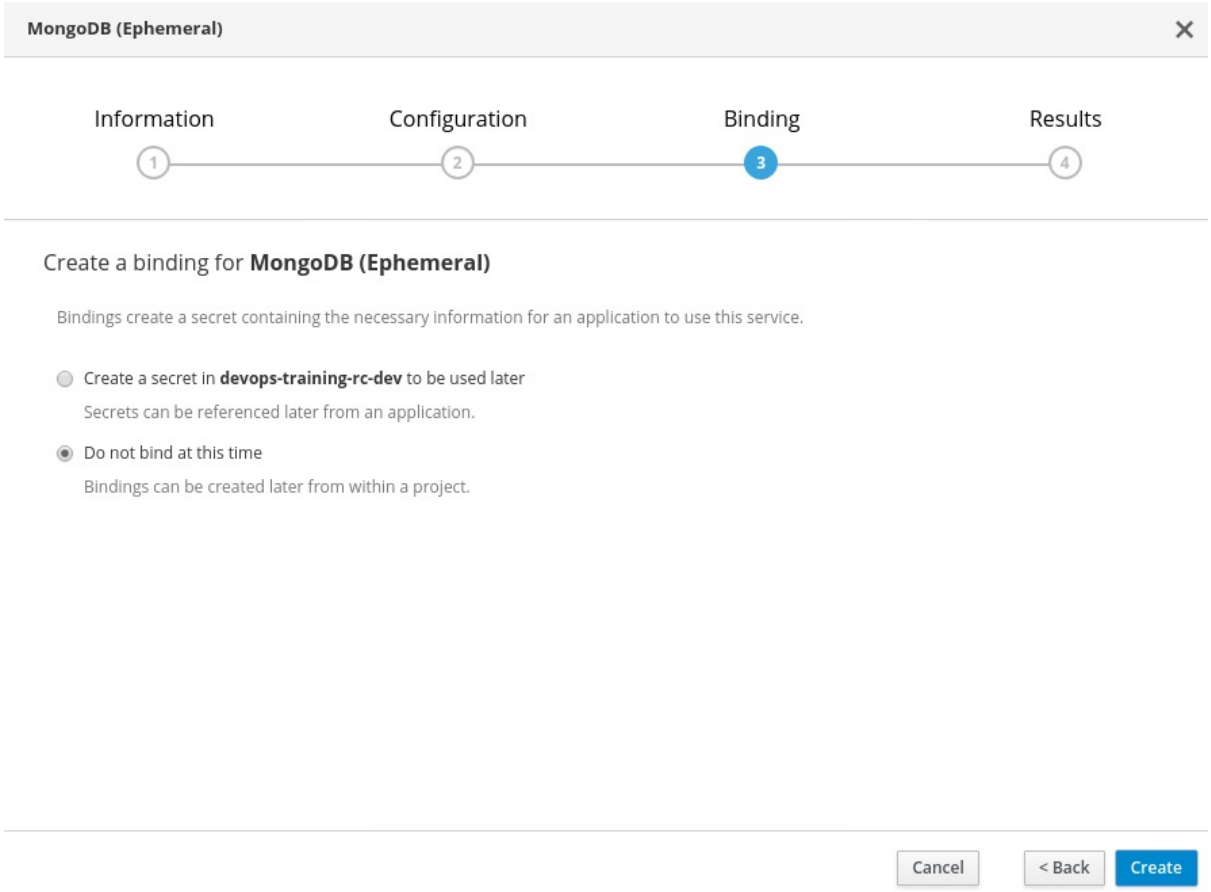

MongoDB (Ephemeral)
Information
Configuration
Binding
Results
1
2
3
4


MongoDB (Ephemeral)
Red Hat, Inc.
DATABASE MONGODB
View Documentation
Get Support
Default plan
MongoDB database service, without persistent storage. For more information about using this template, including OpenShift considerations, see https://github.com/sclorg/mongodb-container/blob/master/3.2/README.md.
WARNING: Any data stored will be lost upon pod destruction. Only use this template for testing
This template provides a standalone MongoDB server with a database created. The database is not stored on persistent storage, so any restart of the service will result in all data being lost. The database name, username, and password are chosen via parameters when provisioning this service.


Cancel
< Back
Next >


MongoDB (Ephemeral)
Information
Configuration
Binding
Results
1
2
3
4
* Memory Limit
512Mi
Maximum amount of memory the container can use.
Namespace
openshift
The OpenShift Namespace where the ImageStream resides.
* Database Service Name
mongodb-stewartshea
The name of the OpenShift Service exposed for the database.
MongoDB Connection Username


Username for MongoDB user that will be used for accessing the database.
MongoDB Connection Password
Cancel
< Back
Next >
MongoDB (Ephemeral)
Information
Configuration
Binding
Results
1
2
3
4
MongoDB Connection Username
dbuser
Username for MongoDB user that will be used for accessing the database.
MongoDB Connection Password
dbpass
Password for the MongoDB connection user.
* MongoDB Database Name
rocketchat
Name of the MongoDB database accessed.
MongoDB Admin Password
Password for the database admin user.
* Version of MongoDB Image
3.2
Version of MongoDB image to be used (2.4, 2.6, 3.2 or latest).
Cancel
< Back
Next >




- Bonus: determine how to create this from the CLI

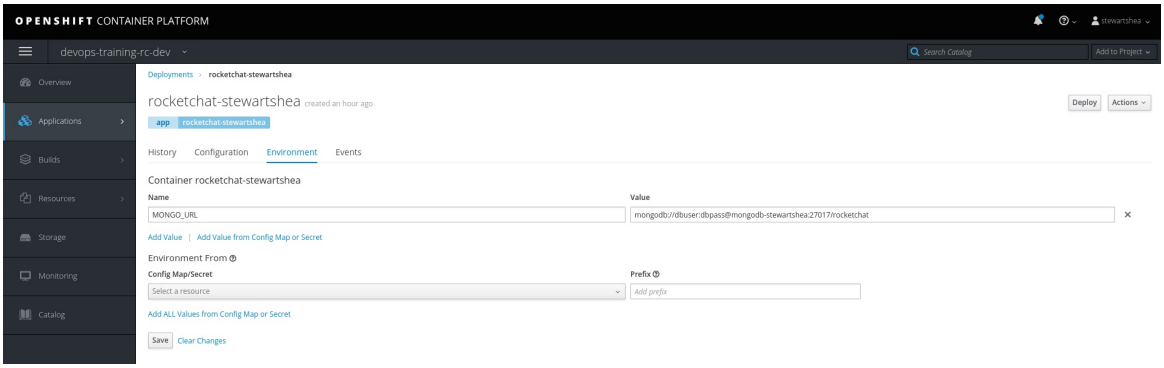
## Deployment Configuration Options

As a result of using a generic `new-app` style deployment, as opposed to openshift specific templates, a lot of defaults are leveraged.

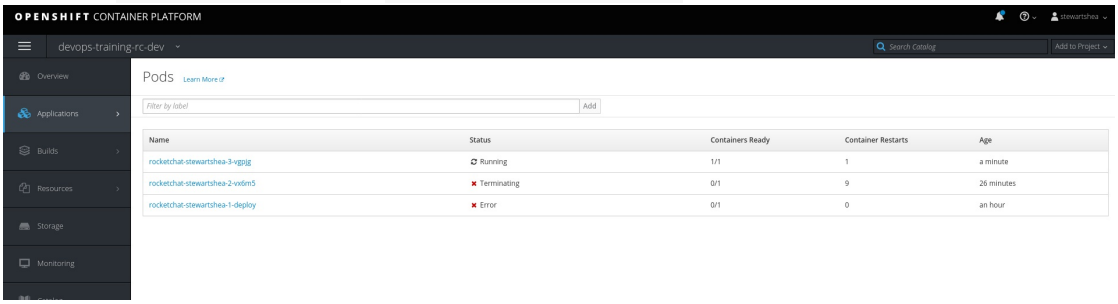
## Environment Variables

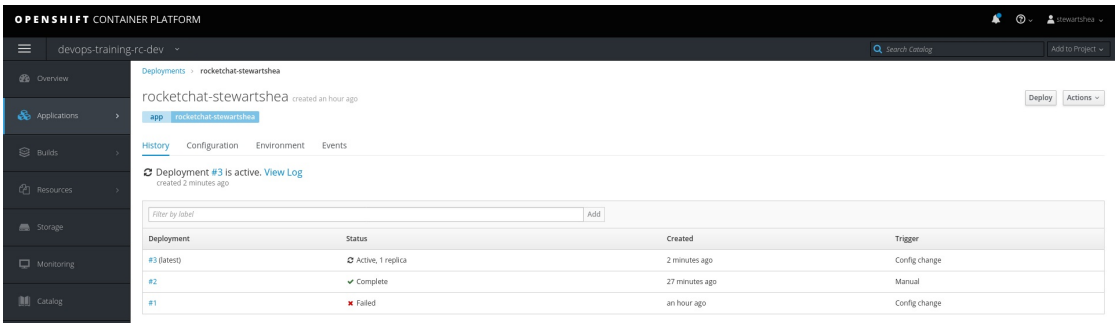
By default we currently have no environment variables attached to our deployment configuration. So, while the app trying to start, and a database has been deployed, the app does not know how or where to connect to. Add an environment variable to the deployment configuration.

- In the Web Console, navigate to `Applications -> Deployments`, and select your deployment
- Select the `Environment Tab`
- Add the following environment variable with the connection string details configured for mongodb



- Click save and take note of what happens next
  - Navigate to `Applications -> Pods` and `Applications -> Deployments` to notice the changes



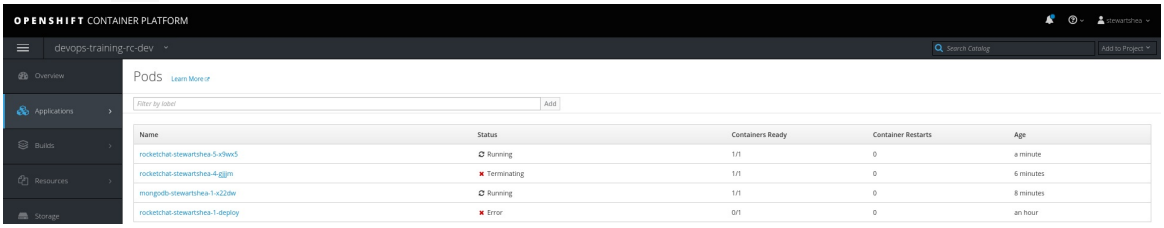

While you are waiting for the application to redeploy, expose the route to the public internet.

- From the Web Console, navigate to `Applications -> Routes`
- Select `Create Route`
  - Customize the name of the route, such as `rocketchat-[username]`
  - Ensure the service it points to is your particular service

## Exploring Health Checks

With the new deployment running, monitor the readiness of the pod.

- Navigate to `Applications -> Pods`
- Notice that `1/1` containers are ready

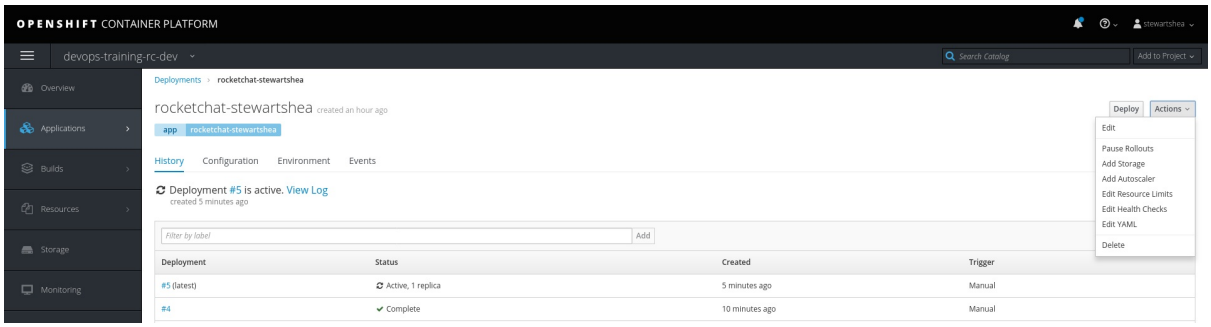



- Visit the application route, however, and notice that the application is not ready

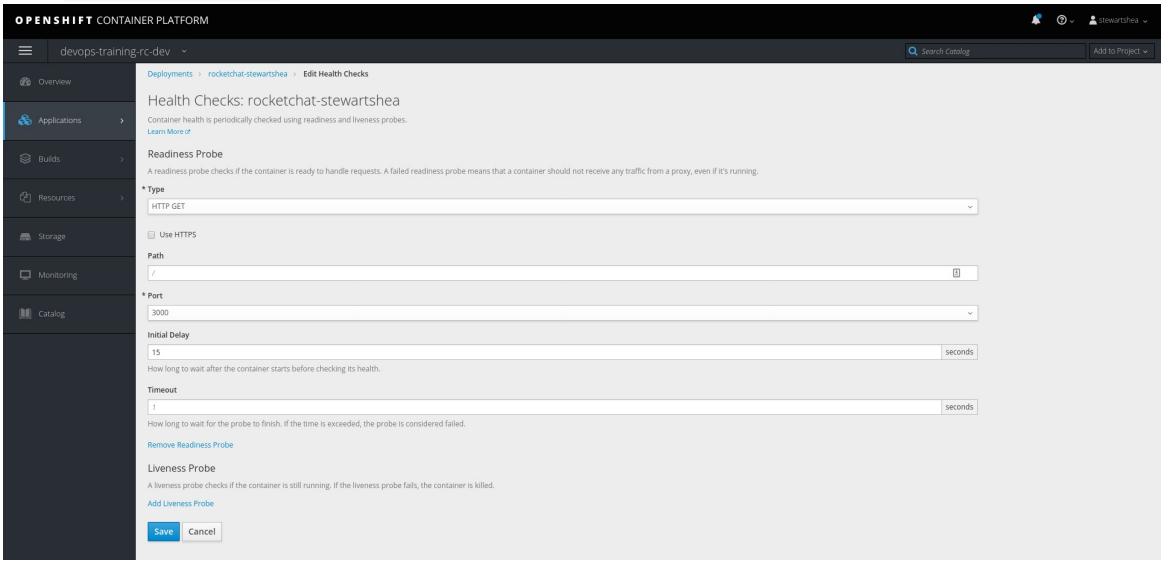

## Adding a Healthcheck

A container that is marked `ready` when it is not is an indication of a lack of (or misconfigured) healthcheck. Let's add a healthcheck.

- Navigate to `Applications -> Deployments`
- Select the appropriate deployment
- Select `Actions` and then `Edit Health Checks`




- Select `Add Readiness Probe` and leverage the HTTP GET defaults for this application, setting an initial delay of 15 seconds




- While the new deployment rolls out, continue to refresh the public route and validate that it stays up
- At the same time, monitor the pod `Containers Ready` column from `Applications -> Pods` and notice what happens when it becomes ready

## Exploring Deployment Configuration Options

Additional actions are avalable to edit your deployment configuration. Review and explore;

- Resource Limits
- Healthcheck liveliness probes
- YAML

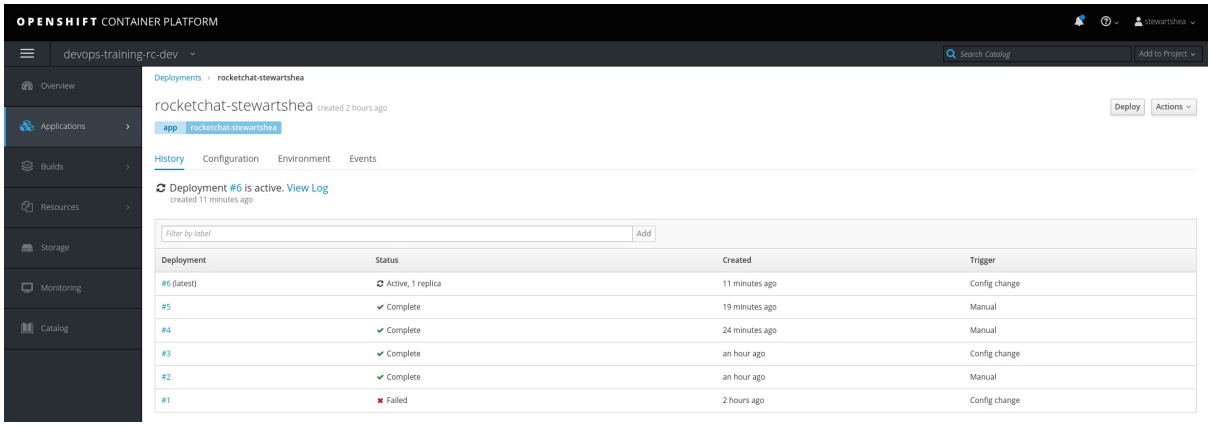
## Versioning a Deployment Configuration

At this point in time, your deployment configuration has undergone many changes, such as adding environment variables and adding health checks. Review the deployment configuration `History` tab:

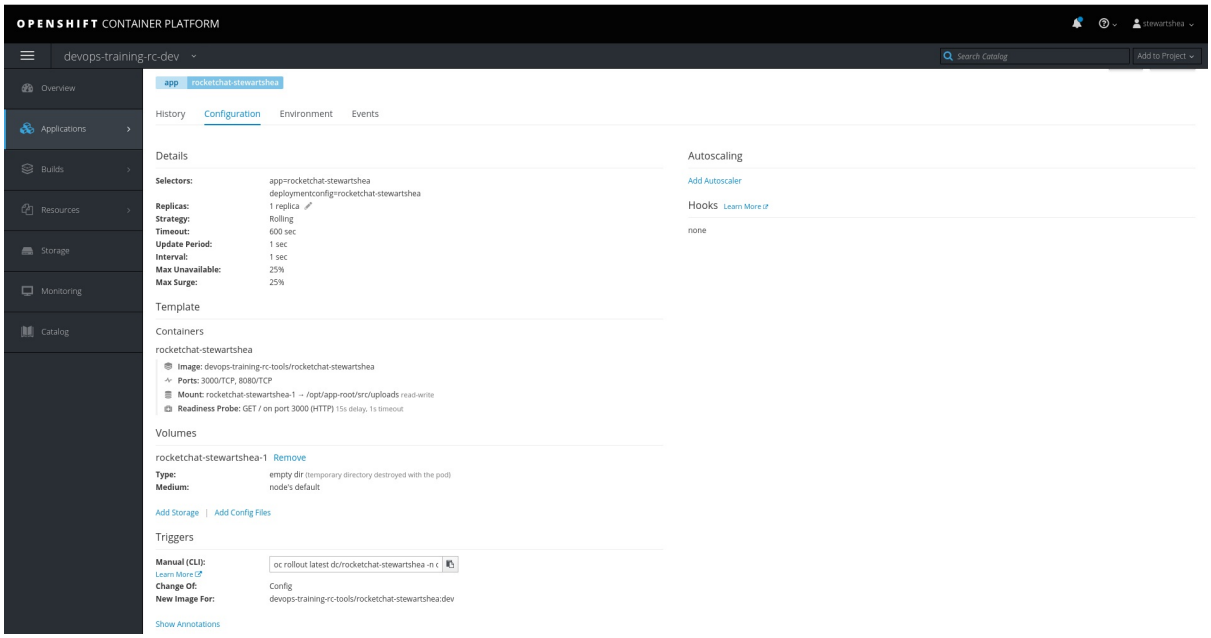
- Select Deployment #1, right-click, and open in a new tab
- Select your latest deployment version, right-click, and open in a new tab
- Compare the differences - this can be done through the UI or by comparing the YAML

## Changing Deployment Configuration Triggers

While reviewing the different deployment versions, take note of the `Trigger` column.

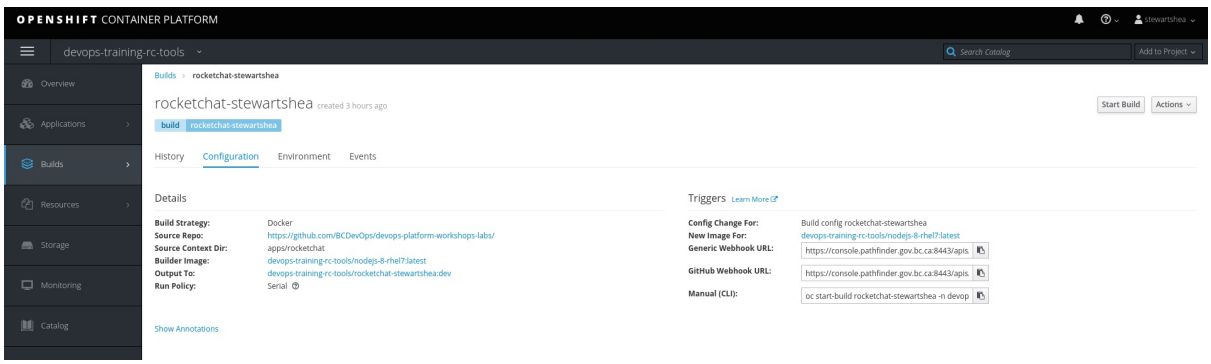



- Navigate to the `Configuration` tab of the deployment and review the currently configured Triggers



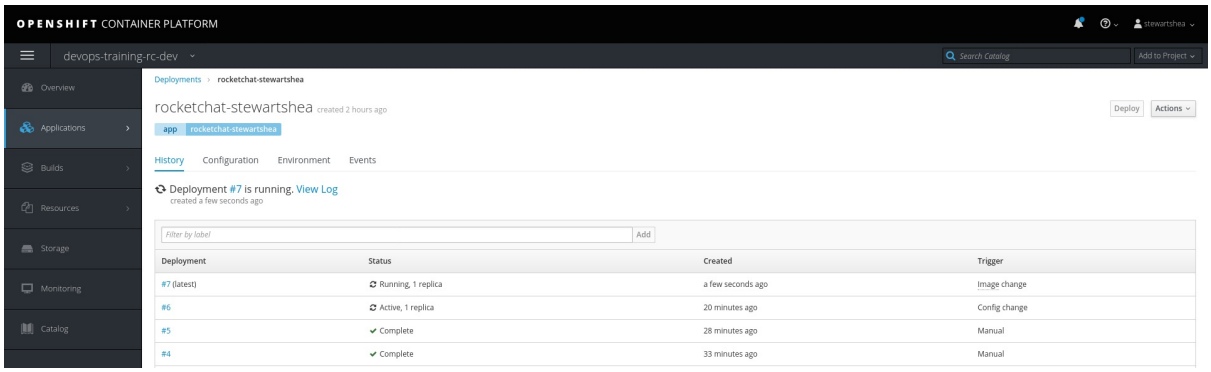

Explore how an Image can also trigger a deployment

- Navigate to your original build and investigate the available triggers



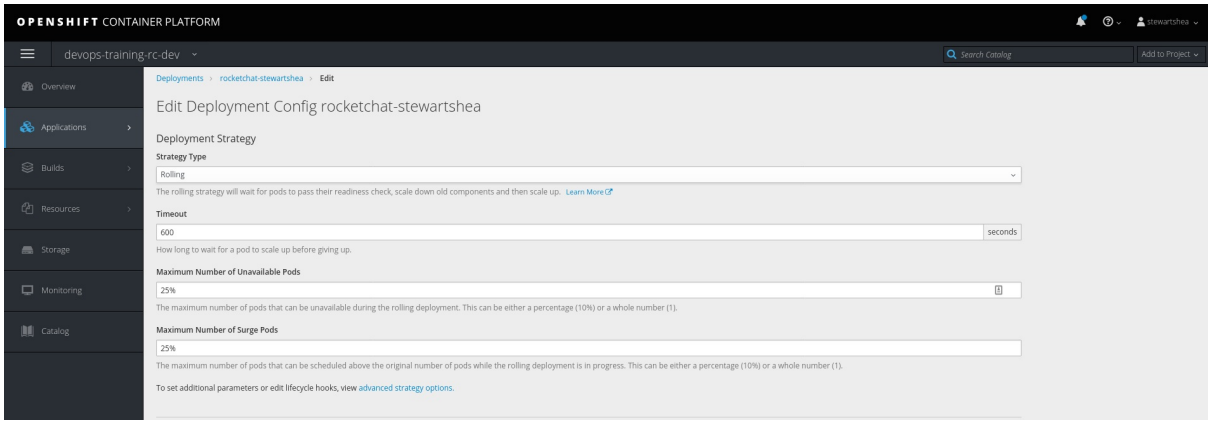

- Edit the buildconfig to change the output image to the `dev` tag

- Start a new build and monitor your the `dev` deployment when the build completes




## Changing the Deployment Strategy Option

The default deployment configuration provides a `Rolling Update` style deployment, which waits for the container to be ready prior to cutting over traffic and terminating the previous container.




- Change the strategy to a `Recreate` and redeploy a couple of times
- Refresh the browser URL right after a new deployment and observe the behavior
- Change the strategy back to `Rolling`

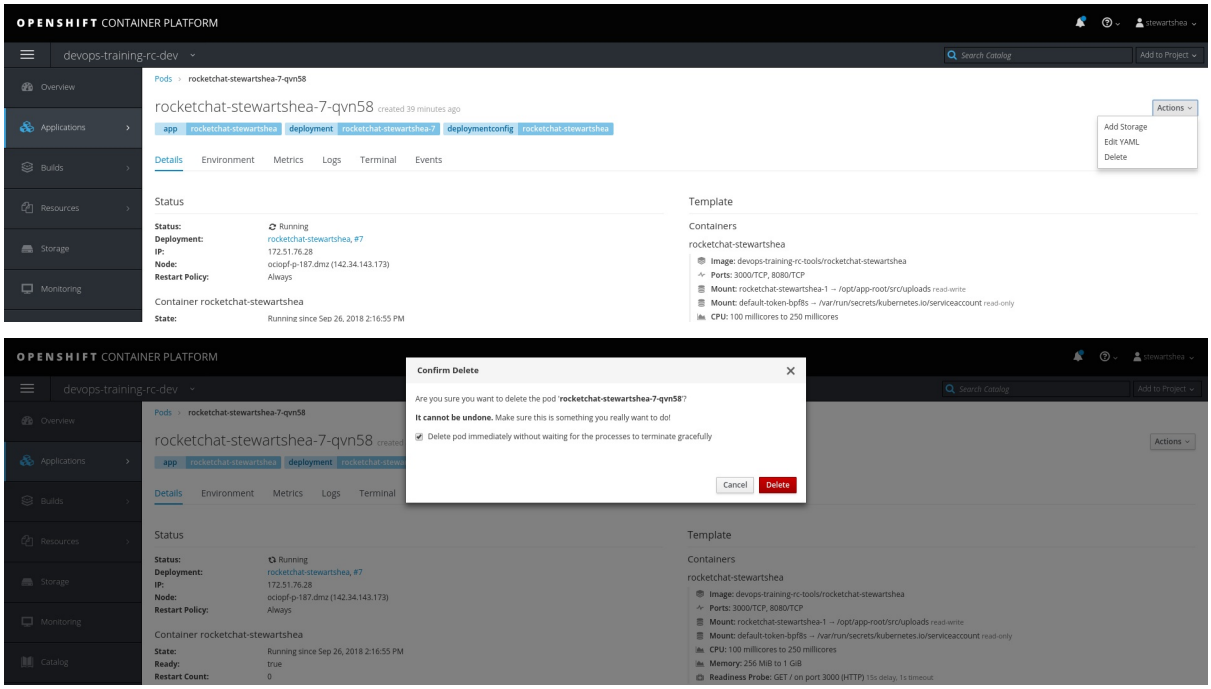
# Application Availability

Prior to beginning this lab, nagivate to the public route you created and finish the initial setup of your application. Ensure to select `Keep standlone` on the last page of the initial Rocket.Chat application setup.

## Single Pod Applications

Single pod applications are not highly-available and can be abruptly terminated if the pod crashes or due to regularly scheduled platform maintenance. In order to simulate the effect on an application:

- Navigate to the pod, select `Actions` and `Delete`
  - Select `Delete pod immediately`







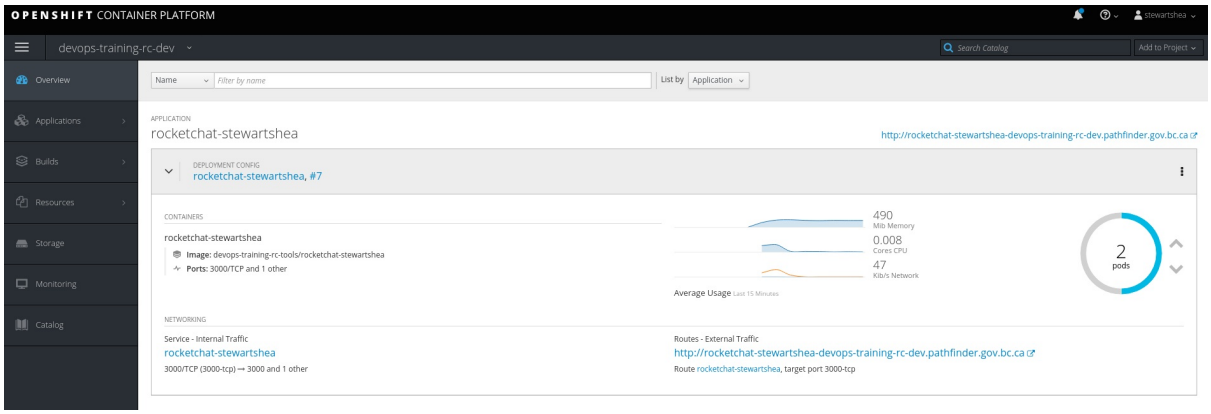
- Refresh URL of application

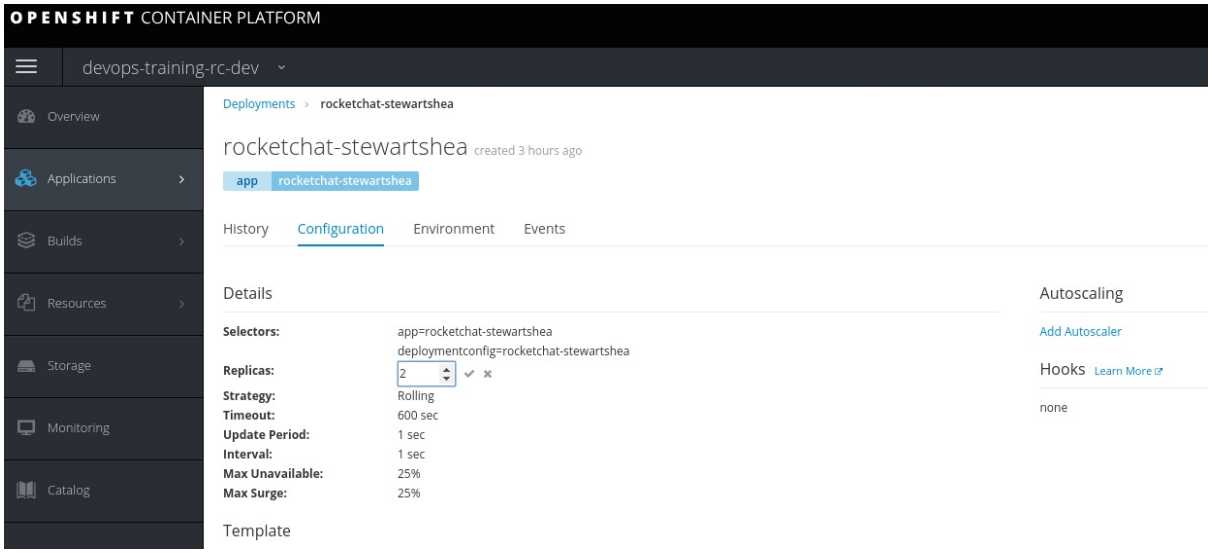

## Scaling Pods

To increase the availability of an application and defend against unplanned outages or planned maintenance tasks, an application must have multiple pods/instance running. For this reason, stateless applications are desirable as they do not require custom clustering configurations.

**Note** Stateful applications do not support "scaling pods" as a form of high availability. Such a stateful example would be the mongodb database. For this reason, this lab focuses on the rocketchat application which will function with multiple pods. Please refer to specific application documentaion for details on scalability support.

- Navigate to `Overview` or `Deployments` in the Web Console and increase the pod count to 2

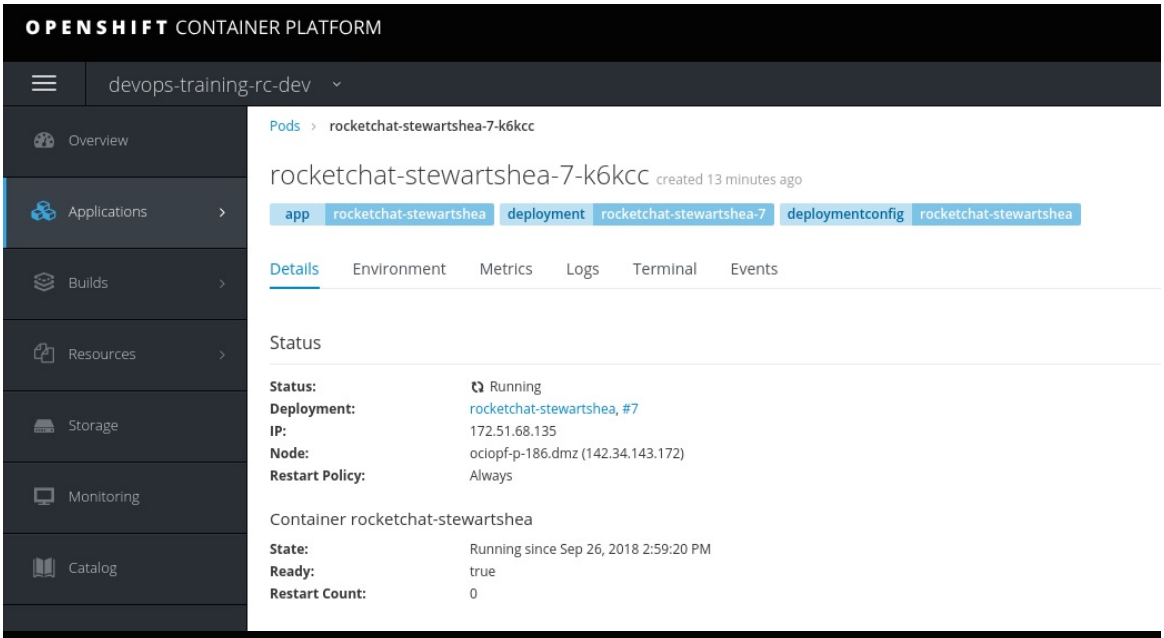


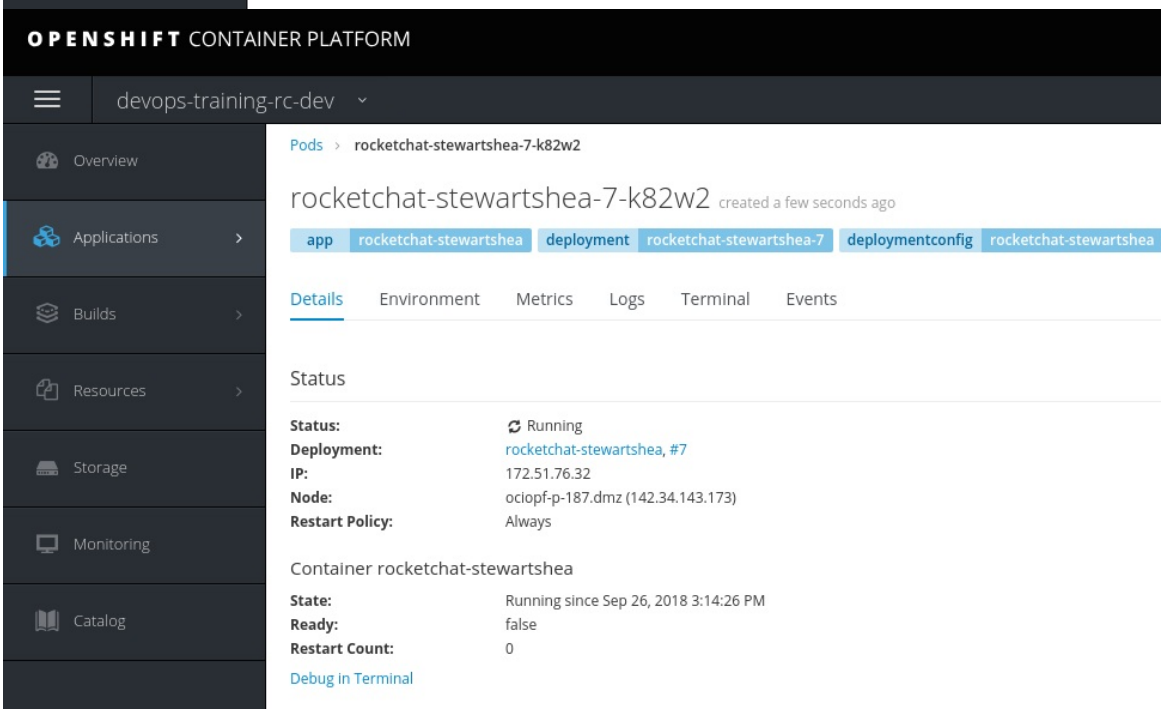




- Or from the CLI

```
oc scale dc/rocketchat-[username] --replicas=2
```

- Notice the balancing across nodes by exploring the details of each pod




- Or from the CLI notice the hosts the pod runs on (in the last field)

```
oc get pods -o wide  | grep rocketchat-[username]
```

```
- The output should look similar to this:
```
$ oc get pods -o wide  | grep rocketchat-stewartshea
rocketchat-stewartshea-7-k6kcc    1/1       Running   0          16m       172.51.68.135   ociopf-p-186.dmz
rocketchat-stewartshea-7-k82w2    0/1       Running   0          1m        172.51.76.32    ociopf-p-187.dmz
```
```

- Delete single pod, refresh the URL of application and notice that the application is served by the surviving pods
- Perform deployment, refresh the URL of application and notice that the application is served by the surviving pods

# Autoscaling

Autoscaling can be configured on pods to enable OpenShift to add or remove pods as load varies. In general, the Horizonal Pod Autoscaler sets:

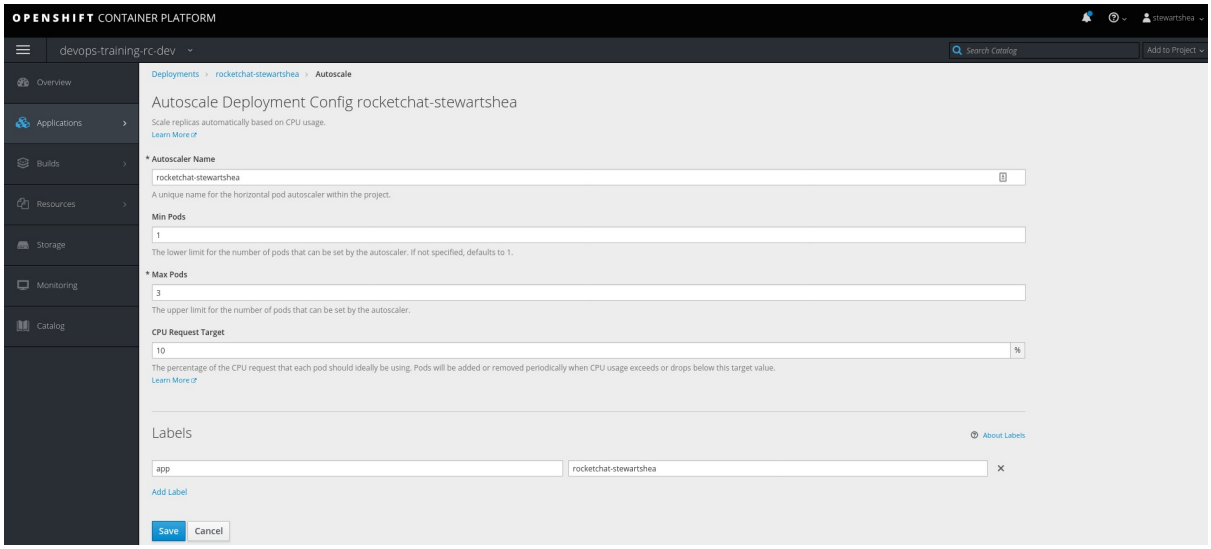
```
- Upper limit of pods
- Lower limit of pods
- Metric threshold to use for scaling tasks
```

Currently CPU and memory metrics are supported, with additional arbitrary metrics support intended for the future.

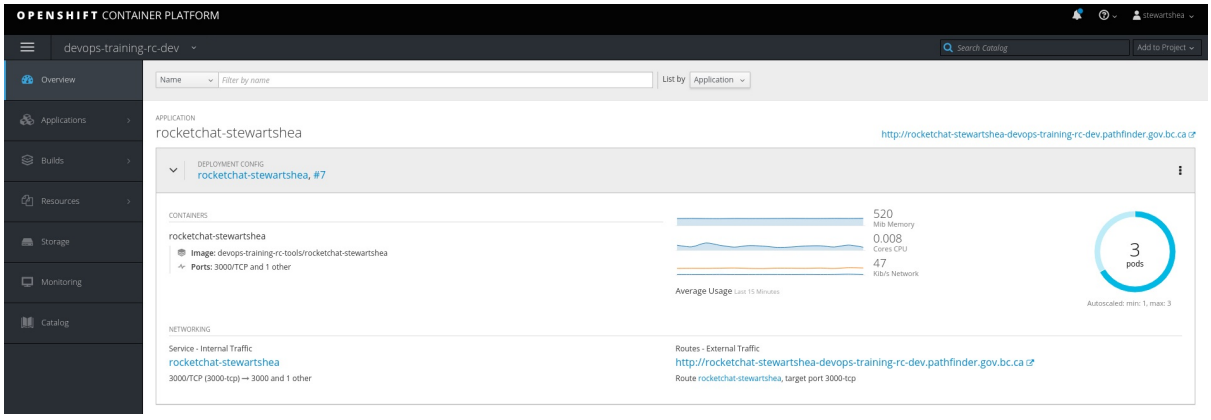
## Autoscaling Pods

This lab will provide a simple demonstration of autoscaling based on CPU, as this is configurable in the Web Console.

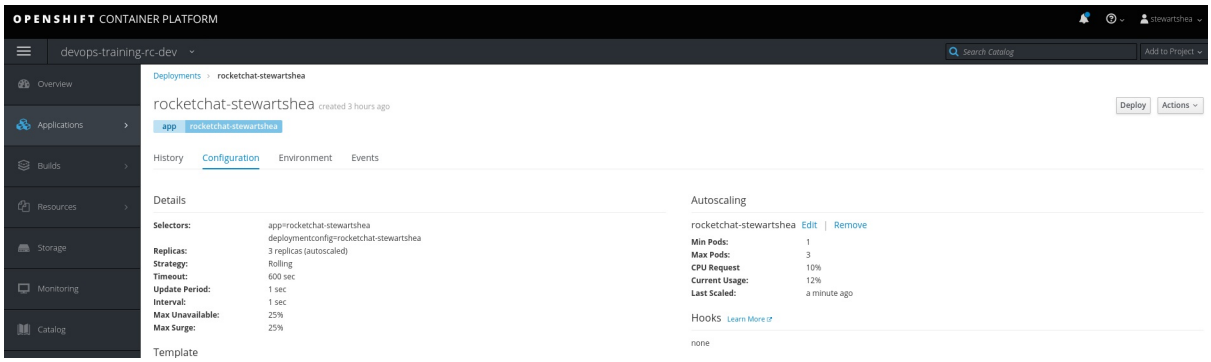
- Navigate to your rocketchat deployment and select the `Actions` dropdown
- Select `Add Autoscaler`
- Configure an upper and lower limit of pods
- Configure a very low CPU Request Target (such as 1 or 2%) so that you can test it easily by browsing the web application
    - In a production environment you would taget something like 70-80%




- Browse to the application to generate some load and monitor the behavior of the pods
    - Generate some activity such as creating messages, channels, etc.



- Review the deployment configuraiton and try to add or remove repicas

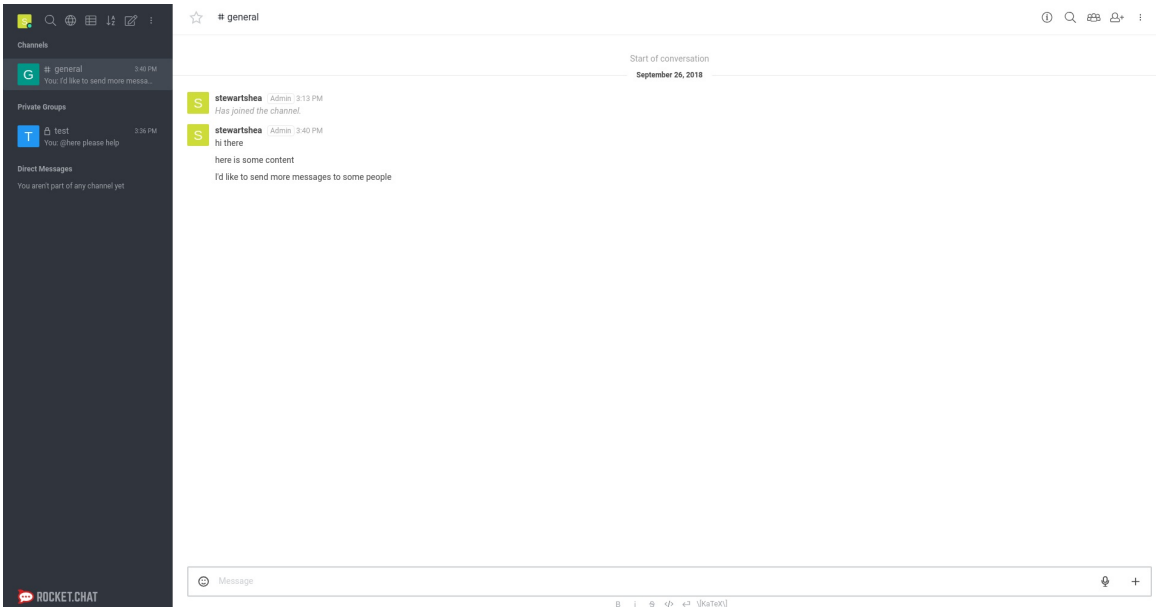



- Remove the autoscaler

# Persistent Storage

Up to this point you have leveraged a single mongodb pod with ephemeral storage. In order to maintain the appliation data, persistent storage is required.

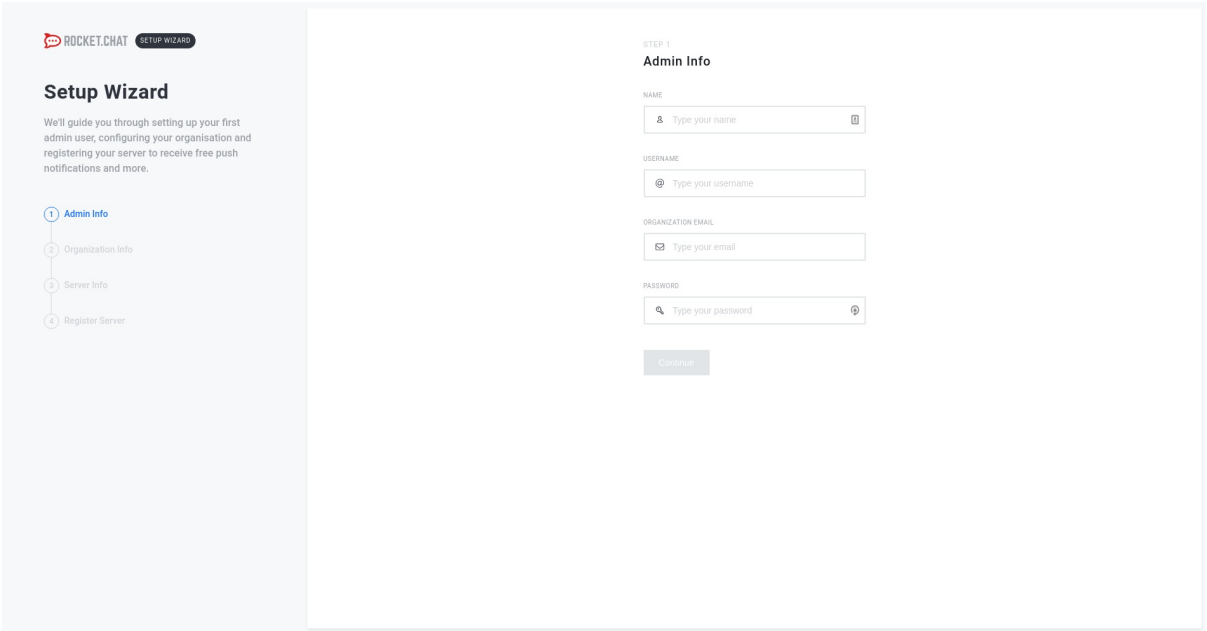
- Let's first take a look at our application prior to this lab




## Deleting Pods with Ephemeral Storage

To understand what will happen when a pod with ehemeral storage is removed,

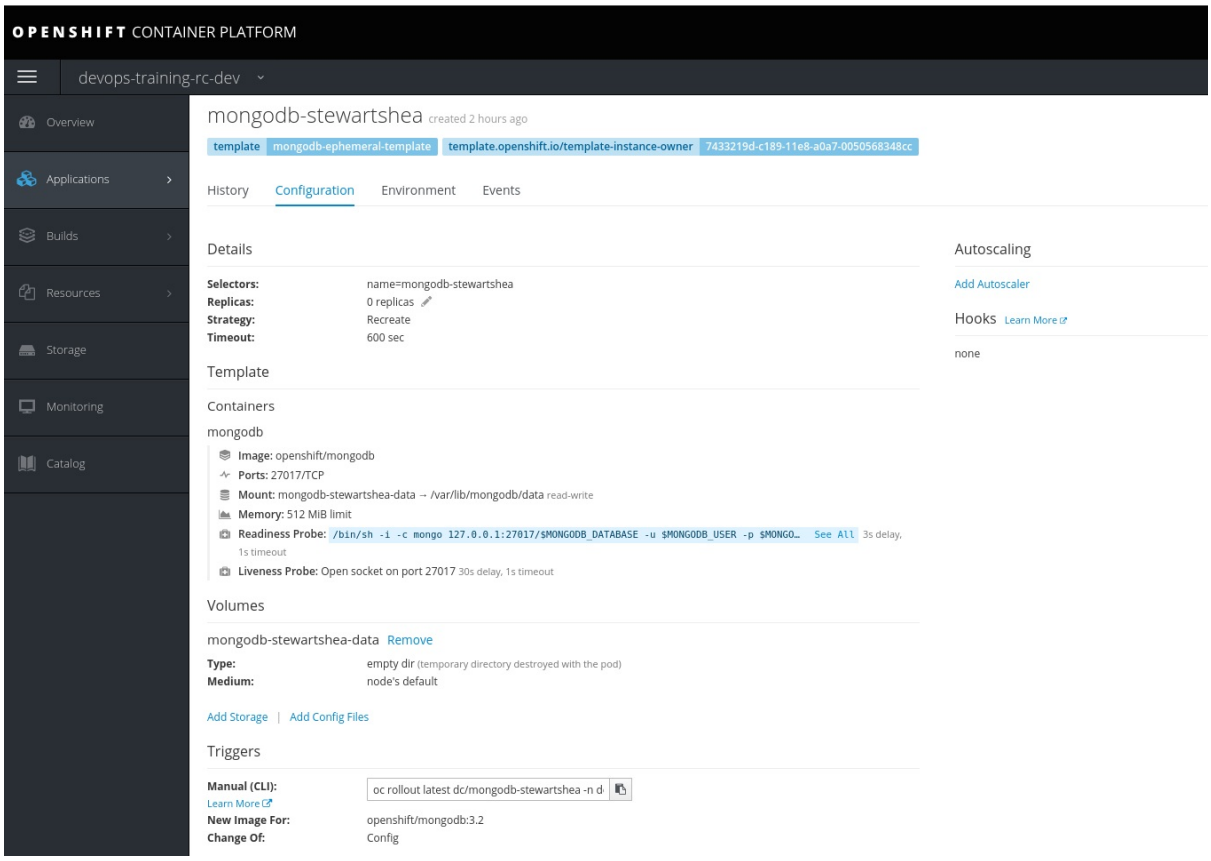
- Scale down both the rocketchat and mongo applications to 0 pods
- Scale back up each application pod to 1 replica

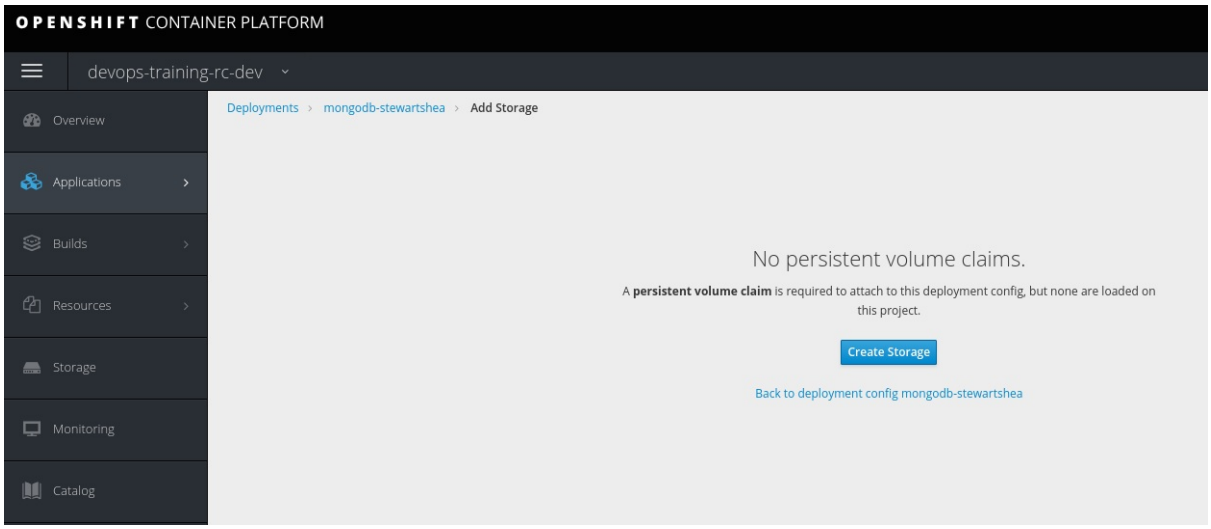



## Adding Storage to Existing Deployment Configurations

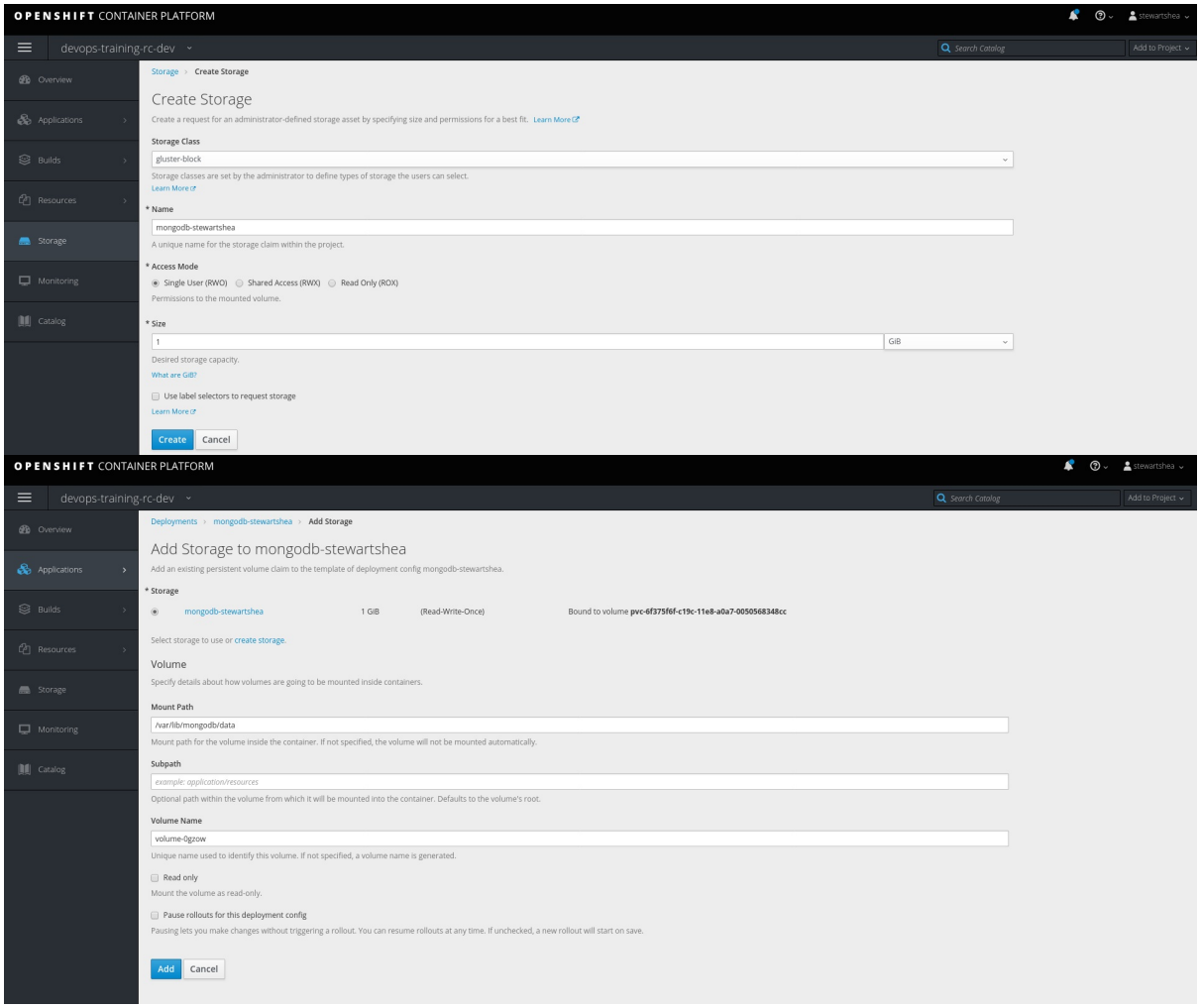
Now that we notice all messages and configuaration is gone, let's add persistent storage to the mongodb pod.

- Scale down both the rocketchat and mongo applications to 0 pods
- Edit the `mongodb-[username]` configuration
  - Remove the `emptyDir` volume
  - Add a new volume by selecting `Add Storage`




- Select the `gluster-block` storage class, set the type to RWO (which is block storage), and the size to 1GB, with a name of `mongodb-[username]` **note** Each application will have a preferred storage type. This is NOT the recommended storageclass for mongo, but is useful in an upcoming lab.

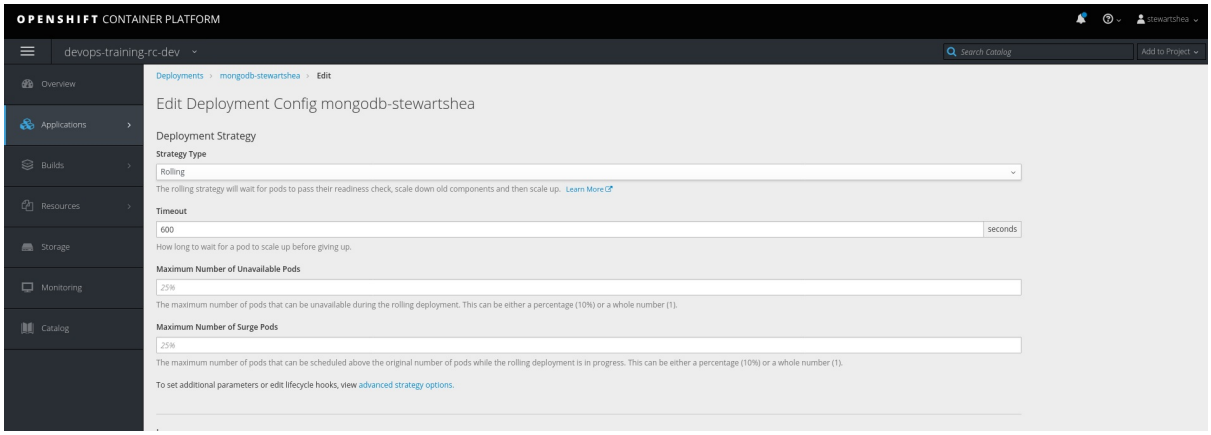



- Scale up `mongodb-[username]` instance to 1 pod
- When mongo is running, scale `rocketchat-[username]` to 1 pod
- Configure RocketChat again
- Scale down and scale back up both the database and the rocketchat app
- Verify that data was persisted

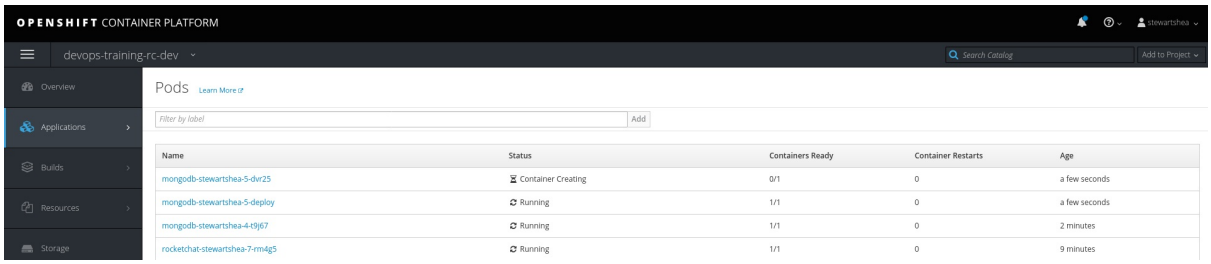
## RWO Storage

RWO storage (which was selected above) can only be attached to a single pod at a time, which causes issues in certain deployment stategies.

- Ensure your `mongodb-[username]` deployment is set to rolling




- Redeploy with Rolling Deployment
- Notice and investigate the issue

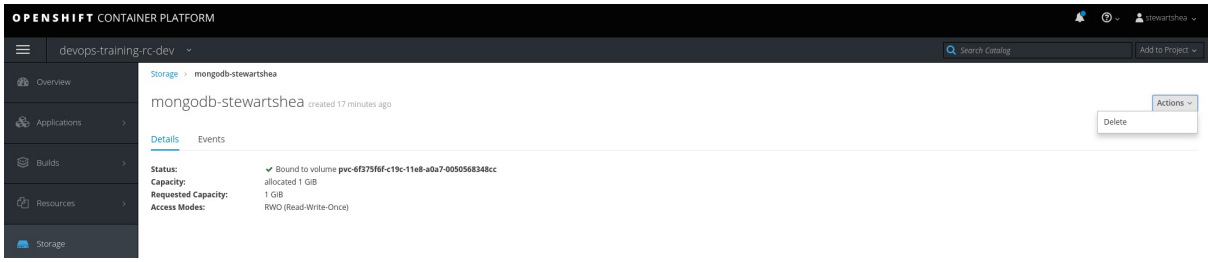



- Switch to recreate

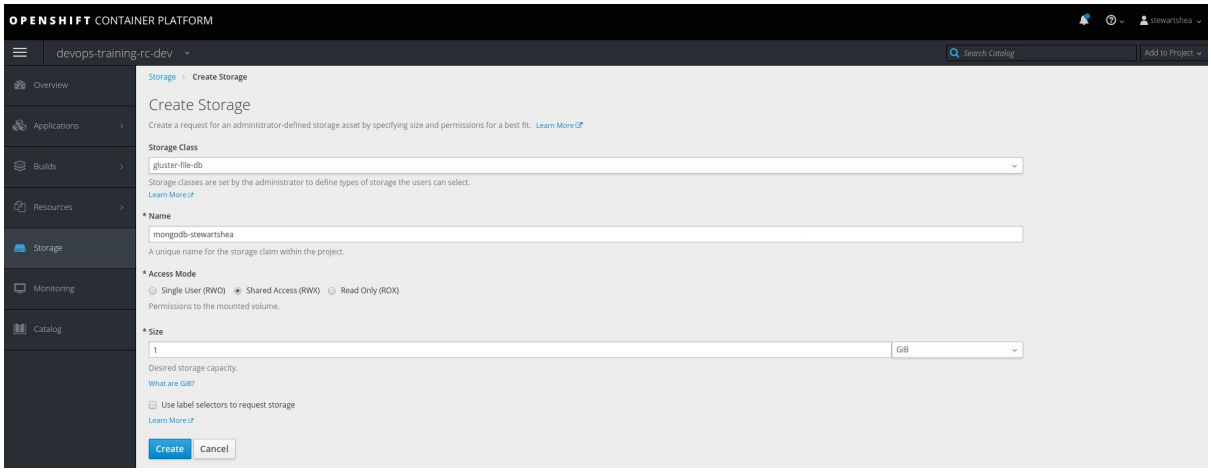
## RWX Storage

RWX storage allows muliple pods to access the same PV at the same time.

- Scale down `mongodb-[username]` to 0 pods




- Remove the previous storage volume and recreate as `gluster-file-db` with type RWX




- Scale down `mongodb-[username]` to 1 pods
- Redeploy with Rolling deployment

# Persistent Configurations

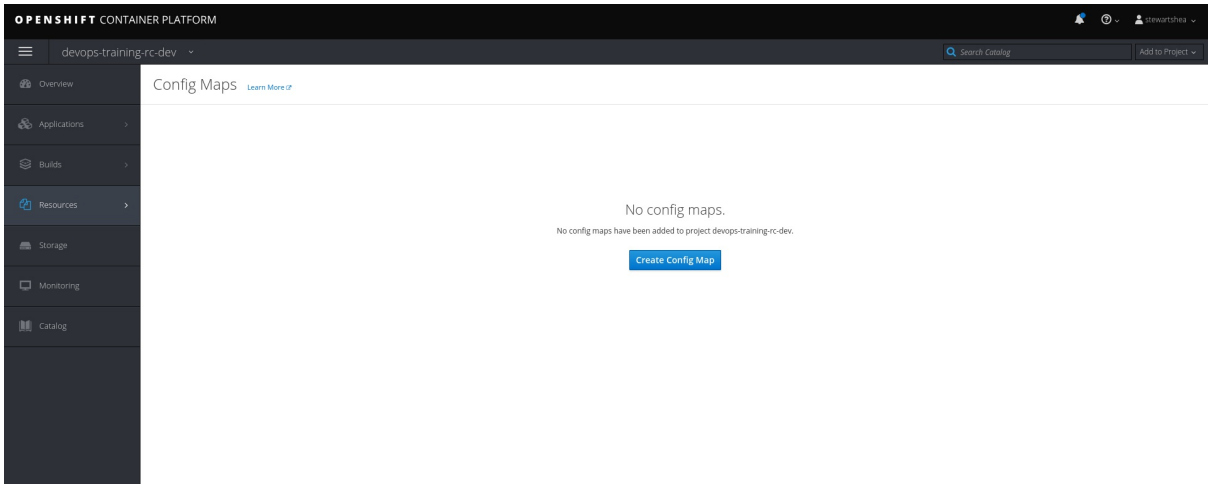
In cases where configurations need to change frequently or common configurations should be shared across deployments or pods, it is not ideal to build said configurations into the container image or maintain multiple copies of the configuration. OpenShift supports `configMaps` which can be a standalone object that is easily mounted into pods. In cases where the configuration file or data is sensitive in nature, OpenShift supports `secrets` to handle this sensitive data.

## ConfigMaps

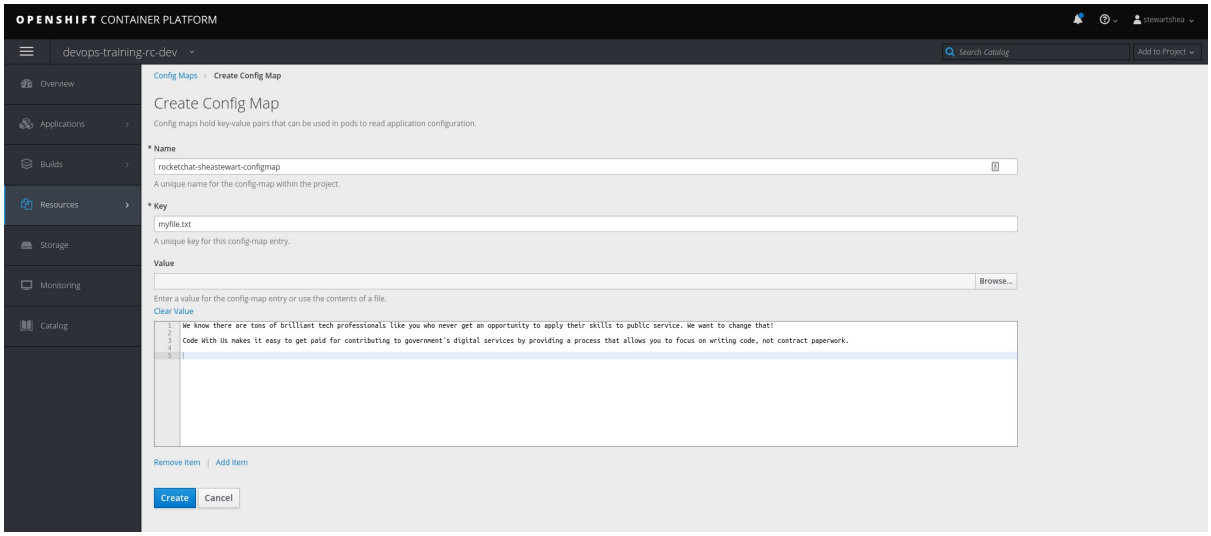
## Creating a Config Map and Adding it to a Deployment

Create a configMap with arbitrary data and mount it inside of your `rocketchat-[username]` pod:

- In the Web Console, create a `configMap` by navigating to `Resources -> Config Maps` *Note* this can be performed directly from the deployment, or independently as these steps illustrate

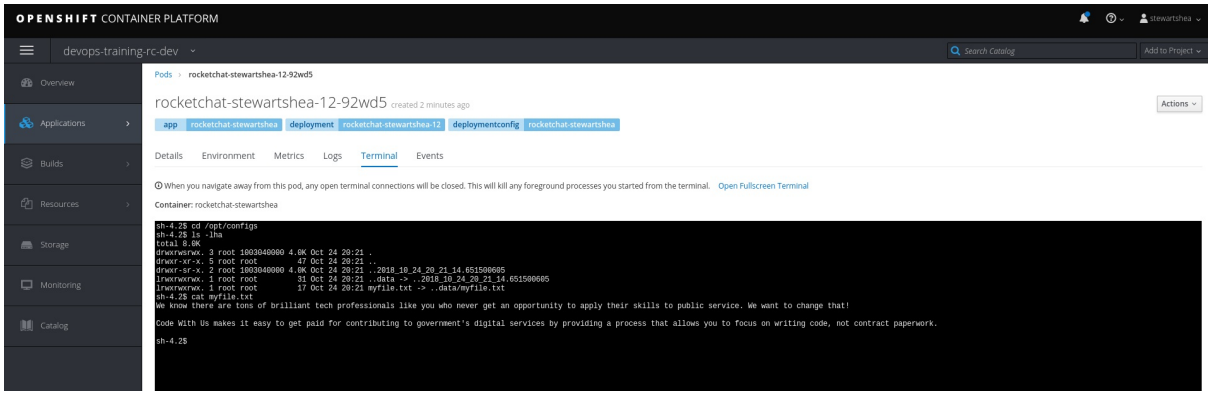



- Name your configmap `rocketchat-[username]-configmap` , add an arbitrary key (ie. file name) and content; add as many files as you like




- Attach the `configMap` to your `rocketchat-[username]` deployment by nativating to the `Configuration` tab and selecting `Add Configuration Files`

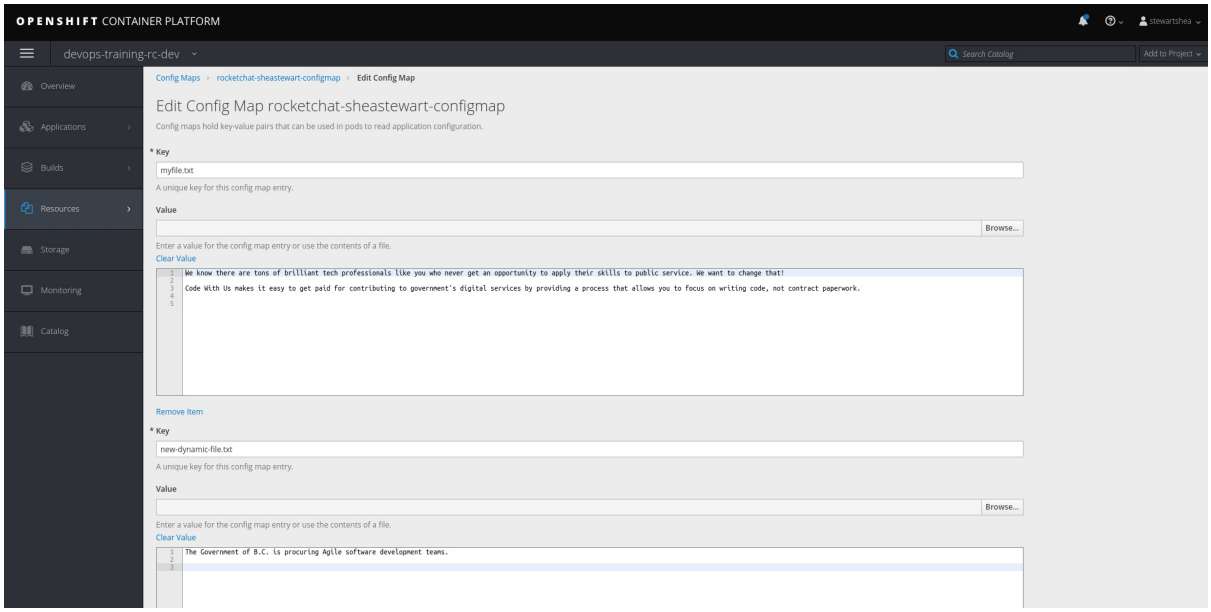
- This change will trigger a new deployment of your `rocketchat-[username]` pod
- Using the pod terminal in the Web Console or `oc rsh`, explore the path of the mounted configMap



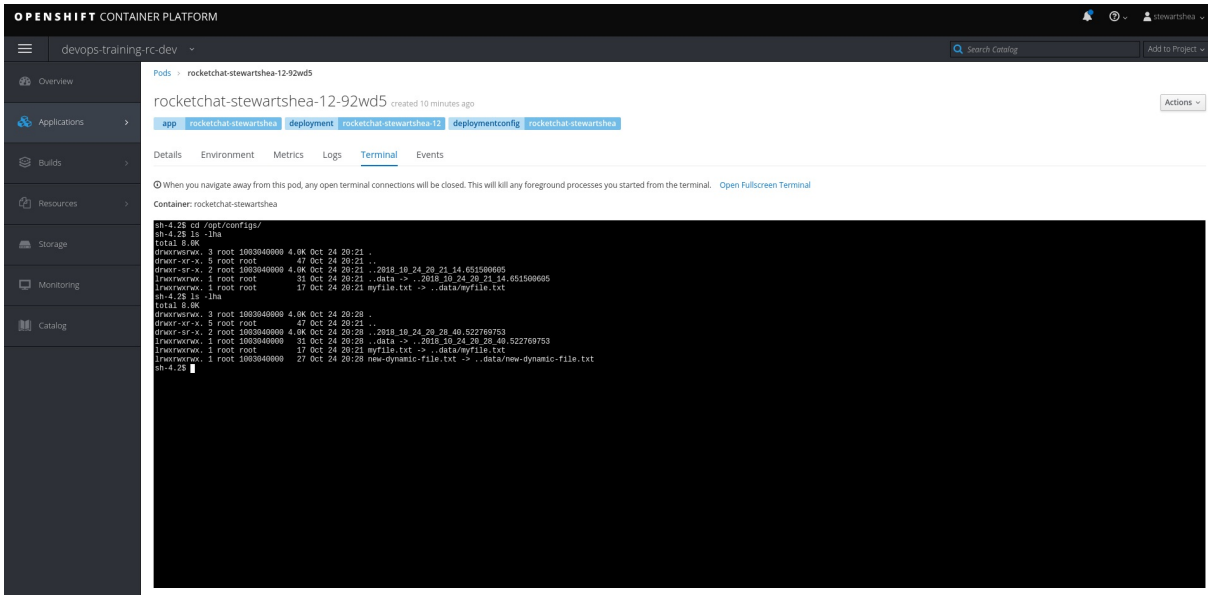

## Changing Config Map Content

The content in your `configMap` can be changed and is dynamically updated in the pod. With that said, if the application does not support live reload of its configuration, a new deployment will be required for the changes to be picked up.

- Edit your `configMap` to change the content of the file or add a new file




- Using the pod terminal in the Web Console or `oc rsh`, explore the path of the mounted configMap



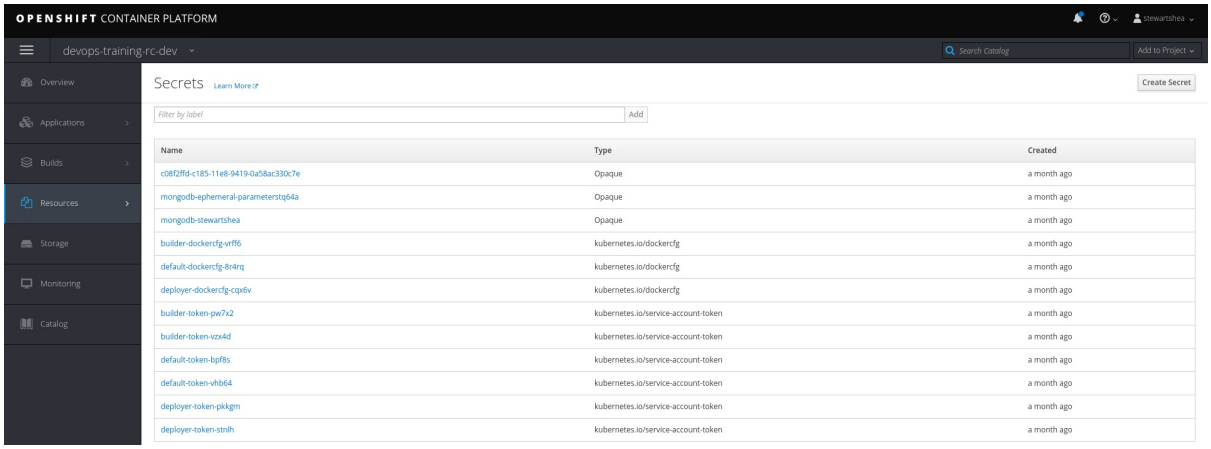

## Secrets

Secrets can be added in a similar way as config maps but are geared towards the management of sensitive information. In OpenShift, these are base64 encoded, and encrypted on disk when stored in the cluster. In Pods, they never live on disk (unlike configmaps) and are only in memory. Secrets, from the Web Console, are focused on supporting:

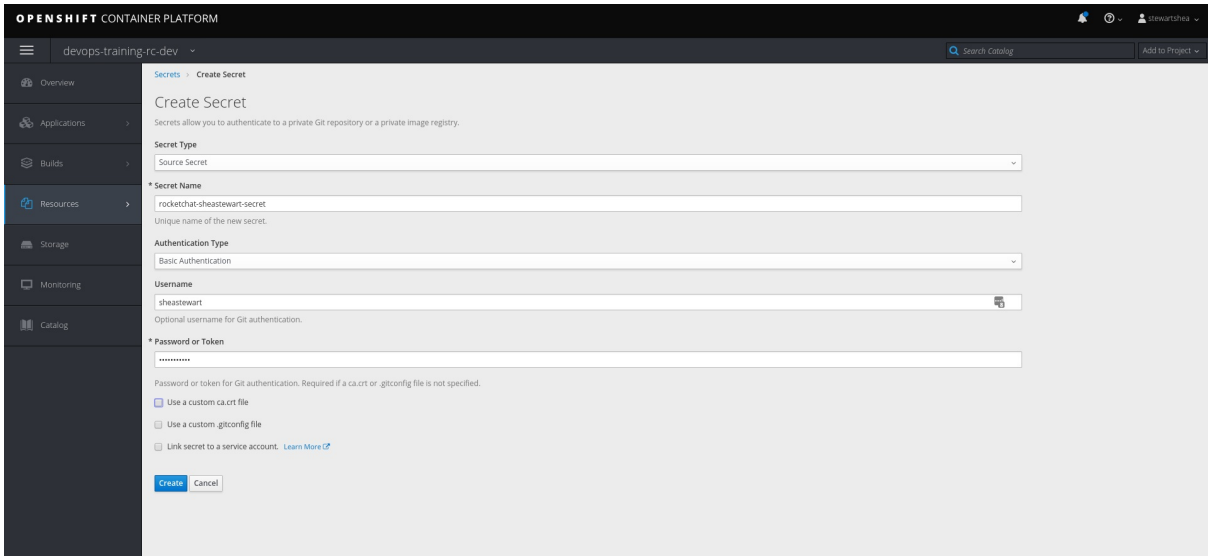
- Username/Passwords
- SSH Keys
- SSL Certificates
- Git config files

"Opaque" secrets are supported and can contain any type of data, however, these must be configured on the commandline with the `oc` cli.

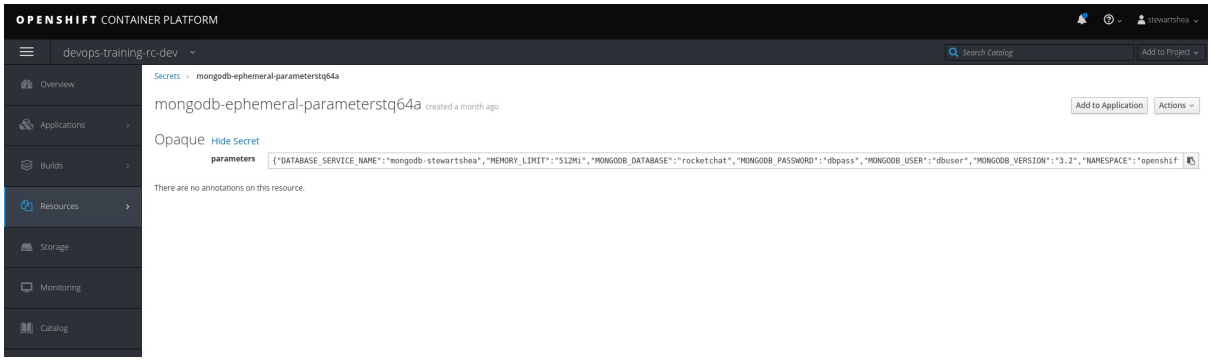
- In the Web Console, create a `secret` by navigating to `Resources -> Secrets -> Create Secret` *Note* this can be performed directly from the deployment, or independently as these steps illustrate



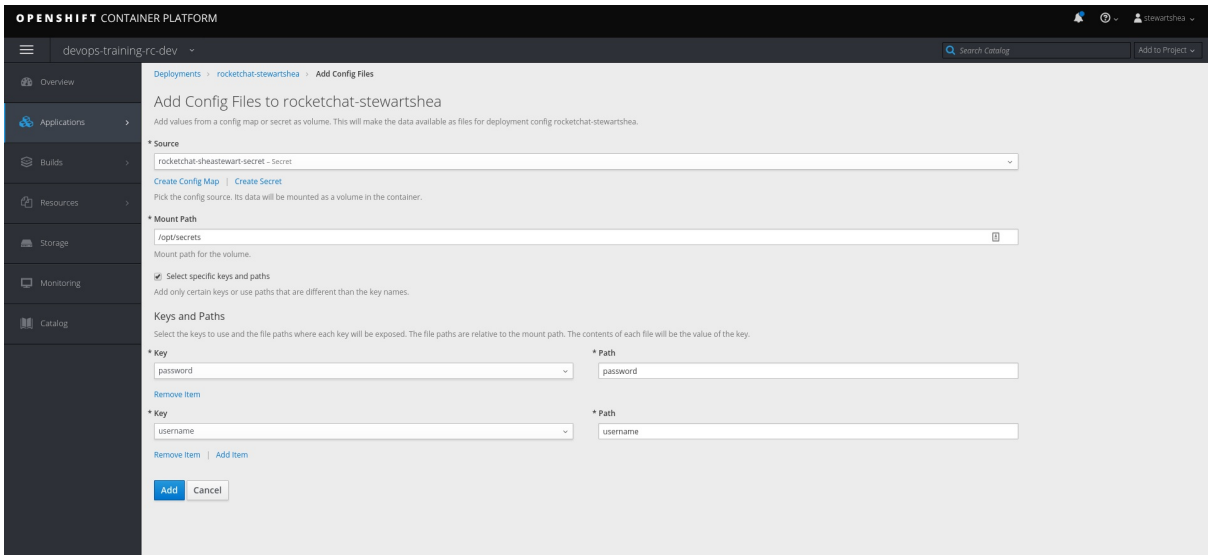

- Name your secret `rocketchat-[username]-secret` , add an arbitrary username/data or SSH key



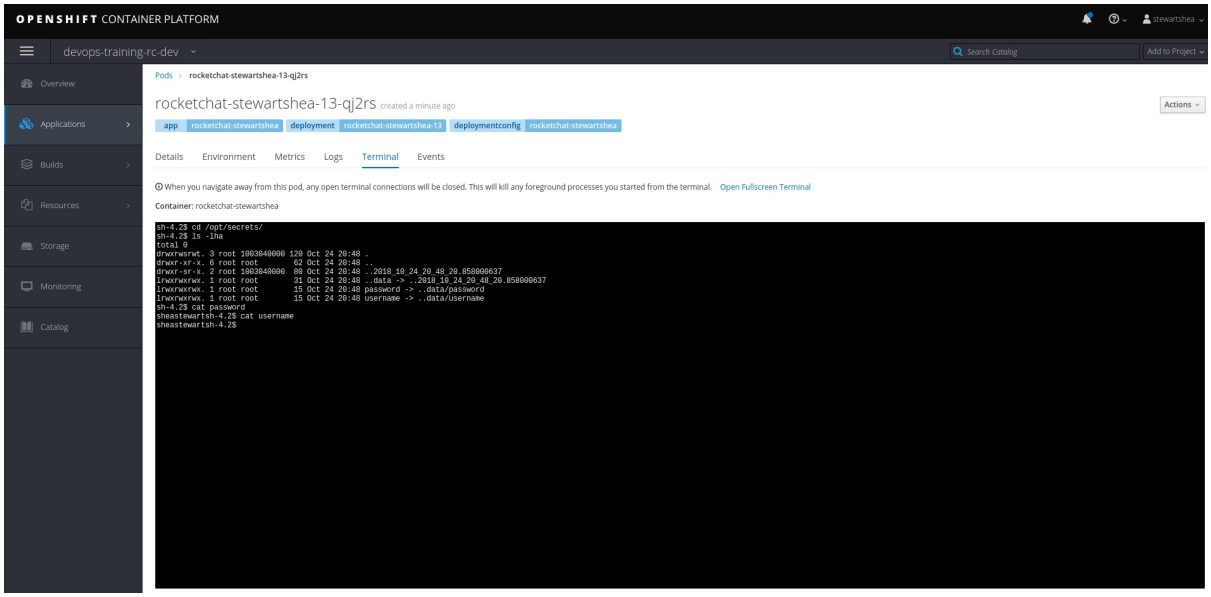

- Explore the other mongo secrets to see different variaions of secret data




- Attach the `secret` to your `rocketchat-[username]` deployment by nativating to the `Configuration` tab and selecting `Add Configuration Files`
- Select the `secret` and set the mount directory; split up the keys into separate files if you like




- This change will trigger a new deployment of your `rocketchat-[username]` pod
- Using the pod terminal in the Web Console or `oc rsh` , explore the path of the mounted `secret`




- From the cli, review the secret with `oc describe secret rocketchat-[username]-secret`

```
oc describe secret rocketchat-sheastewart-secret
Name:         rocketchat-sheastewart-secret
Namespace:    devops-training-rc-dev
Labels:       <none>
Annotations:  <none>

Type:  kubernetes.io/basic-auth

Data
====
password:  11 bytes
username:  11 bytes
```

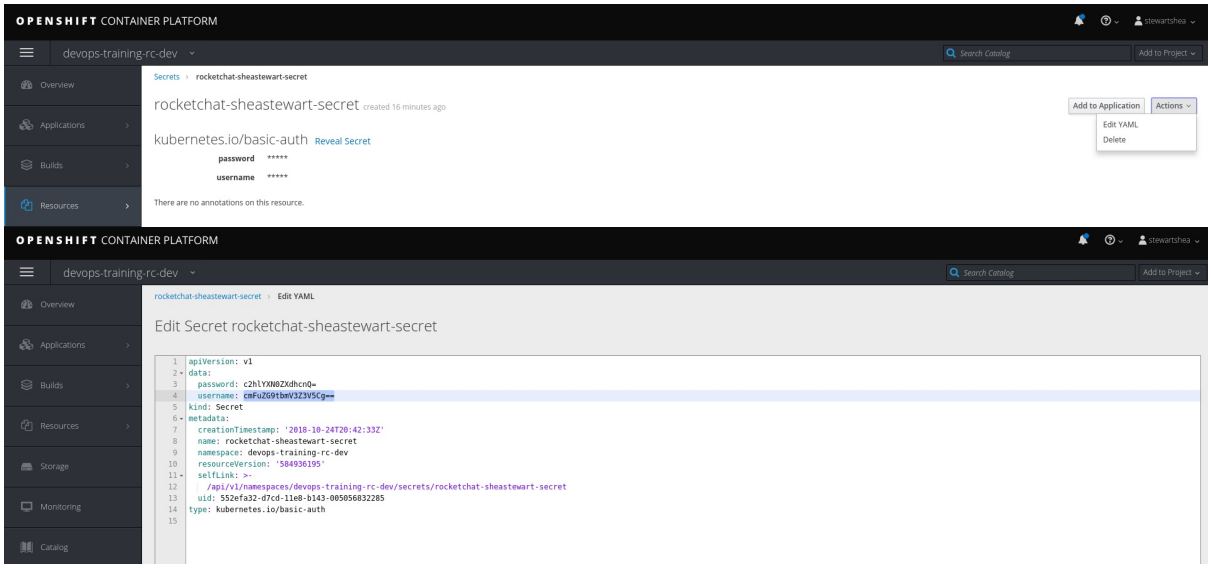
- Export the secret to view the contents with `oc get --export secret rocketchat-[username]-secret -o yaml`

```
oc get --export secret rocketchat-sheastewart-secret -o yaml
apiVersion: v1
data:
  password: c2hlYXN0ZXdhcnQ=
  username: c2hlYXN0ZXdhcnQ=
kind: Secret
metadata:
  creationTimestamp: null
  name: rocketchat-sheastewart-secret
type: kubernetes.io/basic-auth
```

- In order to reveal the contents of each key, base64 decode is required *Note* The Web Console automatically performs the base64 decode

```
echo "c2hlYXN0ZXdhcnQ=" | base64 -d
```

- To edit an existing secret, use `Edit Yaml` from the Web Console or `oc edit secret rocketchat-[username]-secret` from the cli

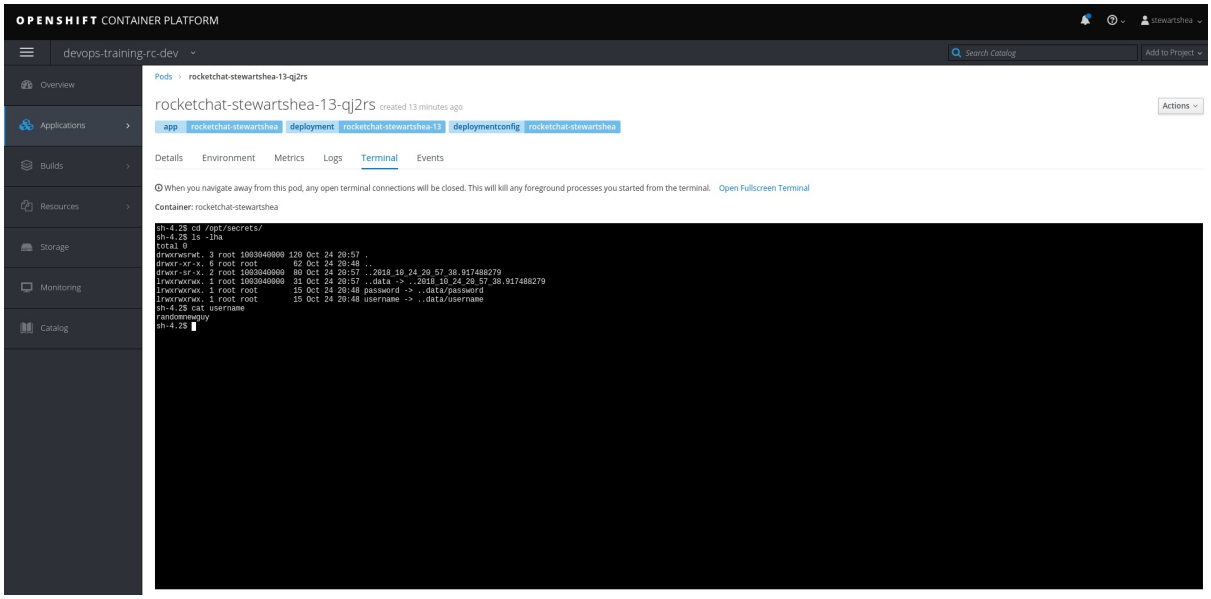



- The updated content must be base64 encoded manually if updating in-place

```
echo "randomnewguy" | base64
cmFuZG9tbmV3Z3V5Cg==
```

Or visit https://www.base64encode.org/ for encoding & decoding base64

- Similar to `configMaps`, the updated secret is automatically applied to the pod; no additional deployment is required



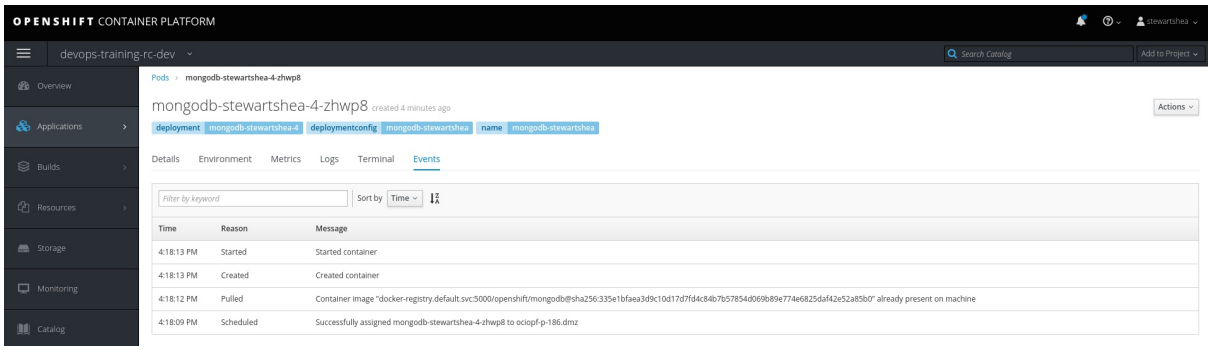
# Event Streams

Event streams exist on many objects as well as at the project level. The project is the highest level that a developer can explore to see all events with that particular project.

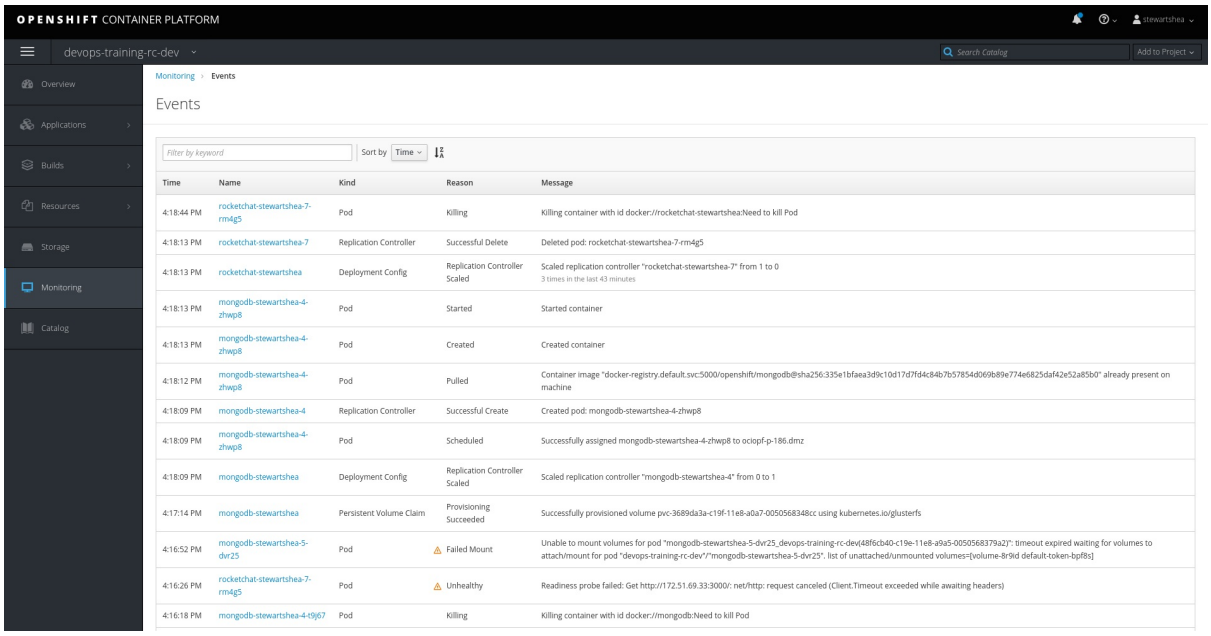
## Exploring Event Streams

The Web Console is the primary tool to visualize events sorted by time.

- Explore the events of a pod




- For project wide events, navigate to `Monitoring -> Events` and select `View Details`




- Or on the CLI in your project

```
oc get events --sort-by='.lastTimestamp'
```

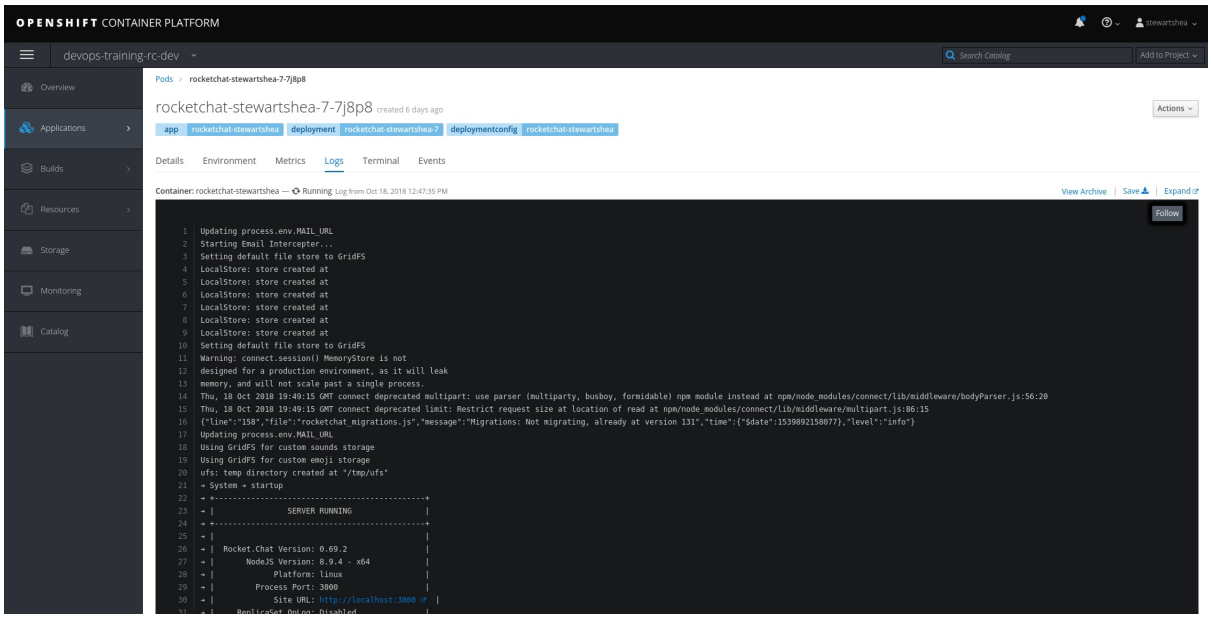
- Navigate through some of the events and review some of the output that could be helpful in debugging pods

# Debugging Containers

## Accessing Local Logs

Logs of a running pod can be accessed from the Web Console or from the `oc` cli:

- The `Logs` tab of any running pod can be used to view active logs for the current pod




- The `oc` command can be used to view or tail the logs:

```
oc logs -f <pod name>
```

If there is more than one container in a given pod, the `-c <container-name>` switch is used to specify the desired container logs.

## Using a Debug Container

## From the Web Console

In this lab, edit yaml of the `mongodb-[username]` deployment config.

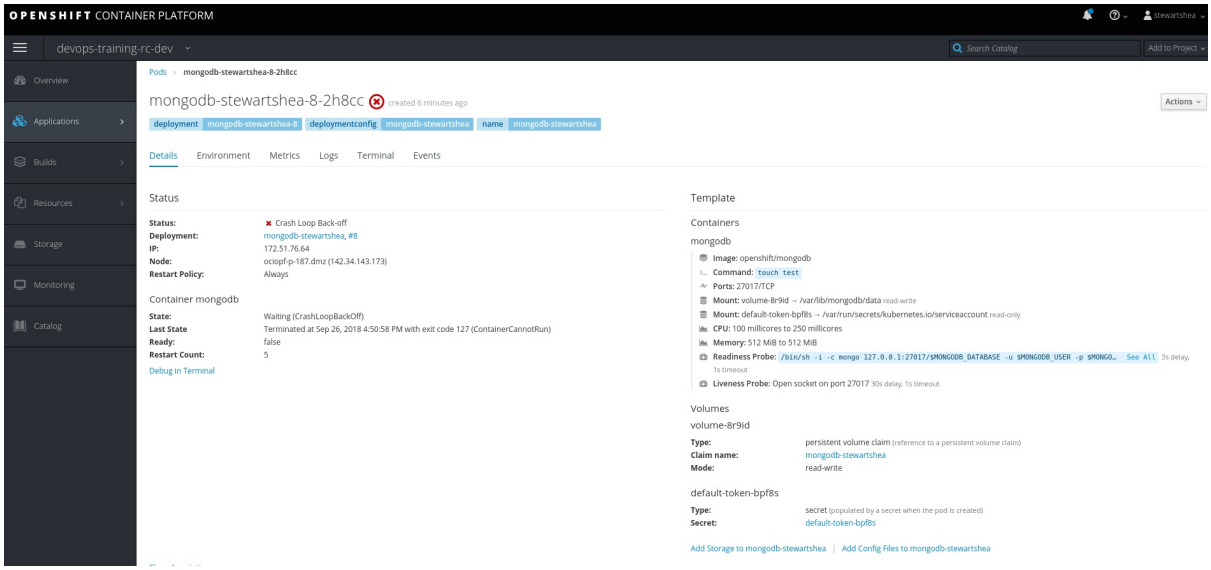
- Under `spec` and `containers`, locate the line:

```
            name: mongodb
```

- Insert the following line at the same indent level:

```
            command: ["touch test"]
```

- Once the deployment change takes effect, notice the CrashLoopBackoff




- Click on `Debug in Terminal`
- Explore your capabilities within this container
- Once done, remove the previously added command, and notice how its placement and structure changed.

## From the CLI

Using the `oc` command, start a debug container.

- Find the name of a pod you would like to debug

```
oc get pods
```

- Run the `oc debug` command to start a debug pod (your output will vary)

```
$ oc debug mongodb-1-s74g8
Defaulting container name to mongodb.
Use 'oc describe pod/mongodb-1-s74g8-debug -n crash-test' to see all of the containers in this pod.
Debugging with pod/mongodb-1-s74g8-debug, original command: container-entrypoint run-mongod
Waiting for pod to start ...
Pod IP: 10.131.22.12
If you don't see a command prompt, try pressing enter.
sh-4.2$
sh-4.2$
sh-4.2$ exit
exit

Removing debug pod ...
```

## RSH and RSYNC

RSH is available to all normal pods through the web console under the `Terminal` tab, as well as through the `oc rsh` command.

- With your choice of access, rsh into one of the application pods and test access within the namespace
  - cURL internal and external resources
  - Test internal name resolution, external name resolution, etc.
  - Explore your userid

RSYNC is also available in many pods, available through the `oc rsync` command.

- On the CLI, type `oc rsync -h`

- Using this command, copy the contents of the mongo data directory to your local machine, or from your machine to the remote pod

## Port Forwarding

The `oc port-forward` command enables users to forward remote ports running in the cluster into a local development machine.

- Find your pod and use the port forward command

```
oc get pods  | grep rocketchat-[username]
oc port-forward [pod name from above] 8000:3000
```

- Navigate to http://127.0.0.1:8000

# Logging and Visualizations

## EFK for Aggregated Logs

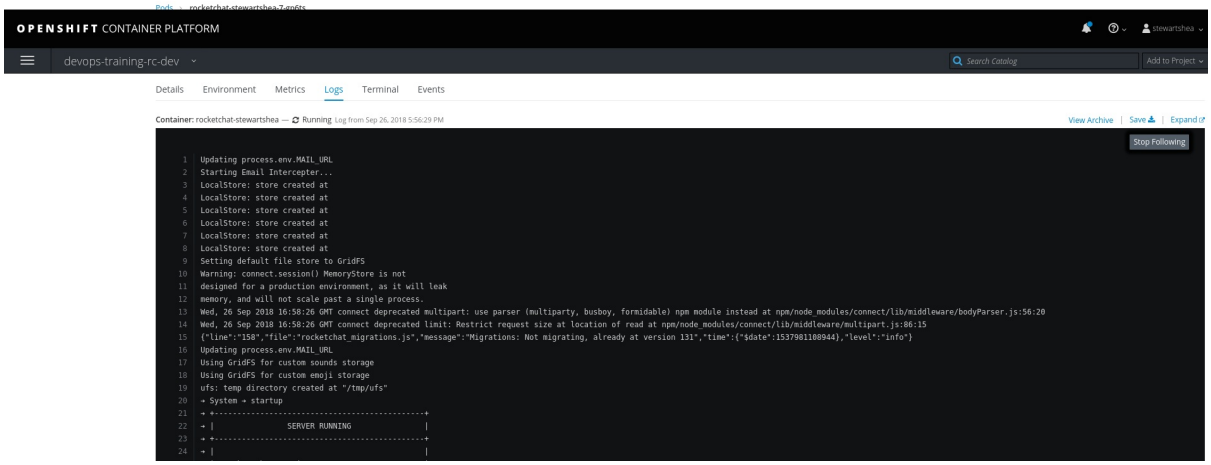
The OpenShift platform provides an aggregated logging stack that is automatically configured to centralize and store logs from application pods. These logs are only retained for a short period of time, currently about 14 days, but can be used to help identify issues with appliation pods.

Kibana is the primary interface for viewing and querying logs.

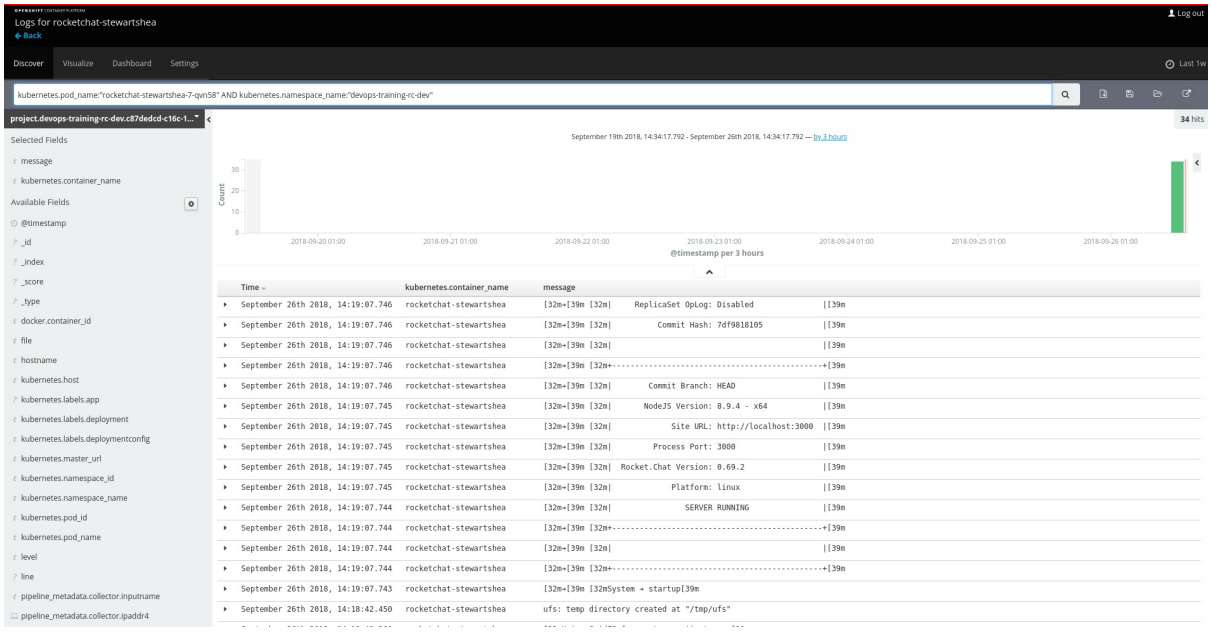
## Access the archive link from a pod

The shortcut towards accessing the Kibana is from the `Logs` tab of a running pod.

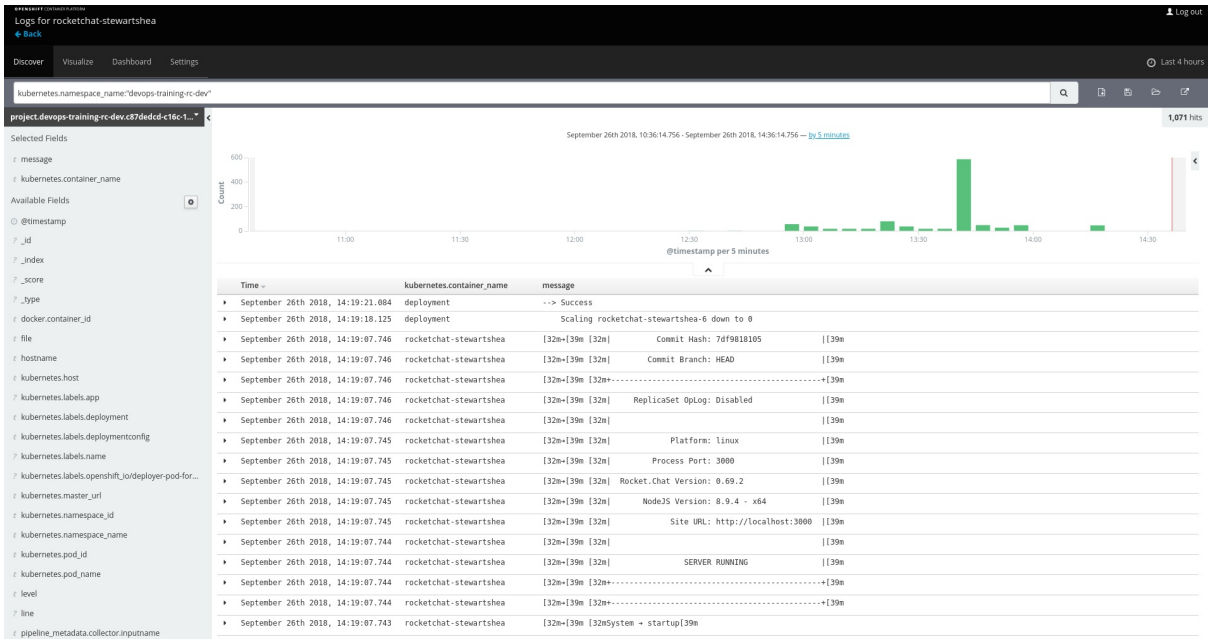
- Select the runing `rocketchat-[username]` pod and select the Logs tab



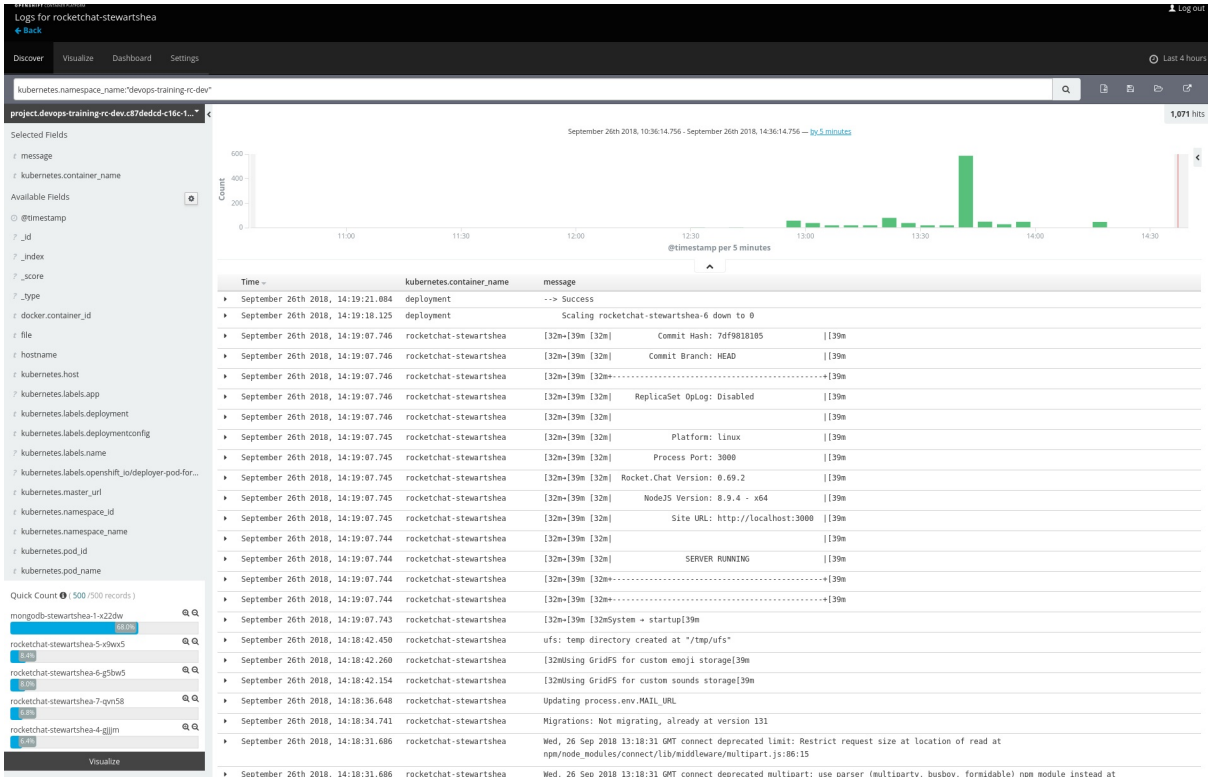

- Select the view archive link to be taken to Kibana
- Review the logging interface and the query that has been automatically populated



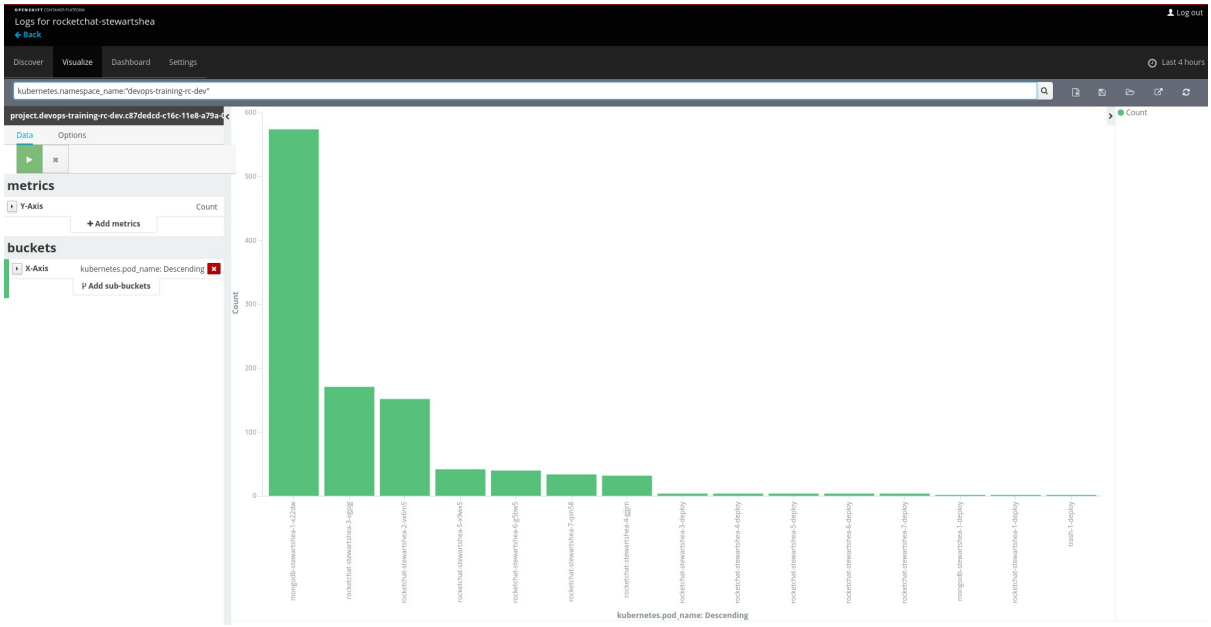
- Modify the query and time picker to select the entire namespace within the last few hours




- Review how Kibana surfaces key information about the log sources in the left panel




- Create a simple visualization of the information surfaced by Kibana




## Access the kibana interface directly

Kibana can also be accessed directly at the url:

- https://kibana.pathfinder.gov.bc.ca

The namespaces you have access to view will be directly related to your project permissions.

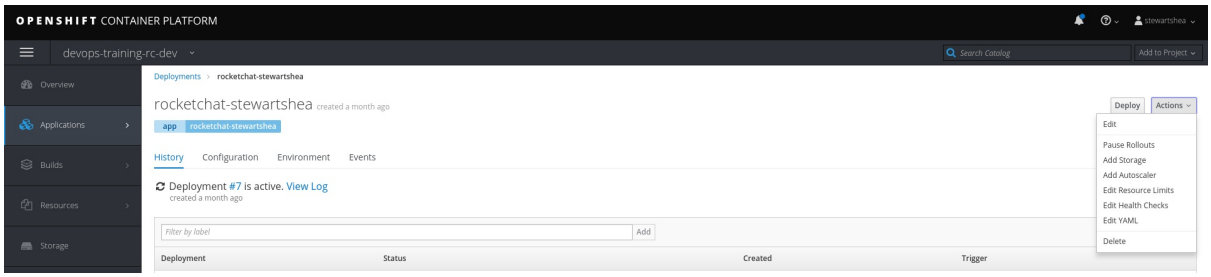
# Resource Requests and Limits

Tuning the resources assigned to a pod will have a direct effect on the performance of the application. Many templates include reasonable CPU and Memory resource configurations, however, new apps simply are deployed with the platform default.

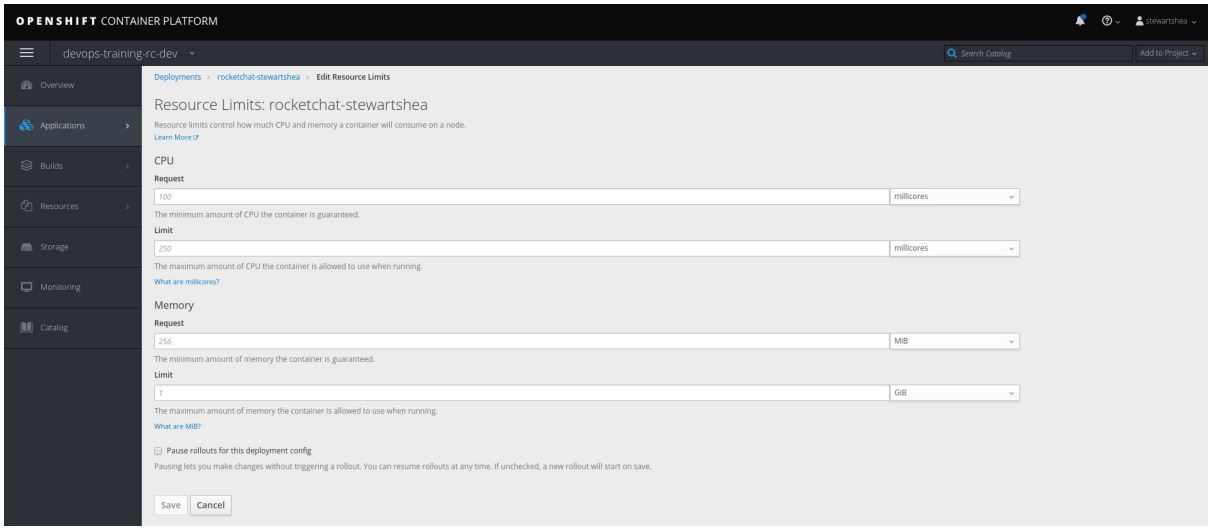
## Explore Default Resource Configurations

Since the Rocket Chat application was built from scratch and not deployed from a template, explore the current resources allocated to the pods:

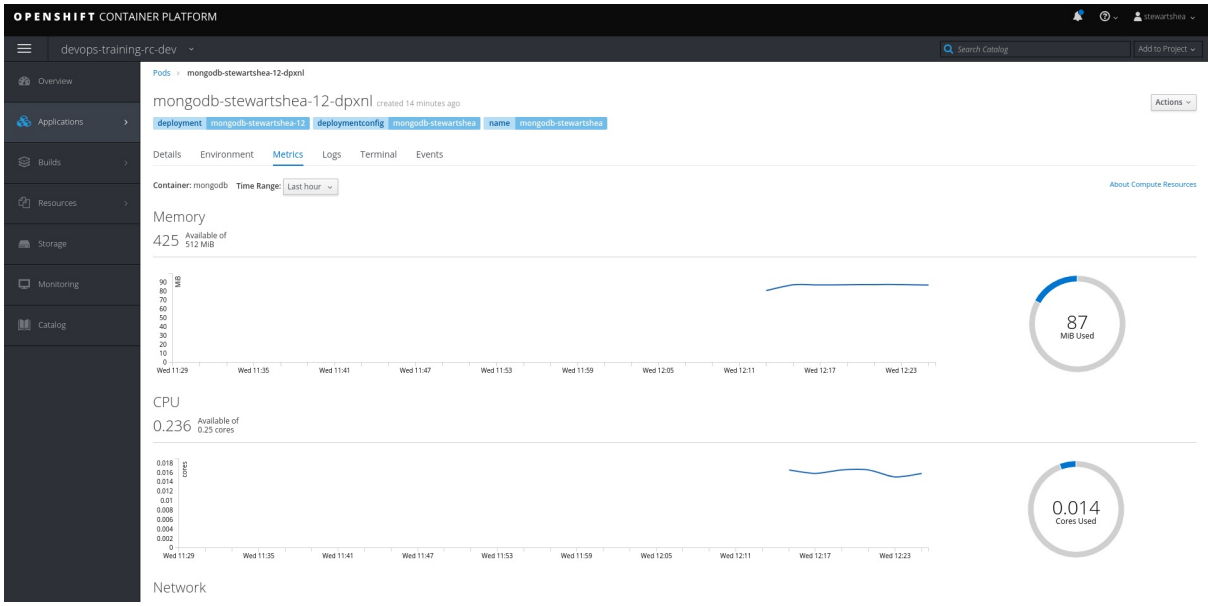
- Navigate to your rocketchat deployment and select `Actions -> Edit Resource Limits`



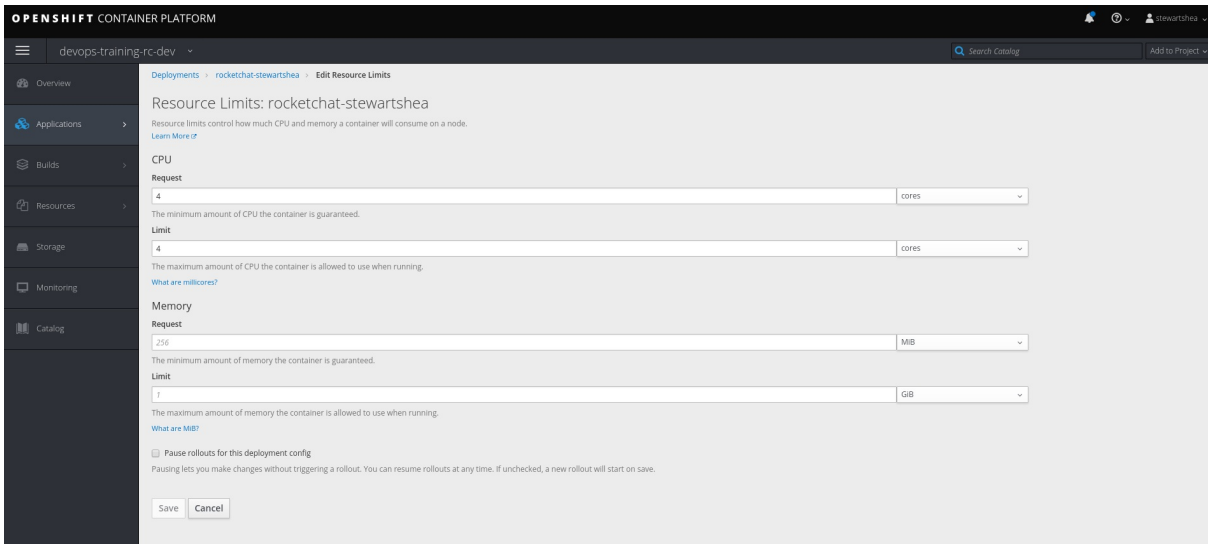

- Notice the defaults that are applied (in a light grey to indicate they are automatically set)




- Review the current metrics of your `rocketchat-<username>` pod




- Reduce the CPU request and limit to `50 millicores` and `100 Megabytes` and monitor the startup time of the pod
- Monitor the startup events of your pod and measure the time it takes to start
- Remove the limits previously imposed, and set your pod to 4 cores for the request and limit




- Monitor the status and speed of the new deployment
- Work with the rest of the class to determine why some pods may have succeeded, and others are failing.
- Remove the requests/limits previously set

# Pod Lifecycle Customization

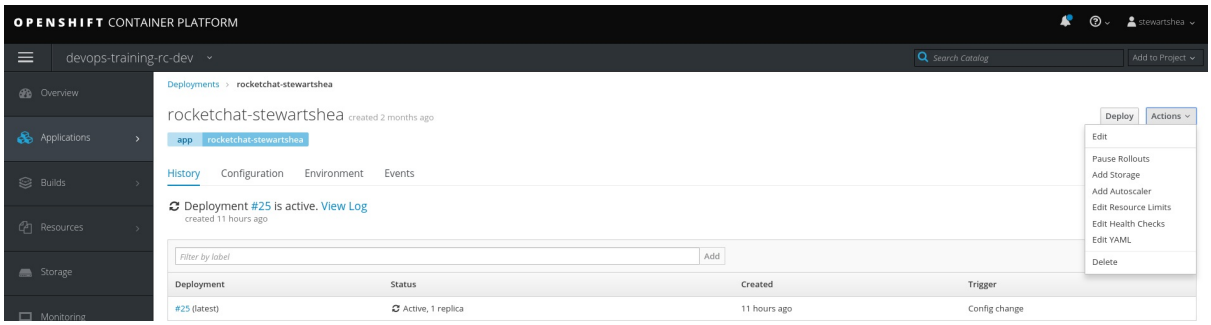
A Pod can be extended beyond the normal operation of the container by allowing developers to:

- add `init` containers
- add `pre` and `post` lifecycle hooks
- modify the default `entrypoint` of a container

## Init Containers

Init containers are best used to prepare the pod for normal operation. In this lab, you will add a simple init container that posts a message to rocketchat with your pod hostname.

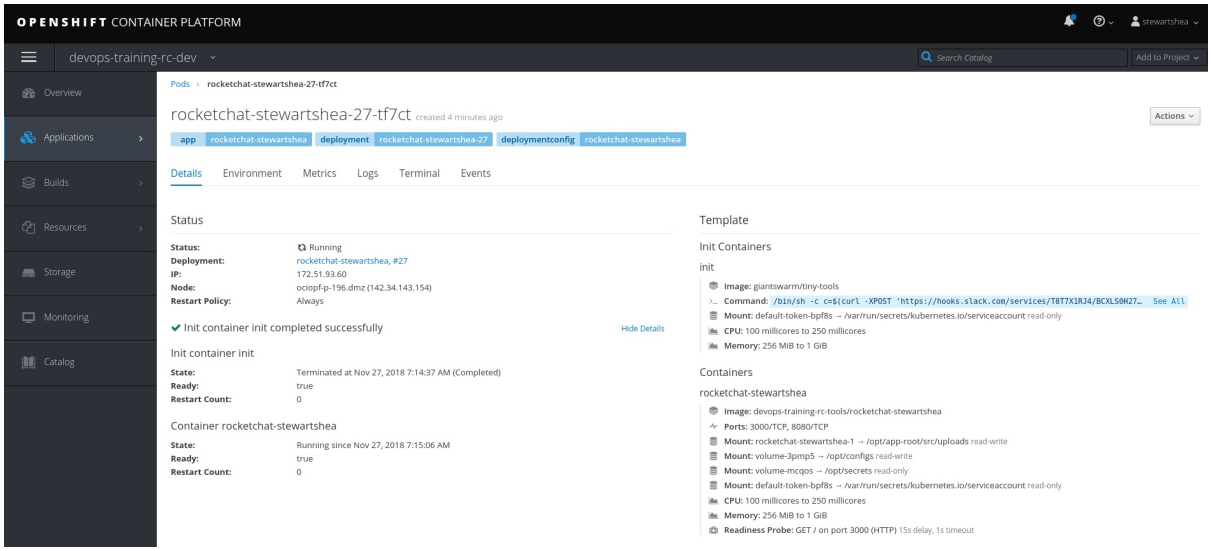
- From the Web Console, navigate to `Applications -> Deployments` and select your `rocketchat-[username]` deploymentconfig
  - If you wish to perform this from the cli with the `oc` tool, type `oc edit dc/rocketchat-[username]`
- Select `Actions -> Edit YAML`




- Add the following section of code under `spec: -> template: -> spec:`

```
      initContainers:
      - name: init
        image: giantswarm/tiny-tools
        command: ["/bin/sh", "-c", "c=$(curl -X POST -H 'Content-Type: application/json' --data '{\"text\":\"Sa
y Hello\"}' https://chat.pathfinder.gov.bc.ca/hooks/xxx/xxx)"]
```

- Select Save
- Ask the instructor to ensure the rocketchat instance is displayed to the class
- Explore the `Pod Details` to notice the differente with the Init Container




In order to obtain logs from the init container, the `oc` command can be used by specifying `-c init`:

```
oc logs rocketchat-[username]-[pod-id] -c init
```

## Lifecycle Hooks

Lifecycle hooks can be configured to start and stop a container properly. The lifecycle hook is tied directly to each container. Add a similar pre and post hook as the `initContainer` to demonstrate when it executes in your rocketchat deployment.

- From the Web Console, navigate to `Applications -> Deployments` and select your `rocketchat-[username]` deploymentconfig
  - If you wish to perform this from the cli with the `oc` tool, type `oc edit dc/rocketchat-[username]`
- Select `Actions -> Edit YAML`
- Add the following section of code under `spec: -> template: -> spec: -> containers`

```
          lifecycle:
            postStart:
              exec:
                command:  ["/bin/sh", "-c", "c=$(curl -X POST -H 'Content-Type: application/json' --data '{\"te
xt\": \"'\"$HOSTNAME\"' is at the postStart phase, huzzah! "}' https://chat.pathfinder.gov.bc.ca/hooks/xxx/xxx)
"]
            preStop:
              exec:
                command:  ["/bin/sh", "-c", "c=$(curl -X POST -H 'Content-Type: application/json' --data '{\"te
xt\": \"'\"$HOSTNAME\"' is just about to STOPPPPPP! \"}' https://chat.pathfinder.gov.bc.ca/hooks/xxx/xxx)"]
```

- Select Save
- Ask the instructor to ensure the rocketchat instance is displayed to the class

## Overriding the Entrypoint

It may be necessary, from time to time, to override the intitial command/entrypoint of a container image. Generally this is used for troubleshooting purposes, or to override a vendor provided image.

- From the Web Console, navigate to `Applications -> Deployments` and select your `rocketchat-[username]` deploymentconfig
  - If you wish to perform this from the cli with the `oc` tool, type `oc edit dc/rocketchat-[username]`
- Select `Actions -> Edit YAML`

- Add the following section of code under `spec: -> template: -> spec: -> containers`

```
          command:  ["/bin/sh", "-c", "c=$(curl -X POST -H 'Content-Type: application/json' --data '{\"text\":
\"'\"$HOSTNAME\"' is AN OVERRIDING COMMAND! \"}' https://chat.pathfinder.gov.bc.ca/hooks/xxx/xxx)"]
```

- Take note of the pattern that will happen in the rocketchat notification screen
- Remove the previous command to enable the rocketchat instance to start properly again

## References

- https://blog.openshift.com/kubernetes-pods-life/