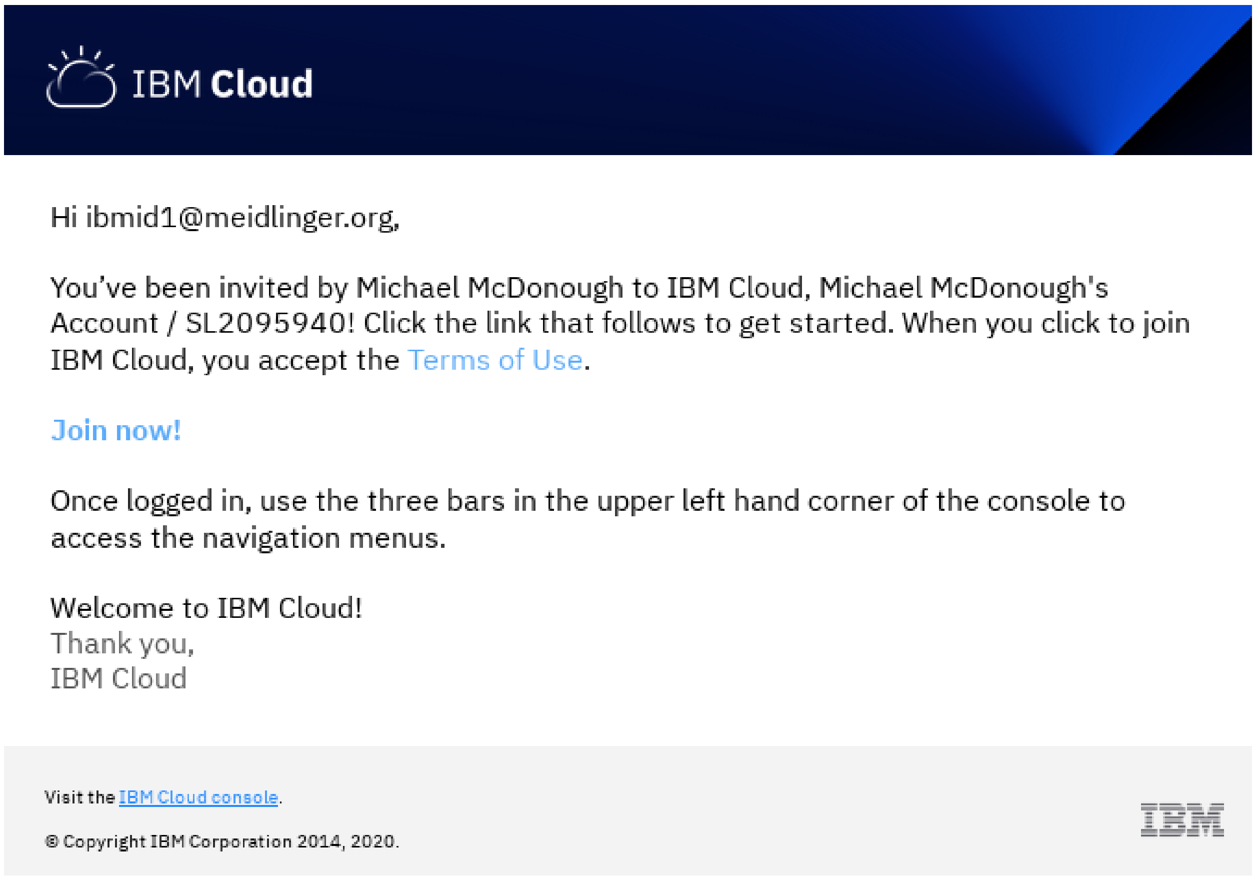
Stone Door Group and Red Hat OpenShift Technical Workshop

Migrating Legacy Java Applications to DevOps enabled OpenShift on IBM Cloud

17 March 2022
Virtual Event, Earth

Exercise 0: Establishing your IBM ID

You should have received an email similar to the following:



IBM Cloud

Hi ibmid1@meidlinger.org,

You've been invited by Michael McDonough to IBM Cloud, Michael McDonough's Account / SL2095940! Click the link that follows to get started. When you click to join IBM Cloud, you accept the Terms of Use.

Join now!

Once logged in, use the three bars in the upper left hand corner of the console to access the navigation menus.

Welcome to IBM Cloud!
Thank you,
IBM Cloud

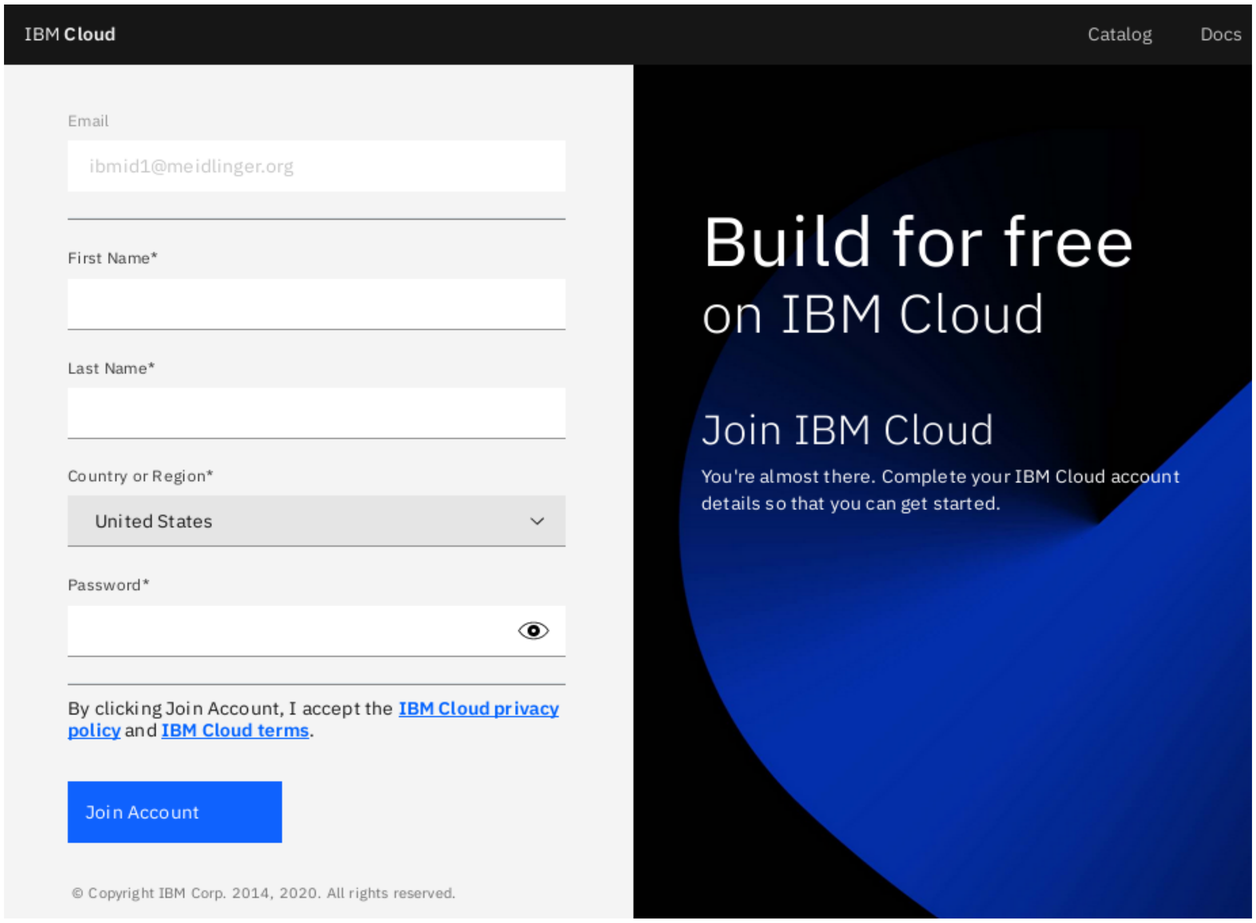
Visit the IBM Cloud console.

© Copyright IBM Corporation 2014, 2020.

1. Click the **Join now!** link in your email.

2. In the web form, add your first and last name, enter a password, and click **Join Account.**

   Note: The web form does not accept some special characters. If you use initials for your name (like "J. D. Doe"), remove spaces and periods so that only alphabetic characters are used in the form (example: "JD" for first name).



IBM **Cloud**

Catalog
Docs

Email
ibmid1@meidlinger.org

First Name*

Last Name*

Country or Region*
United States

Password*

By clicking Join Account, I accept the IBM Cloud privacy policy and IBM Cloud terms.

Join Account

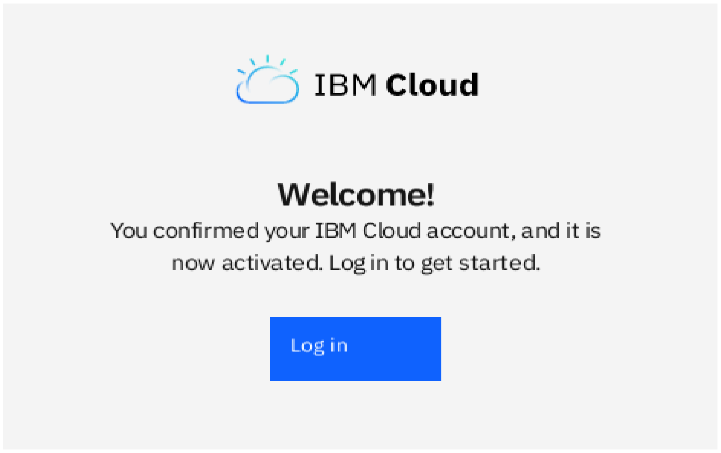
© Copyright IBM Corp. 2014, 2020. All rights reserved.

Build for free
on IBM Cloud

Join IBM Cloud

You're almost there. Complete your IBM Cloud account details so that you can get started.

On successful registration, you should see a confirmation similar to the following:



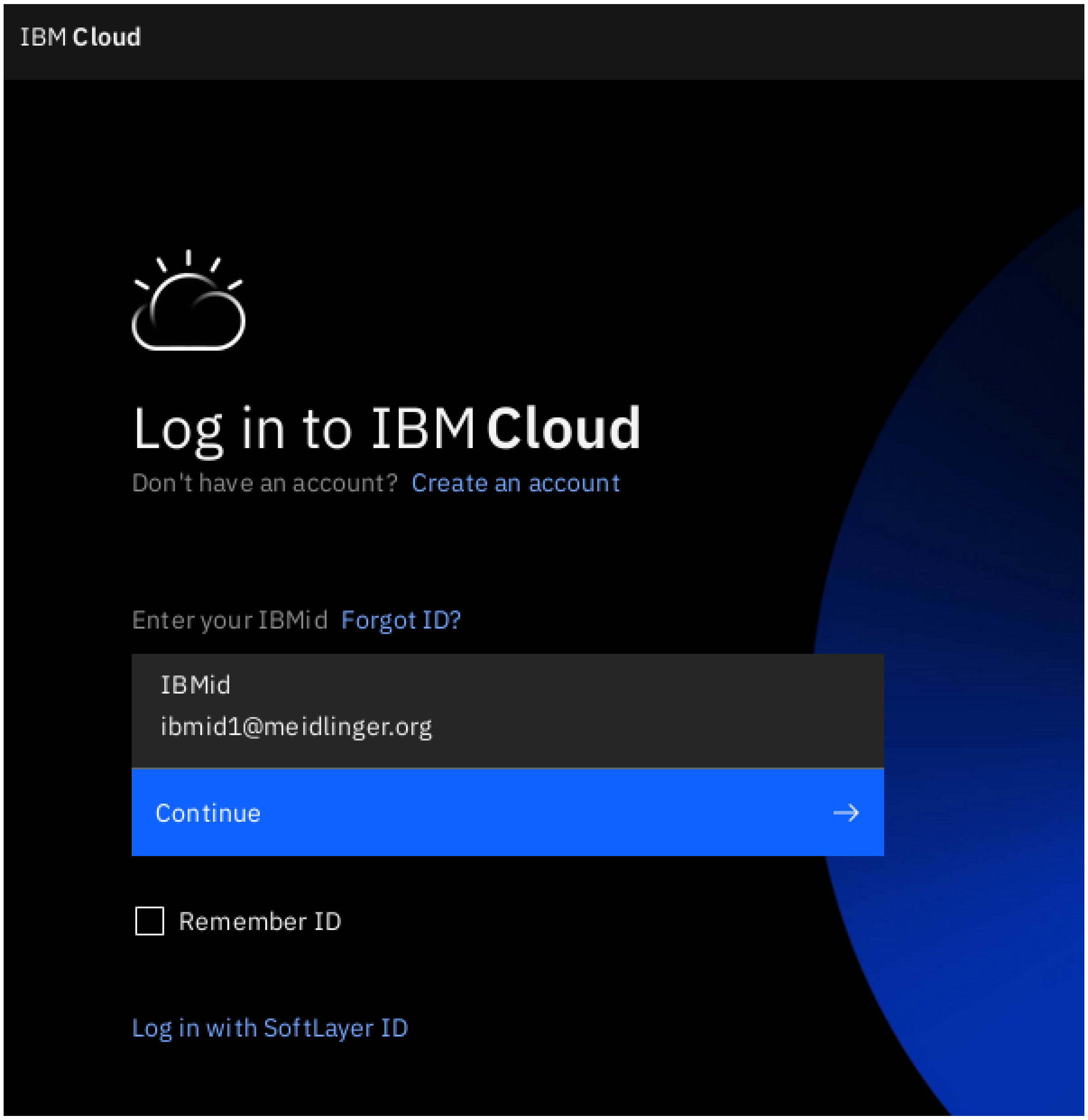
IBM **Cloud**

Welcome!

You confirmed your IBM Cloud account, and it is now activated. Log in to get started.

Log in

3. Click the **Log in** button to show the login page. Enter your IBM ID and click **Continue.**


IBM Cloud

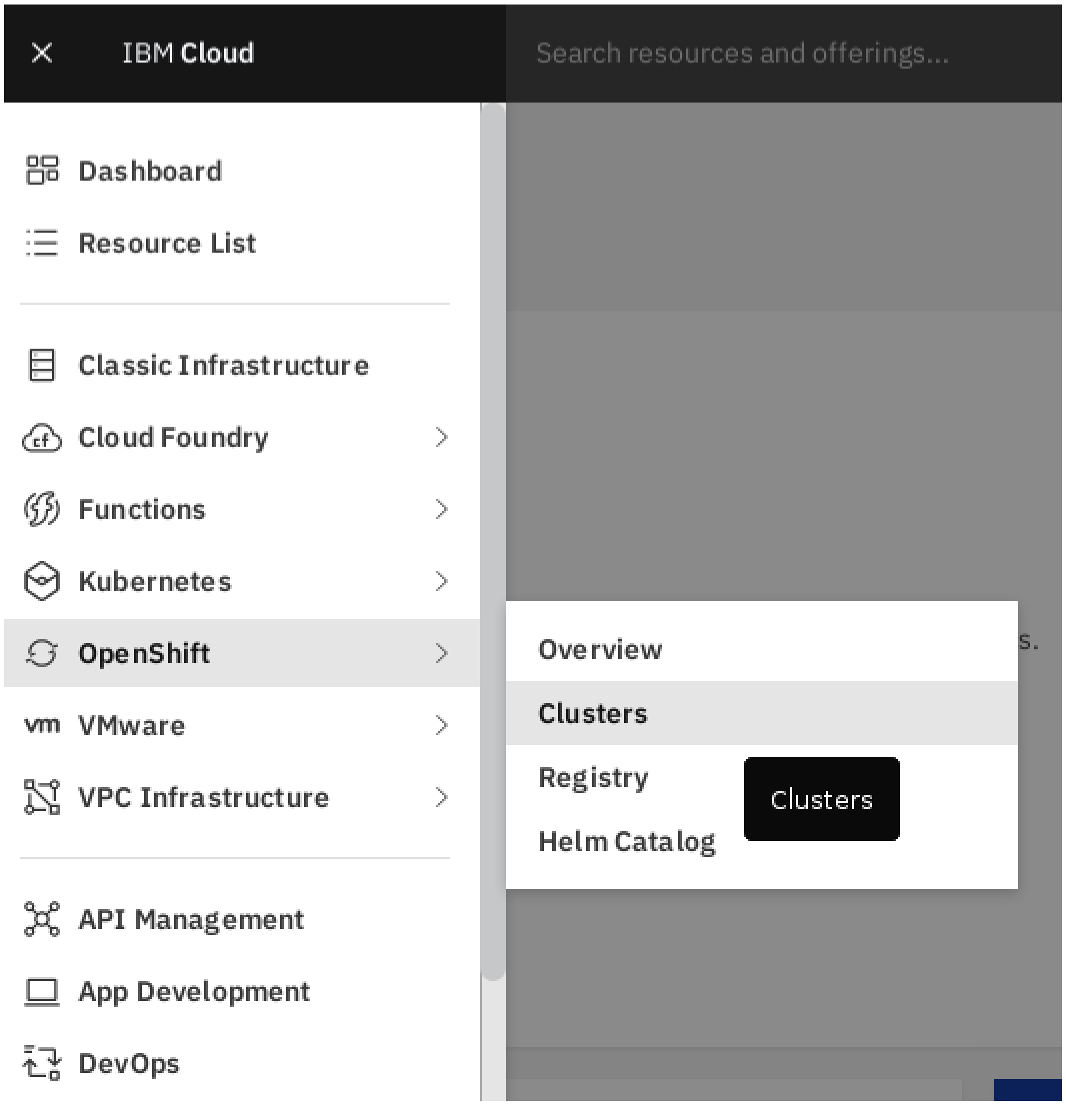
Log in to IBM Cloud

Don't have an account? Create an account

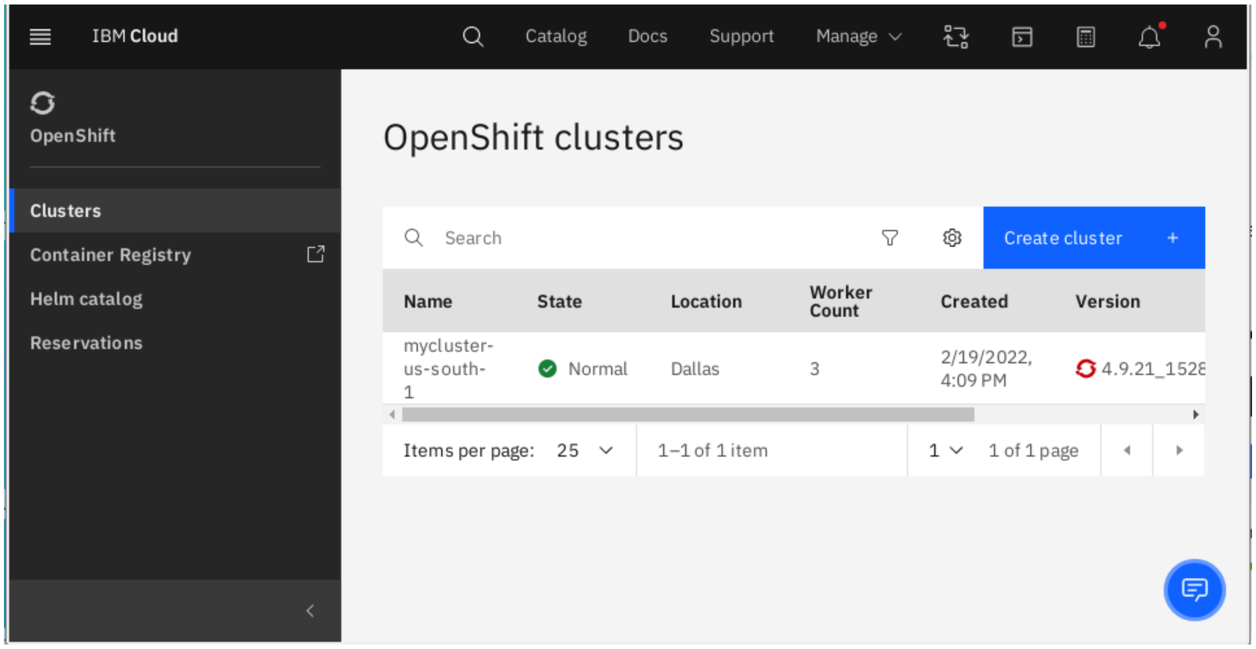
Enter your IBMid Forgot ID?

IBMid: ibmid1@meidlinger.org

4. After entering your password and logging into the IBM Cloud web interface, navigate to your clusters list by clicking the menu link in the upper left corner, then click **OpenShift > Clusters**.



5. Once you reach your clusters list, click **mycluster-us-south-1** to access the OpenShift cluster.

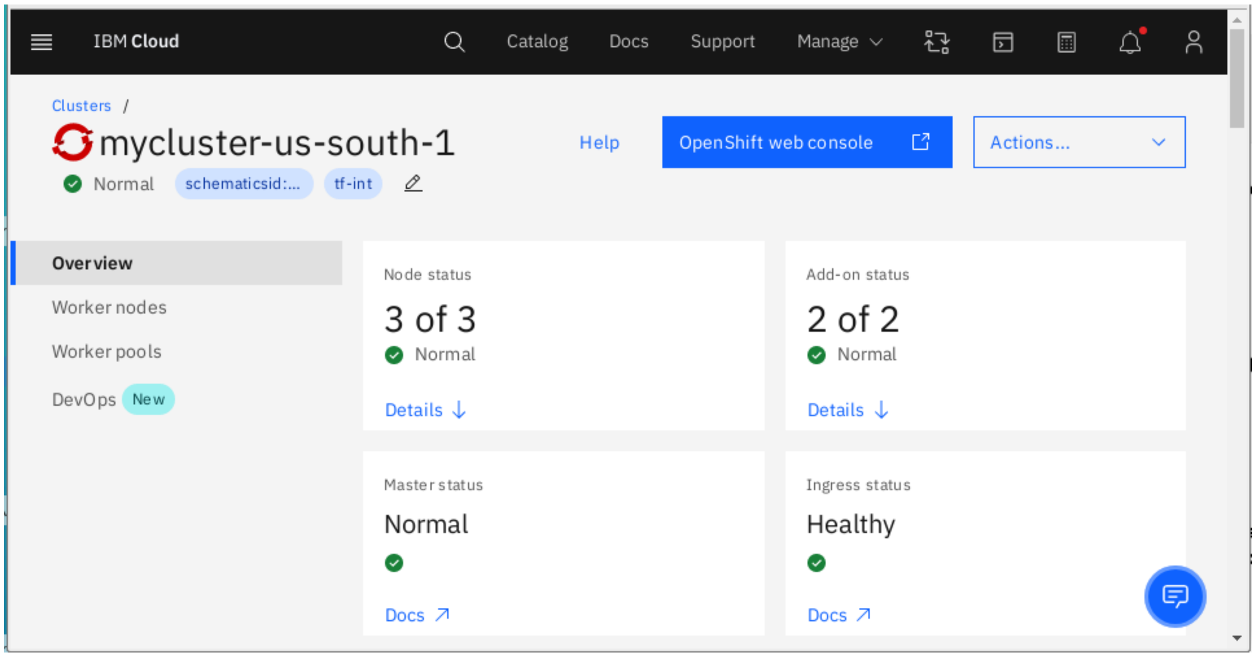


OpenShift clusters

Name	State	Location	Worker Count	Created	Version
mycluster-us-south-1	Normal	Dallas	3	2/19/2022, 4:09 PM	4.9.21_1528

Items per page: 25 | 1–1 of 1 item | 1 | 1 of 1 page

6. Finally, click the **OpenShift web console** button in the upper right to access the OpenShift web interface.



Node status
3 of 3
Normal
Details

Add-on status
2 of 2
Normal
Details

Master status
Normal
Docs

Ingress status
Healthy
Docs

Exercise 1: Verifying Access to the Environment

Log into the web console

1. Open your web browser and connect to the following URL.

   https://console-openshift-console.mycluster-us-south-764377-85853ed96992ad4cfd8a12f081018409-0000.us-south.containers.appdomain.cloud/k8s/cluster/projects

2. Enter the user name and password for your IBM ID.

Log into the bastion host

3. Using your SSH client, connect to the following host and provide the following credentials:

   bastion01.demo1.pd.stonedoor.io

   Username: the portion of your IBM ID which precedes the @ sign (all lowercase) and replace “+” with “-” (ex: “student+1001@stonedoorgroup.com” becomes username “student-1001”
   Password: **RedHat2022**

Connect to the OpenShift cluster using the ‘oc’ command from the bastion host

To connect to the OpenShift cluster on the command line, you will supply the URL of the OpenShift API which may be different than the web console URL.

4. Use the following commands to log into the OpenShift cluster and confirm that you can see the project created in step 3 above.

   Note: The backslash “\” is used to escape (ignore) the carriage return so that all the text below is considered to be on the same line.

   ```
   oc login -u <IBM ID> --insecure-skip-tls-verify=true \
      https://c115-e.us-south.containers.cloud.ibm.com:30465

   oc whoami
   oc get projects
   ```

Set INITIALS environment variable

Use the following command to set the INITIALS environment variable for your bastion host account:

echo “export INITIALS=<*your initials*>” >> ~/.bashrc
source ~/.bashrc

So if your initials are “C. J.” your command would look like this (please use lowercase letters):

echo “export INITIALS=cj” >> ~/.bashrc
source ~/.bashrc

Note: The **${INITIALS}** notation will be used throughout this document. Always replace the **${INITIALS}** with the same unique string.

Exercise 2: Deploying a Java Application using an OpenShift Template

Create a New Project for your Application

1. Use the following command from the bastion host to create your new project.

   oc new-project ${INITIALS}-java-template

2. Confirm that the project was created by running the following command.

   oc get projects

3. You can also confirm that the project was created by looking at the **Projects** page in the web console.

   https://console-openshift-console.mycluster-us-south-764377-85853ed96992ad4cfd8a12f081018409-0000.us-south.containers.appdomain.cloud/k8s/cluster/projects

Deploy the Application

4. Navigate to the application catalog for your project (change the ${INITIALS} in the URL to your initials).

   https://console-openshift-console.mycluster-us-south-764377-85853ed96992ad4cfd8a12f081018409-0000.us-south.containers.appdomain.cloud/catalog/ns/${INITIALS}-java-template

5. Select the **OpenJDK** template
6. Review the information about the application to be deployed and click **Create.**
7. On the **Administrator** perspective, select **Workloads > Pods**. You should see a pod named **openjdk-app-1-build**.
8. Click the **openjdk-app-1-build** pod and select the **Logs** tab. This will show you the log output of the application build process.
9. When the build has completed, you should see a message similar to the following.

   ```
   Successfully pushed
   image-registry.openshift-image-registry.svc:5000/${INITIALS}-java-template/java@sha256:d0bf05ee333421cdf1554dca588edf27f82f08a2b5d5fad2b14cfea25bbea0a9
   Push successful
   ```

10. When you see the above message, click **Workloads > Pods** again and confirm that you have a running application pod. It will have the same first two terms of the build pod (openjdk-app-1) but will then have a five character random string. It will look similar to the following.

    ```
    openjdk-app-1-bk9z5
    ```

11. Connect to the route for the application and confirm that the web site is reachable from a browser.

    Note: you can find the routes for all applications in your project by navigating to **Networking > Routes** in the OpenShift Web Console's left menu.

    http://openjdk-app-${INITIALS}-java-template.mycluster-us-south-764377-85853ed96992ad4cfd8a12f081018409-0000.us-south.containers.appdomain.cloud/

    You should see the phrase "Hello World" in your browser.

Exercise 3: Working with the Deployed Application

Gathering Information

1. From the bastion host command line, run the following command to see some of the resources created by the application template in exercise 2.

   ```
   oc get all
   ```

   You should see the following resources:
   - Pod
   - Service
   - Route
   - ImageStream
   - BuildConfig
   - Build
   - DeploymentConfig

2. In addition to the above resources, you can see other resources and information with the following commands. Run these commands to learn more about these resources.

   ```
   oc get serviceaccounts
   oc get secrets
   oc get events
   oc describe pod <name of pod>
   oc describe service openjdk-app
   oc describe deploymentconfig openjdk-app
   oc describe buildconfig openjdk-app
   ```

Viewing Logs

3. In exercise 2, step 10, you viewed the logs for the build pod. Navigate to the application pod in the web console and view its logs.
4. Run the following command from the bastion host and compare the log output from the CLI to the log output in the web console.

   ```
   oc logs  openjdk-app-1-build
   ```

Interacting with Containers

5. Run the command below to find the pod name for your running application. Look for the pod with Status = Running and note its name, which will end with a 5 character random string.

   oc get pods

6. Run the following commands to log into your running container

   oc rsh openjdk-app-1-*RRRRR* (where RRRRR is a 5 character random string)

7. Switch to the **bash** shell.

   exec bash

8. Note the running container's UID and GID.

   id

   Notice that the UID is a non-privileged user and that the GID is root. This is the standard configuration for containers running in OpenShift.

9. Exit the pod and delete it. Watch the replication controller spin up a new pod.

   exit
   oc delete pod openjdk-app-1-RRRRR
   oc get pods

Making Changes at Run Time

10. Run the following command to edit the Deployment.

    Note: The default editor in this configuration will be **vi**. If you are not familiar with vi, you may skip this step. If you prefer the **nano** editor, you may optionally run this command first to set that as your default editor: **export OC_EDITOR=nano**.

    oc edit deploymentconfig openjdk-app

11. On line 18, change the number of replicas to **2**. Save and quit, then observe the second pod spinning up automatically with the following command.

```
oc get pods -w
```

The **-w** switch means watch. Updates to the status of pods will be shown on the command line. When you see the new pod running, you can break out of the command with **Ctrl-C** and get the current state of the pods with **oc get pods** without the **-w** switch.

12. Run the following command to change the number of running pods without editing the DeploymentConfig.

```
oc scale --replicas=3 deploymentconfig openjdk-app
oc get pods
```

13. Run the following command to change the number of running pods back to “1” without editing the Deployment.

```
oc scale --replicas=1 deploymentconfig openjdk-app
oc get pods
```

Exercise 4: Cloning an Application into Your Own Git Repository

Create a User Account

If you already have an account on github.com, you may use that account for these lab exercises. If you do not already have an account, you can sign up for one at github.com at the following URL.

The webhooks configured in this workshop are specific to github.

https://github.com/join

Create Your Repository

1. Create a new repository called **openjdk-app-sdg-workshop**. This should be a public repository so that you do not need to configure authentication for these exercises. Select **Initialize this Repository with a README** (or similar for your git server).
2. On the bastion host, clone the repo. You can get the clone URL from the repository itself by clicking the **Clone** button and selecting **Clone with HTTPS**.

```
cd $HOME
```

```
git clone <URL>
```

You should have a directory in your bastion host home directory named **openjdk-app-sdg-workshop**

Clone the Original Repository to the Bastion Host

3. Run the following command from your home directory on the bastion host to clone the example application.

   ```
   git clone https://github.com/jboss-openshift/openshift-quickstarts
   ```

   You should now have a directory named **openshift-quickstarts** in your bastion host home directory.

Push the Code to Your Repository

4. Copy the code from the **openshift-quickstarts** directory to the **openjdk-app-sdg-workshop** directory.

   ```
   cp -r ~/openshift-quickstarts/* ~/openjdk-app-sdg-workshop/
   ```

5. Upload the code to your git server.

   ```
   cd ~/openjdk-app-sdg-workshop/
   ```

   Update the git config with your information (must match your git login)

   ```
   git config --global user.email "you@example.com"
   git config --global user.name "Your Name"
   git config --global push.default simple
   ```

   Add code, commit, and push.

   ```
   git add *
   git commit -m “initial upload”
   git push
   ```

   Enter your username and password when prompted.

Exercise 5: Using the ‘oc new-app’ Command to Deploy an Application

Create a New Project for this version of the Application

1. From the bastion host command line, run the following command to create a new project.

   ```
   oc new-project ${INITIALS}-new-app
   ```

Deploy the Application

2. Run the following command to deploy the Java code from the git repository you created in exercise 4.

   ```
   oc new-app  java:latest~https://<repo_url> --context-dir=/undertow-servlet
   ```

   Note: include the **.git** extension the repo_url.

Expose the Service

The template that was deployed in exercise 2 included a route for the application. The **oc new-app** command will create several of the resources to support an application on OpenShift but it does not create a route. You will have to create a route by exposing the service with the following command.

```
oc expose svc openjdk-app-sdg-workshop
```

Verify the Application Deployment

3. Get the route for the application and connect to it in your browser. Do you see the application?

Exercise 6: Rebuilding an Application after Code Change (manually)

Update Code and Push it to Repository

1. Edit the home page of the application and change the **Hello World** text to **Hello IBM Cloud World.** If you are not familiar with **vi** or other GNU/Linux editors, **nano** is relatively easy to use.

   ```
   cd ~/openjdk-app-sdg-workshop
   nano
   ./undertow-servlet/src/main/java/org/openshift/quickstarts/undertow/servlet/ServletServer.java
   ```

   ctrl-w (opens search box)
   Hello (search for overview)

   change to **Hello IBM Cloud World**

   ctrl-x (exit)
   Y (confirm save)
   <Enter> (exit editor)

2. Add, commit, and push the update.

   ```
   git add
   ./undertow-servlet/src/main/java/org/openshift/quickstarts/undertow/servlet/ServletServer.java
   git commit -m "updated Hello World heading"
   git push
   ```

Use ‘oc start-build’ to Re-build the Application

3. Run the following command to start a new build with the updated code.

   ```
   oc start-build buildconfig.build.openshift.io/openjdk-app-sdg-workshop
   ```

4. You can run the following command to follow the build progress.

   ```
   oc logs -f build/openjdk-app-sdg-workshop-2
   ```

Verify the Change

5. Connect to the route for the application after the new pods are launched and confirm the update to the web page.

Exercise 7: Deploy Sample Application for OpenShift Pipelines

Now that you are familiar with deploying and managing applications manually, let’s look at managing applications with OpenShift Pipelines. You will use a simple application during this tutorial, which has a frontend and backend.

1. Create a project for the sample application that you will be using in this tutorial:

   oc new-project ${INITIALS}-pipelines-tutorial

2. Run the following command to see the pipeline service account:

   oc get serviceaccount pipeline

   OpenShift Pipelines automatically adds and configures a ServiceAccount named pipeline that has sufficient permissions to build and push an image. This service account will be used later in the tutorial.

3. Install the apply-manifests and update-deployment tasks from the openshift/pipelines-tutoral repository.

   oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/01_pipeline/01_apply_manifest_task.yaml -n ${INITIALS}-pipelines-tutorial

   oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/01_pipeline/02_update_deployment_task.yaml -n ${INITIALS}-pipelines-tutorial

4. List the tasks:

   oc get tasks

   You should see output similar to the following:

```
NAME              AGE
apply-manifests     10 seconds ago
update-deployment   4 seconds ago
```

5. We will be using **buildah** ClusterTasks, which gets installed along with the OpenShift Pipelines Operator. To list ClusterTasks on the cluster, use the following command.

   oc get clustertasks

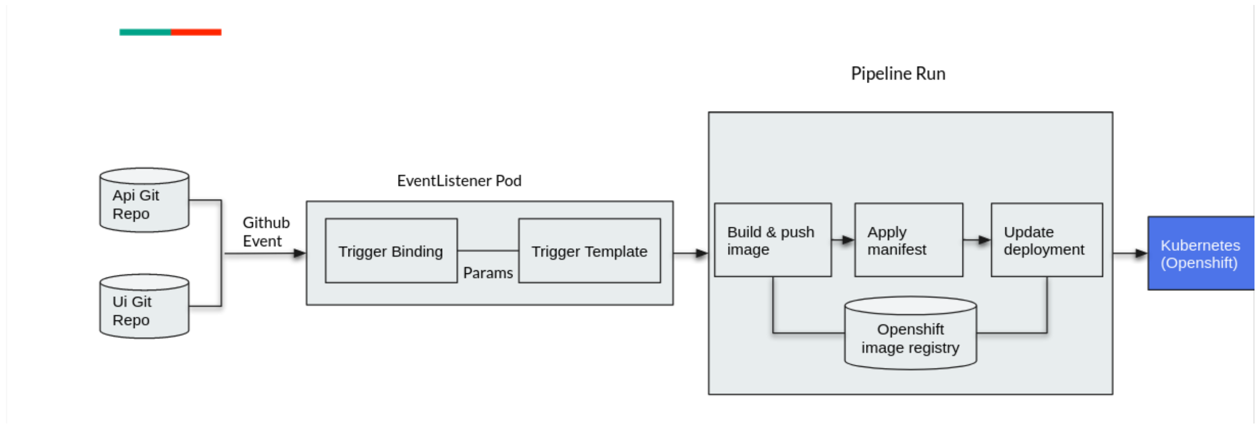
   You should see output similar to the following:

```
NAME                    DESCRIPTION   AGE
buildah                                1 day ago
buildah-v0-11-3                           1 day ago
jib-maven                               1 day ago
kn                                  1 day ago
maven                                   1 day ago
openshift-client                          1 day ago
openshift-client-v0-11-3                     1 day ago
s2i                                 1 day ago
s2i-dotnet-3                             1 day ago
…
```

Exercise 8: Creating a Pipeline

A pipeline defines a number of tasks that should be executed and how they interact with each other via their inputs and outputs.

In this exercise, you will create a pipeline that takes the source code of the application from GitHub and then builds and deploys it on OpenShift.



1. Create the pipeline by running the following:

   oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/01_pipeline/04_pipeline.yaml -n ${INITIALS}-pipelines-tutorial

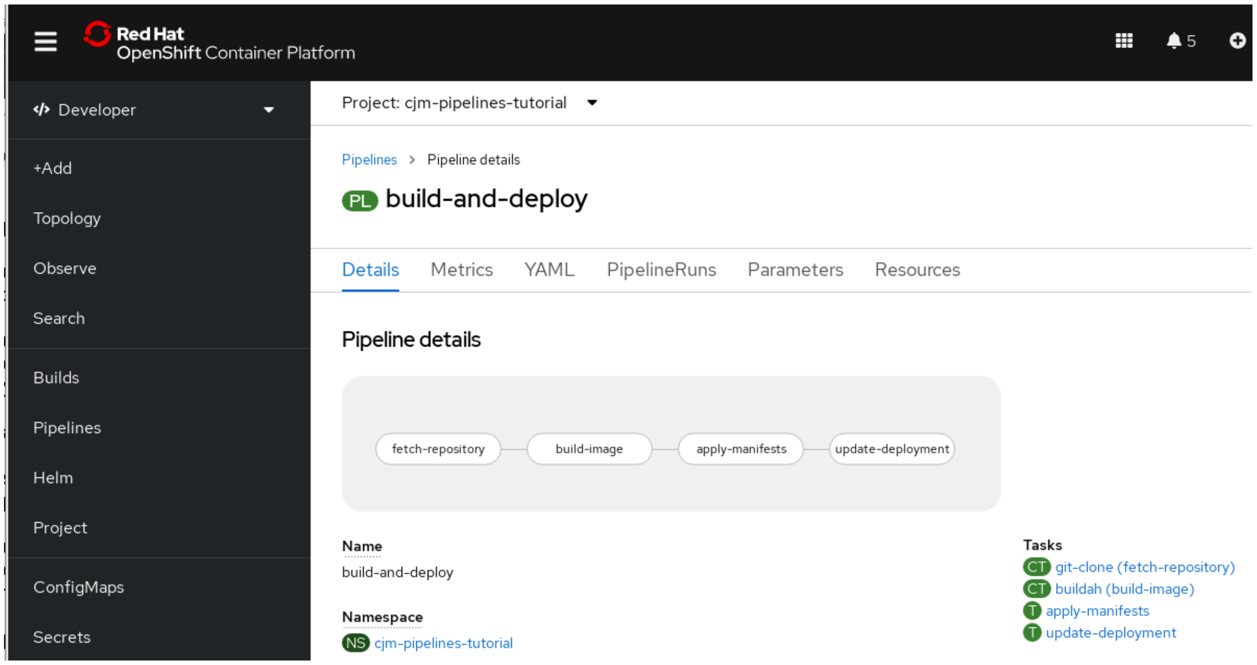
2. List your pipelines

   oc get pipelines

3. Review the pipeline yaml

   oc get pipeline build-and-deploy -o yaml

   You may alternately look at the YAML in the web console by navigating to the **Pipelines** menu in the **Developer** perspective:



Exercise 9: Triggering the Pipeline

Now that the pipeline is created, you can trigger it to execute the tasks specified in the pipeline.

1. A **PipelineRun** is how you can start a pipeline and tie it to the git and image resources that should be used for this specific invocation. You can start the pipeline using **tkn**:

```
tkn pipeline start build-and-deploy \
   -w
name=shared-workspace,volumeClaimTemplateFile=https://raw.githubusercontent
.com/openshift/pipelines-tutorial/master/01_pipeline/03_persistent_volume_claim.
yaml \
   -p deployment-name=pipelines-vote-api \
   -p git-url=https://github.com/openshift/pipelines-vote-api.git \
   -p
IMAGE=image-registry.openshift-image-registry.svc:5000/${INITIALS}-pipelines-tut
orial/pipelines-vote-api \
   --use-param-defaults
```

```
tkn pipeline start build-and-deploy \
   -w
name=shared-workspace,volumeClaimTemplateFile=https://raw.githubusercontent
.com/openshift/pipelines-tutorial/master/01_pipeline/03_persistent_volume_claim.
yaml \
   -p deployment-name=pipelines-vote-ui \
   -p git-url=https://github.com/openshift/pipelines-vote-ui.git \
   -p
IMAGE=image-registry.openshift-image-registry.svc:5000/${INITIALS}-pipelines-tut
orial/pipelines-vote-ui \
   --use-param-defaults
```

2. As soon as you start the build-and-deploy pipeline, a pipelinerun will be instantiated and pods will be created to execute the tasks that are defined in the pipeline.

```
tkn pipeline list
```

You should see output similar to the following:

```
NAME          AGE          LAST RUN               STARTED          DURATION
STATUS
```

```
build-and-deploy   6 minutes ago   build-and-deploy-run-xy7rw   36 seconds ago   ---
Running
```

3. Check out the logs of the **pipelinerun** as it runs using the tkn pipeline logs command which interactively allows you to pick the pipelinerun of your interest and inspect the logs:

   tkn pipeline logs -f

   You should see output similar to the following:

   ```
   ? Select pipelinerun:  [Use arrows to move, type to filter]
   > build-and-deploy-run-xy7rw started 36 seconds ago
     build-and-deploy-run-z2rz8 started 40 seconds ago
   ```

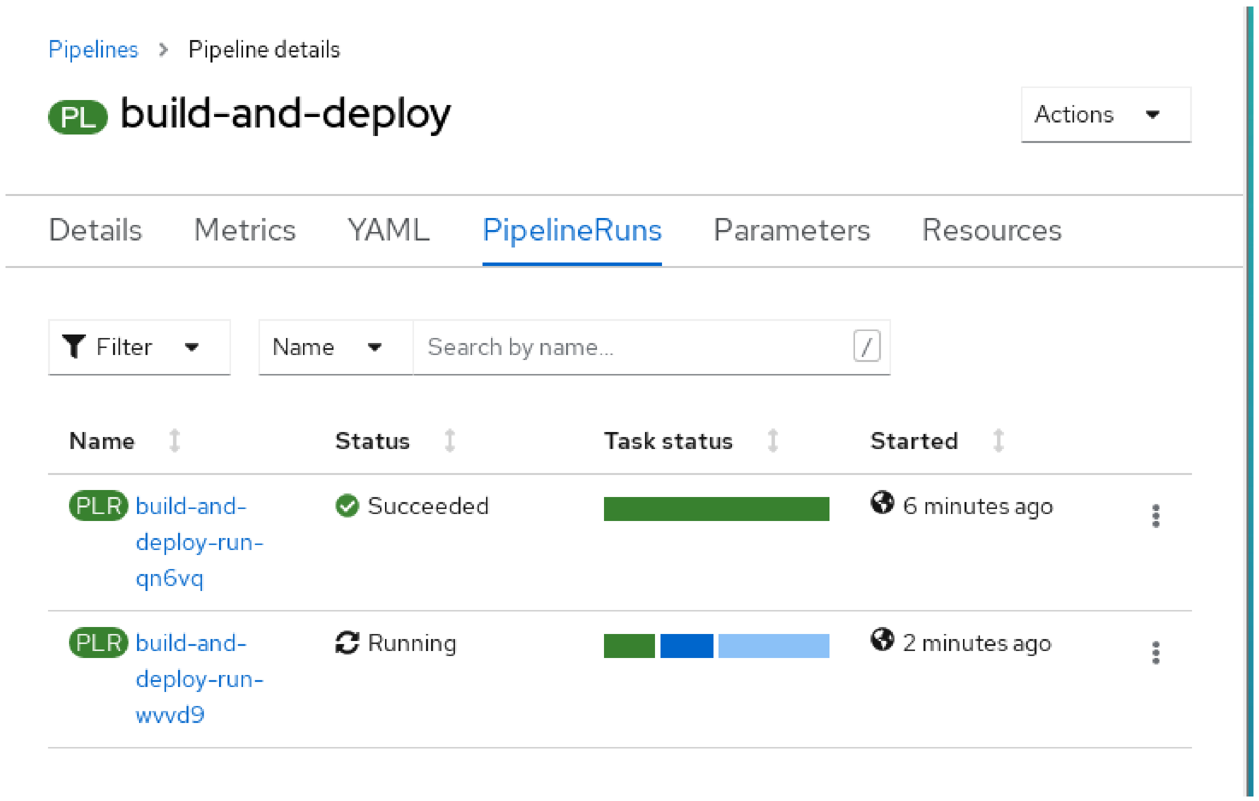
4. Confirm completion by running the following command.

   tkn pipelinerun list

   You should see output similar to the following:

   ```
   NAME                        STARTED      DURATION     STATUS
   build-and-deploy-run-xy7rw   1 hour ago   2 minutes    Succeeded
   build-and-deploy-run-z2rz8   1 hour ago   19 minutes   Succeeded
   ```

5. In the OpenShift web UI, you can see the pipeline runs by clicking on PipelineRuns under your **build-and-deploy** pipeline.


Pipelines > Pipeline details

PL build-and-deploy

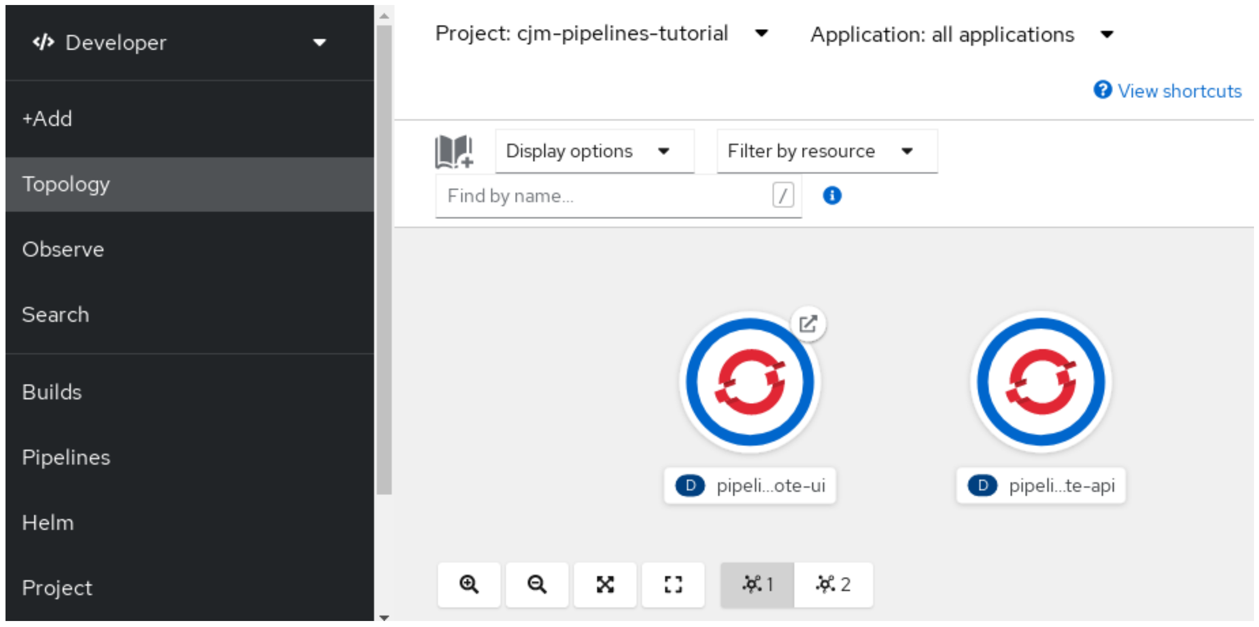
Actions

Details | Metrics | YAML | PipelineRuns | Parameters | Resources

Filter | Name | Search by name...

Name	Status	Task status	Started
PLR build-and-deploy-run-qn6vq	Succeeded	🟩	6 minutes ago
PLR build-and-deploy-run-wvvd9	Running	🟩🟦	2 minutes ago

6. Looking back at the project, you should see that the images are successfully built and deployed. Navigate to the **Topology** page in the **Developer** perspective for your project.



7. You can get the route of the application by executing the following command and access the application (or use the Web UI).

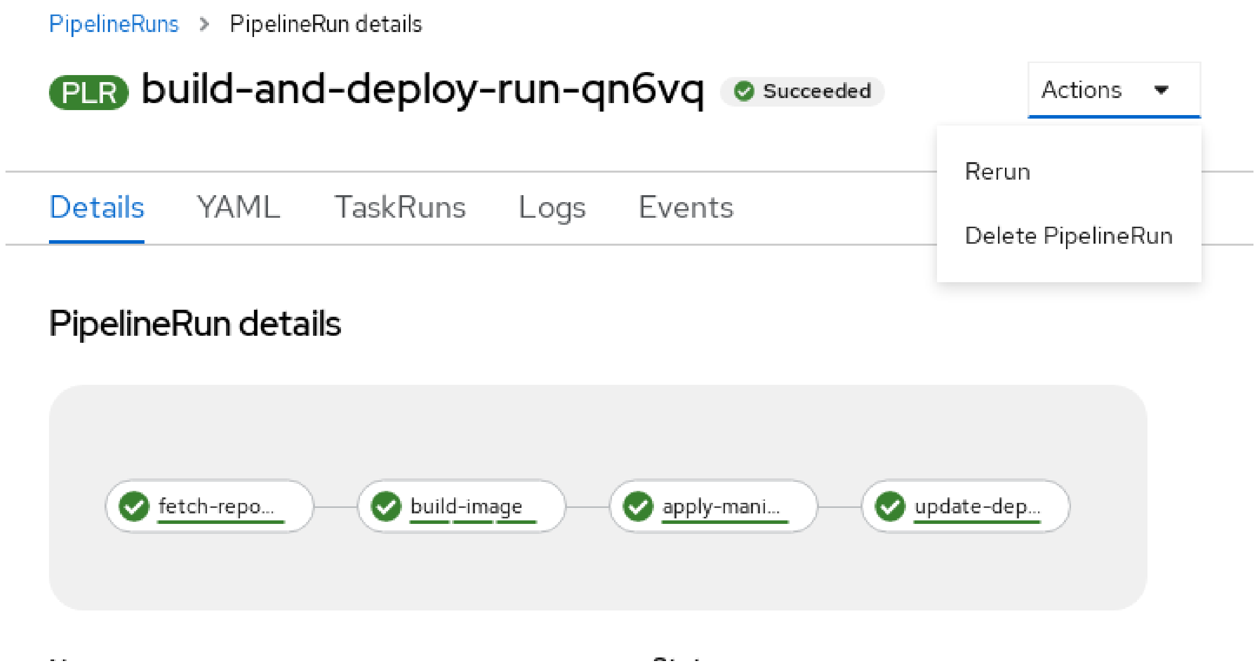
oc get route pipelines-vote-ui --template='http://{{.spec.host}}'

8. If you want to re-run the pipeline again, you can use the following short-hand command to rerun the last pipelinerun again that uses the same pipeline resources and service account used in the previous pipeline run:

   tkn pipeline start build-and-deploy --last

   Note: Whenever there is any change to your repository we need to start pipeline explicitly to see new changes to take effect

   You can also re-run a PipelineRun by selecting **Rerun** from the **Actions** menu on the PipelineRun details page.



Exercise 10: Configuring Triggers

Triggers, in conjunction with pipelines, enable us to configure our Pipelines to respond to external github events (push events, pull requests etc).

In this exercise, we will add a TriggerTemplate, TriggerBinding, and an EventListener to our project.

Prepare Resource Files

These files need to be cloned locally and one updated to match the actual project name in use.

```
git clone https://github.com/openshift/pipelines-tutorial
sed -i "s/5000\/pipe/5000\/${INITIALS}-pipe/" \
    pipelines-tutorial/03_triggers/02_template.yaml
```

Trigger Templates

A TriggerTemplate is a resource which has parameters that can be substituted anywhere within the resources of the template.

Run the following command to apply TriggerTemplate.

```
oc create -f pipelines-tutorial/03_triggers/02_template.yaml
```

Trigger Bindings

TriggerBindings are maps that enable you to capture fields from an event and store them as parameters, and replace them in TriggerTemplates whenever an event occurs.

Run the following command to apply the TriggerBinding.

```
oc create -f pipelines-tutorial/03_triggers/01_binding.yaml
```

Triggers

Triggers combine TriggerTemplate, TriggerBindings and interceptors. They are used as ref inside the EventListener.

Run the following command to apply the Trigger.

```
oc create -f pipelines-tutorial/03_triggers/03_trigger.yaml
```

Event Listener

This component sets up a Service and listens for events. It also connects a TriggerTemplate to a TriggerBinding, into an addressable endpoint (the event sink)

Run the following command to create an EventListener.

```
oc create -f pipelines-tutorial/03_triggers/04_event_listener.yaml
```

Note: Creating the EventListener will set up a Service. We need to expose that Service as an OpenShift Route to make it publicly accessible.

Run below command to expose the eventlistener service as a route

```
oc expose svc el-vote-app
```

Exercise 11: Configuring GitHub WebHooks

Now we need to configure the webhook-url for backend and frontend source code repositories with the Route we exposed in the previous exercise.

1. Run below command to get webhook-url:

   ```
   echo "URL: $(oc  get route el-vote-app --template='http://{{.spec.host}}')"
   ```

2. Fork the backend and frontend source code repositories so that you have sufficient privileges to configure GitHub webhooks. To fork the repositories, click the **fork** button at the top of each repository.

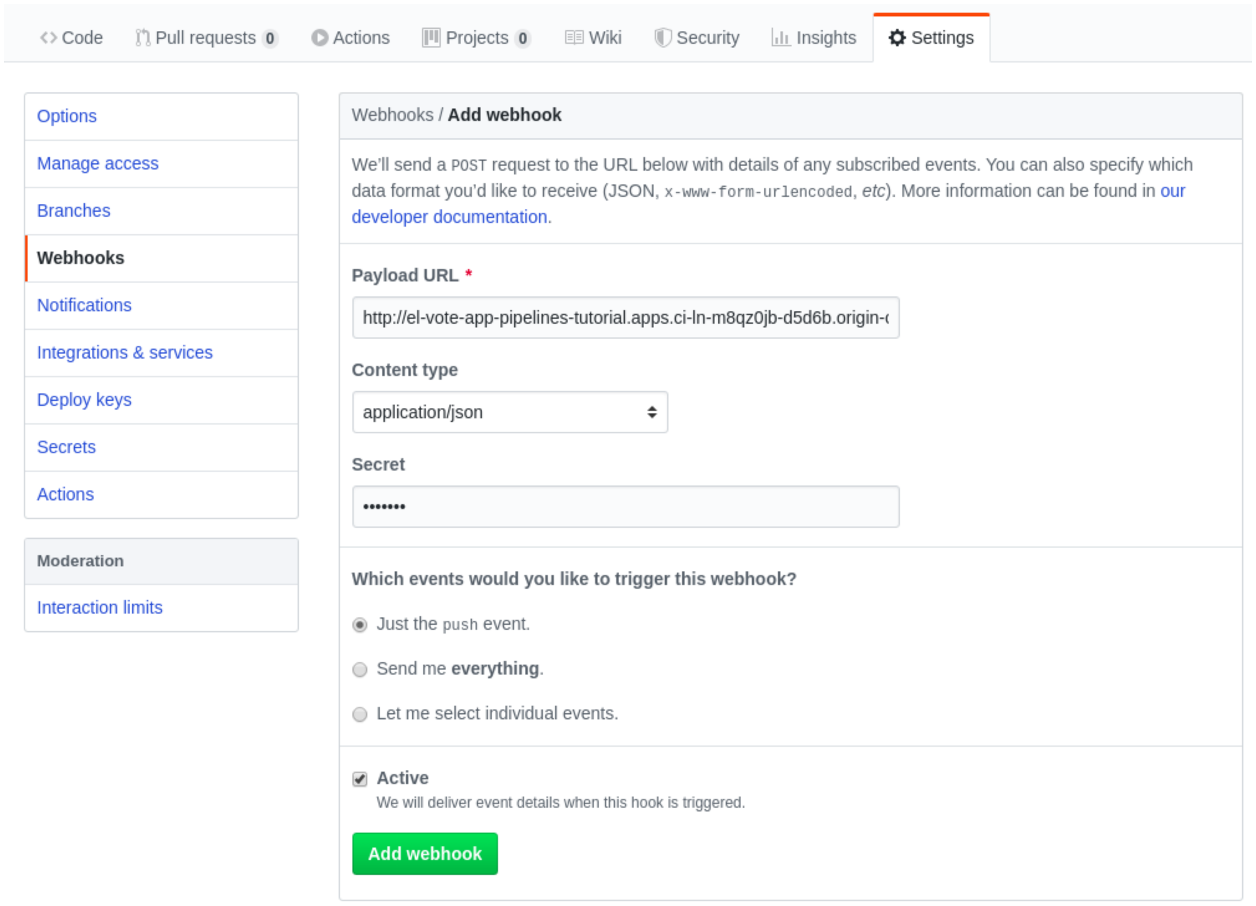
   Repos:
   http://github.com/openshift/pipelines-vote-api.git
   http://github.com/openshift/pipelines-vote-ui.git

3. Open the forked github repo (Go to **Settings** > **Webhooks**) click on **Add webhook**

4. Add the results of the following command to payload URL:

   ```
   echo "$(oc  get route el-vote-app --template='http://{{.spec.host}}')"
   ```

5. Select Content type as **application/json** > Add secret eg: **1234567** > Click on Add Webhook



Webhooks / **Add webhook**

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *

http://el-vote-app-pipelines-tutorial.apps.ci-ln-m8qz0jb-d5d6b.origin-

Content type

application/json

Secret

•••••••

Which events would you like to trigger this webhook?

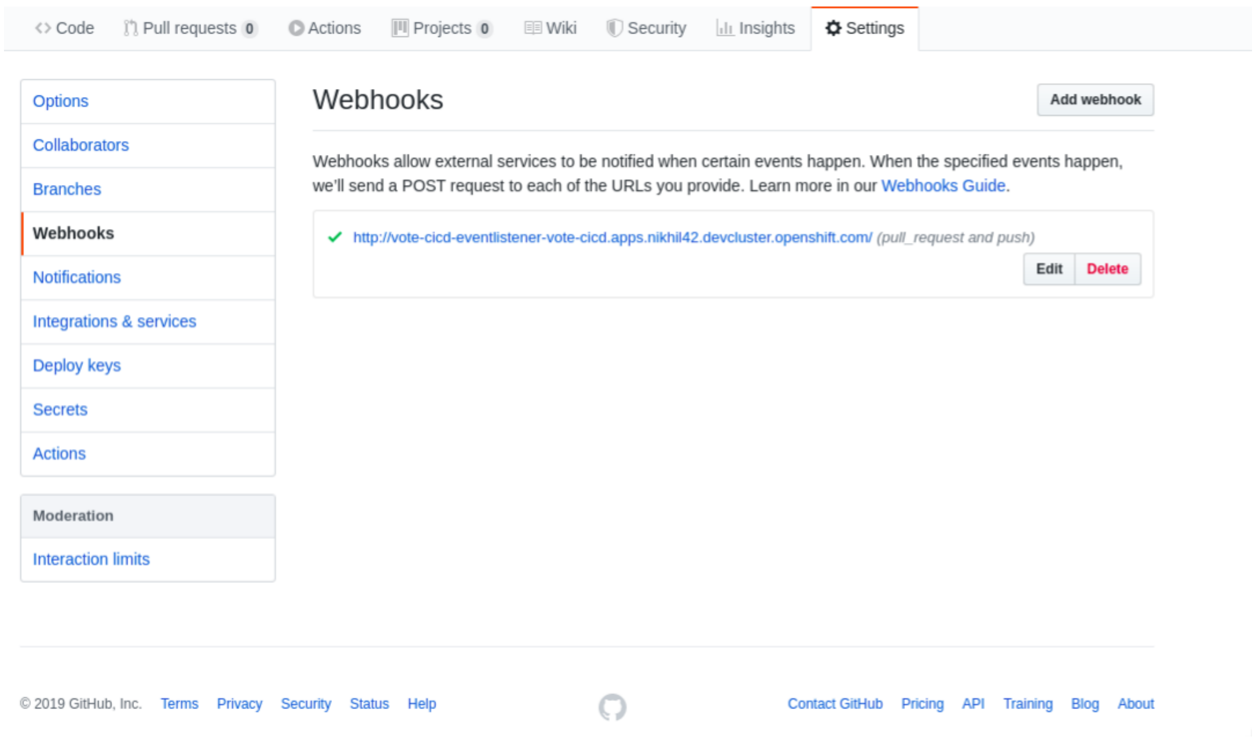
[✓ Just the `push` event.]

Send me **everything**.

Let me select individual events.

[✓ **Active**]
We will deliver event details when this hook is triggered.

Perform these steps for both **pipelines-vote-ui** and **pipelines-vote-api**. Now we should see a webhook configured on your forked source code repositories (on your GitHub repositories, go to Settings>Webhooks).



Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

✓ http://vote-cicd-eventlistener-vote-cicd.apps.nikhil42.devcluster.openshift.com/ *(pull_request and push)*

Trigger pipeline Run

When we perform any push event on the **pipelines-vote-api** repository the following should happen.

A. The configured webhook in the pipelines-vote-api GitHub repository should push the event payload to our route (exposed EventListener Service).

B. The Event-Listener will pass the event to the TriggerBinding and TriggerTemplate pair.

C. The TriggerBinding will extract parameters needed for rendering the TriggerTemplate. Successful rendering of TriggerTemplate should create two PipelineResources (source-repo-vote-api and image-source-vote-api) and a PipelineRun (build-deploy-vote-api)

We can test this by pushing a commit to the **pipelines-vote-api** repository from GitHub web ui or from terminal.

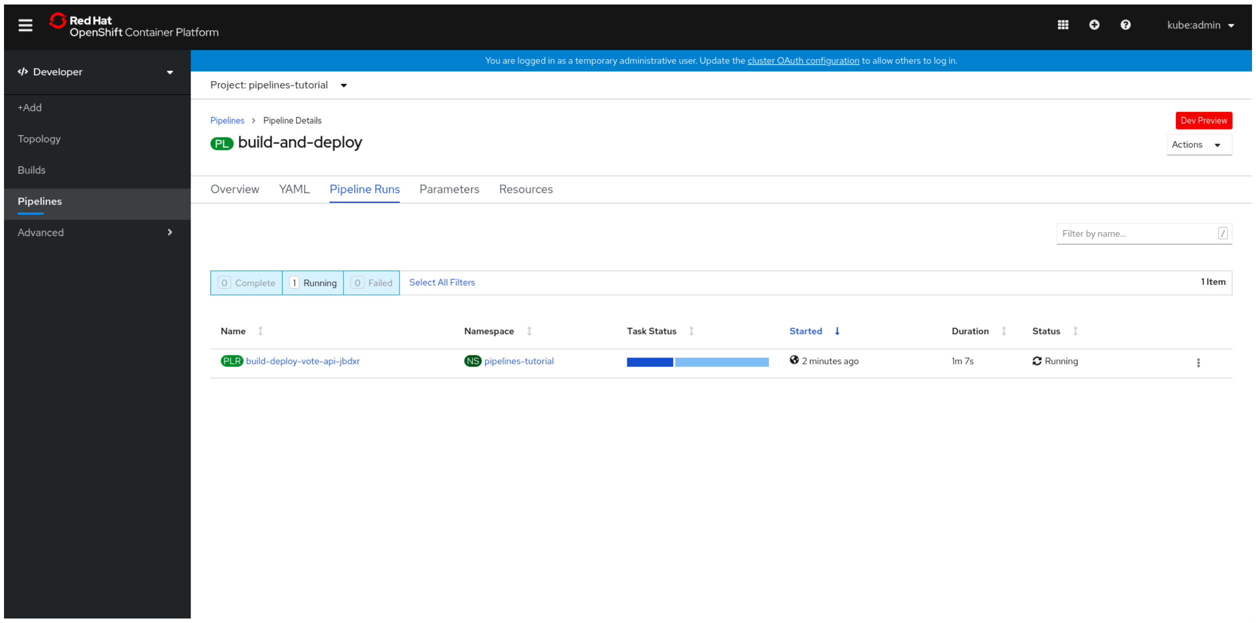
D. Push an empty commit to vote-api repository.

   Option A
   Note: Clone one of the forked repos, then cd into the repo directory before running the command below.

   git clone <forked repository URL>
   cd <repository directory>
   git commit -m "empty-commit" --allow-empty && git push origin master

   Option B
   Use the GitHub web ui to edit the README file at the top of the repo, then commit the change to the master branch. Scroll down the page to find the commit button.

E. Watch OpenShift WebConsole Developer perspective and a PipelineRun will be automatically created.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Name	Namespace	Task Status	Started	Duration	Status
PLR build-deploy-vote-api-jbdxr	NS pipelines-tutorial		2 minutes ago	1m 7s	Running

Exercise 12: Deploying a legacy application which needs updating to Quarkus

Create a New Project for this Application

F. From the bastion host command line, run the following command to create a new project.

```
oc new-project ${INITIALS}-employees
```

Deploy MySQL

G. Run the following command to deploy the MySQL template into your project.

```
oc process openshift//mysql-persistent -p MYSQL_USER=sdgdemo
MYSQL_PASSWORD=sdgdemo MYSQL_ROOT_PASSWORD=root
VOLUME_CAPACITY=1G MYSQL_DATABASE=emp | oc create -f -
```

H. Run the following commands to confirm the database pod is running.

```
oc get pods
```

Deploy the Application

I. Run the following command to deploy the Java application code.

```
oc new-app
registry.access.redhat.com/ubi8/openjdk-8~https://github.com/meidlinger/sdgdem
ospringboot.git --name=springboot-demo
```

J. Review the build logs

```
oc logs -f bc/springboot-demo
```

K. Expose the service for external access

```
oc expose svc/springboot-demo
```

L. Verify that the application is working. Go to the **Routes** section of the Web UI to determine the URL for your application. When you connect to that URL with a browser, you should see “Greetings from Spring Boot!”

M. Use the following command to add a person to the database.

   ```
   curl
   http://springboot-demo-${INITIALS}-employees.mycluster-us-south-764377-85853ed96992ad4cfd8a12f081018409-0000.us-south.containers.appdomain.cloud/v1/api/sdg/demo/person -d name=Kenny -d title="Consultant"
   ```

N. Append the following path to the URL in your browser to confirm the data entry.

   /v1/api/sdg/demo/person/title?name=Kenny

   You should get “Consultant” in response.

Exercise 13: Deploying the Quarkus application

Deploy the application

1. Run the following command to deploy the Quarkus version of the employees application.

   ```
   oc new-app
   registry.access.redhat.com/ubi8/openjdk-11~https://github.com/meidlinger/sdgdemoquarkus.git#s2i-build --name=quarkus-demo
   ```

2. Expose the service.

   ```
   oc expose svc quarkus-demo
   ```

Test by adding data

3. Use the following command to add a person to the database.

   ```
   curl -X POST
   http://quarkus-demo-cjm-qtest2.mycluster-us-south-764377-85853ed96992ad4cfd8a12f081018409-0000.us-south.containers.appdomain.cloud/v1/api/sdg/demo/person/name/Eddie/title/Guitarist
   ```

   The result should be similar to:

```
{"id":2,"employeename":"Eddie","employeetitle":"Guitarist"}
```

4. Run the following command to list all employees.

   ```
   curl
   http://quarkus-demo-cjm-qtest2.mycluster-us-south-764377-85853ed96992ad4cfd8a12f081018409-0000.us-south.containers.appdomain.cloud/v1/api/sdg/demo/person/all
   ```

   The result should be similar to:

   ```
   [{"id":1,"employeename":"Kenny","employeetitle":"Consultant"},{"id":2,"employeename":"Eddie","employeetitle":"Guitarist"}]
   ```

5. To review the required modifications which converted this application from springboot to quarkus, clone the repositories and use your favorite tools to compare the differences.

END.