Course Exercises

IBM Cloud Garage - Cloud Native Developer Bootcamp for IBM Cloud Private

Course code CK105 ERC 1.0

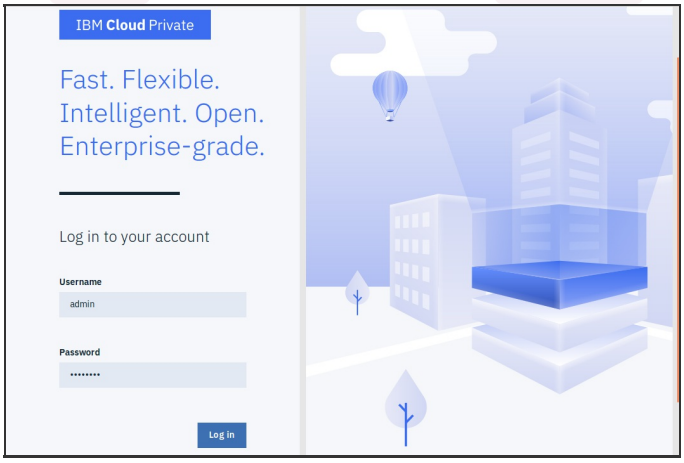
IBM Training

Initial Setup of IBM Cloud Private

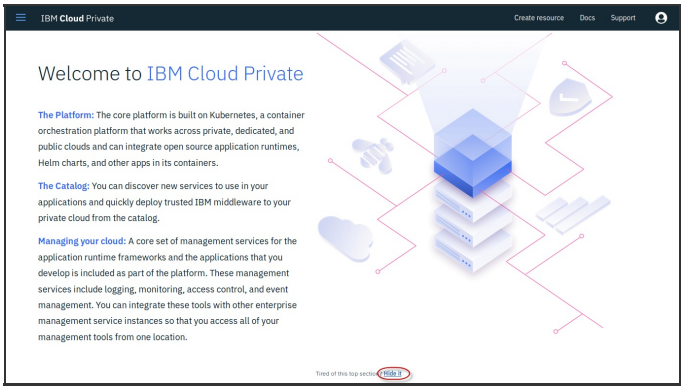
Exercise 1: The Web User Interface

This exercise familiarizes you with the IBM Cloud Private Web User interface.

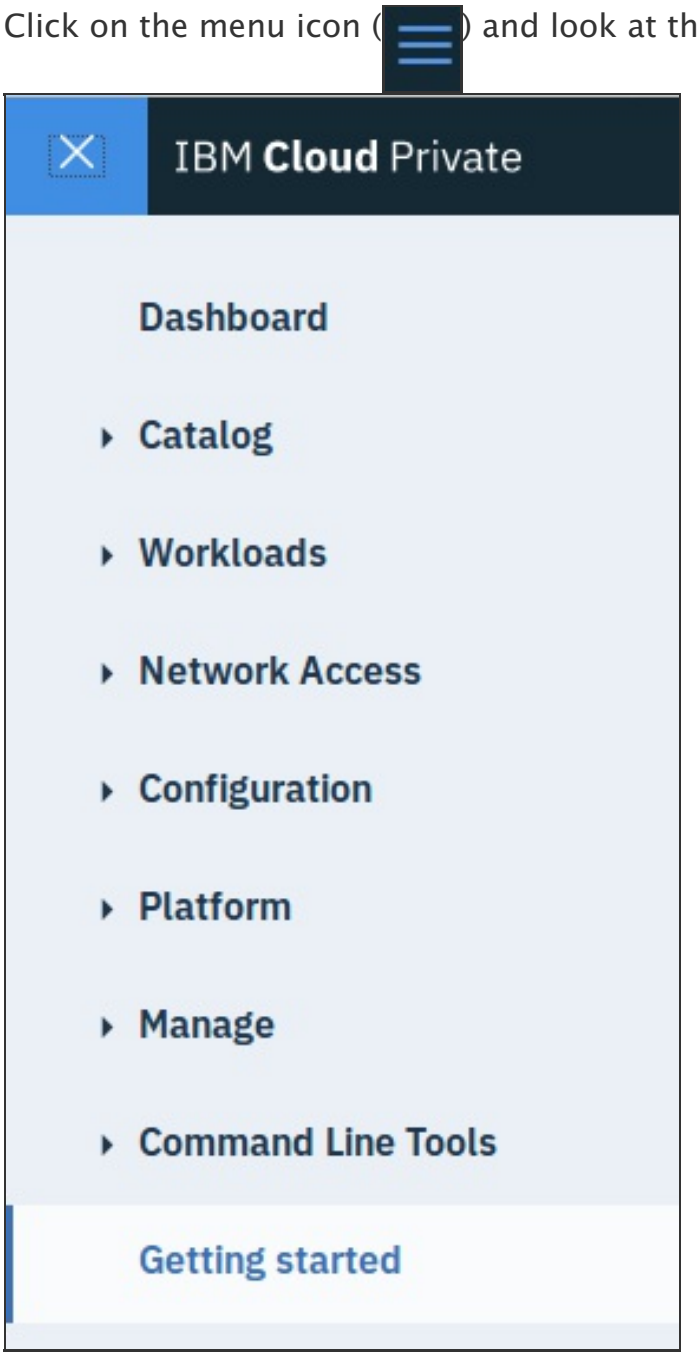
1. Open a Firefox Web browser and go to `https://10.10.1.10:8443`. Login as `admin` with the password of `passw0rd`.



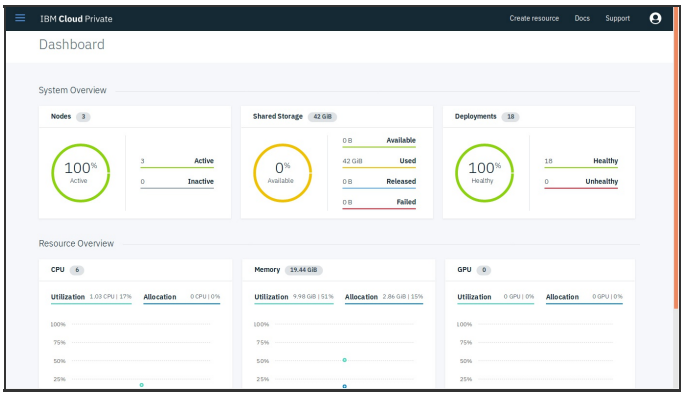
2. Scroll down and hide the Welcome message.



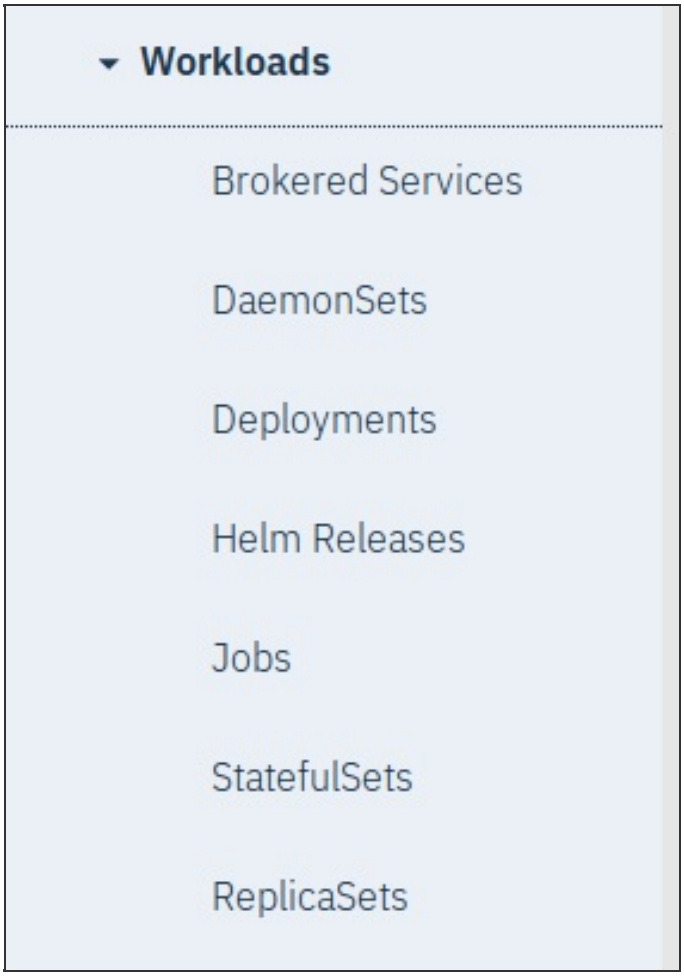
3. Click on the menu icon ( ) and look at the various components.



4. In the dashboard page, you can see the snapshot of the current status of your cluster.



5. Expand the **Workloads** menu. These are the various available workloads that can run in IBM Cloud Private.

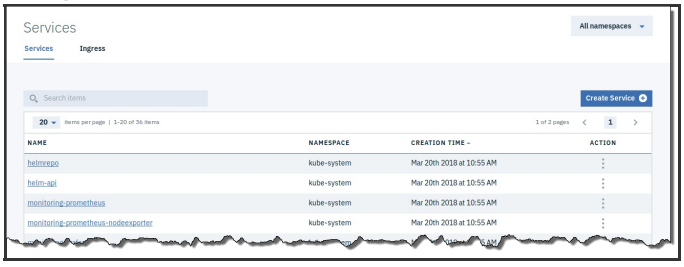


6. Currently there are only system workloads (namespace of `kube-system`). As an example, open the deployments workload (**Workloads** >

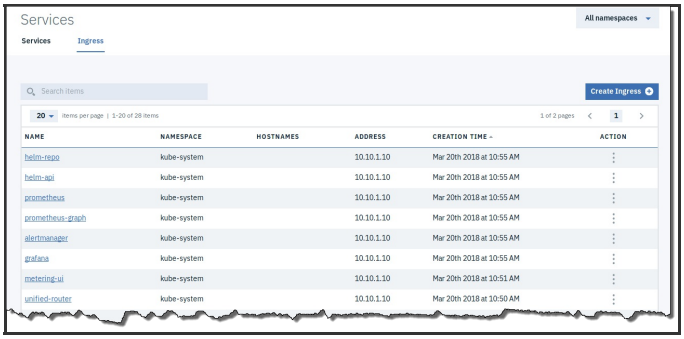
Deployments) to see the system applications.

7. Other resources already defined are **Jobs**, **DaemonSet** and **StatefulSet**.

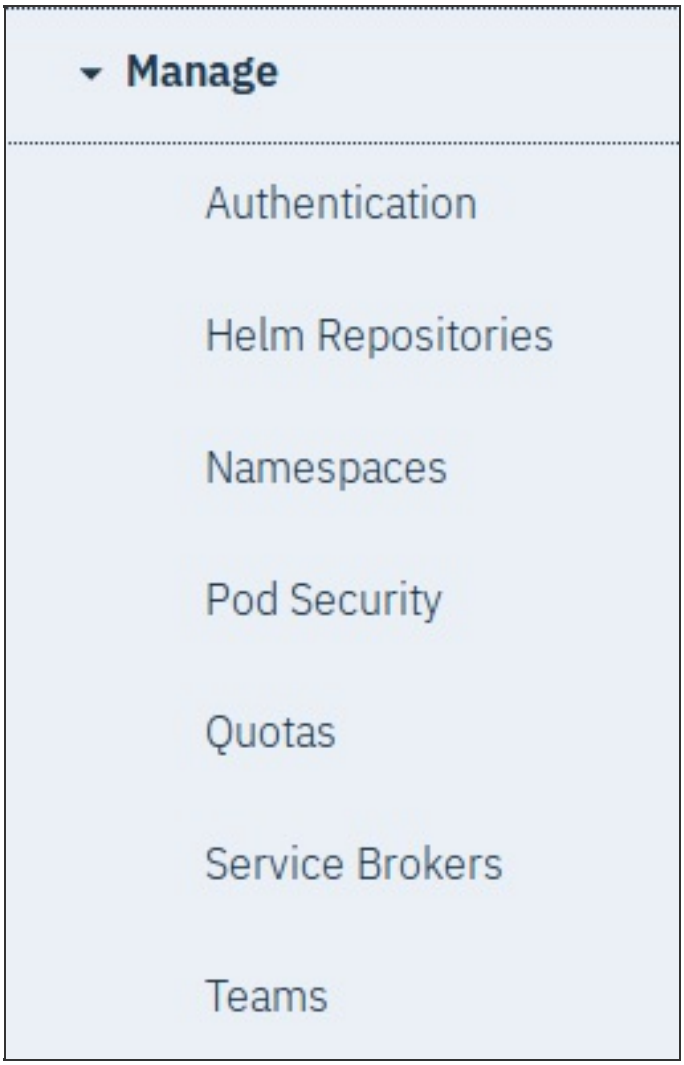
8. Under **Network Access** > **Services**, you can see the list of network endpoints for the workloads.



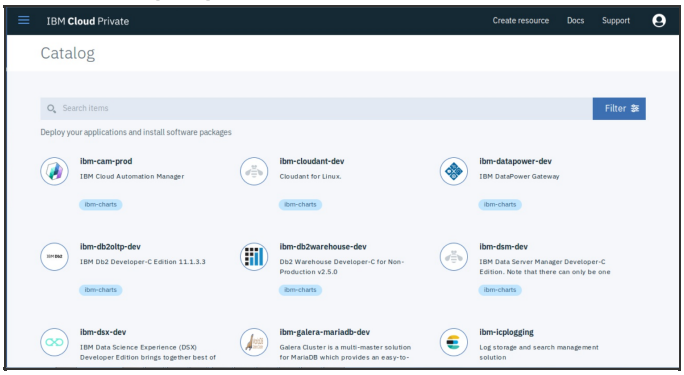
9. Still in the services view, click on the **Ingress** tab. This tab lists the various external access endpoints provided for exposing access to the cluster.



10. Under the **Manage** menu, you can see various cluster management related resources.



11. Click on **Catalog** > **Helm Charts**. Listed here are the applications that can be deployed into IBM Cloud Private.



12. You can continue to explore various options in the Web UI.

Exercise 2: Set up CLI tools

This exercise familiarizes you with the CLI environment for IBM Cloud Private:

1. Open a terminal window.
2. Get helm from `https://master:8443/helm-api/cli/linux-amd64/helm`:
    - Use `wget` to download helm from the IBM Cloud Private master node (command is all on one line).

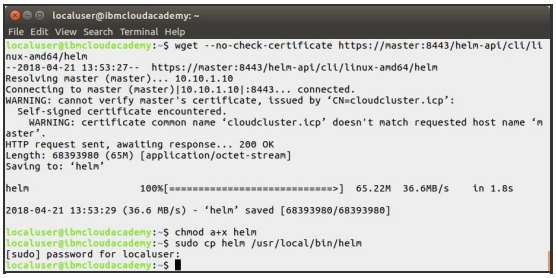
      ```
      wget --no-check-certificate https://master:8443/helm-api/cli/linux-amd64/helm
      ```

    - Set helm to be executable.

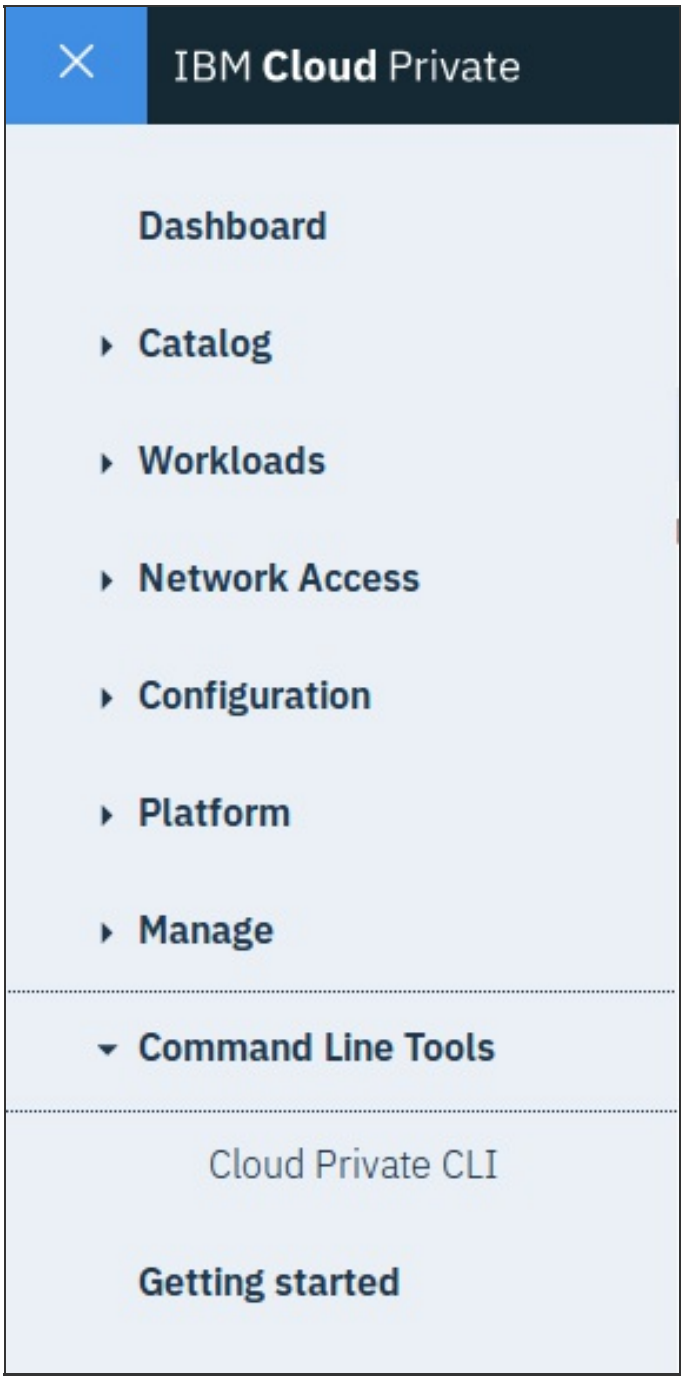
      ```
      chmod a+x helm
      ```

    - Copy helm to /usr/local/bin (the sudo password is `passw0rd`).

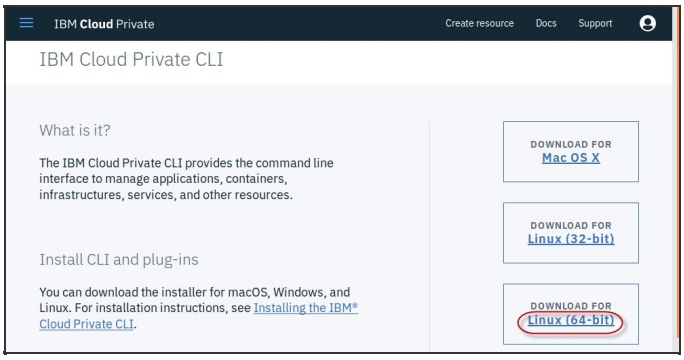
      ```
      sudo cp helm /usr/local/bin/helm
      ```

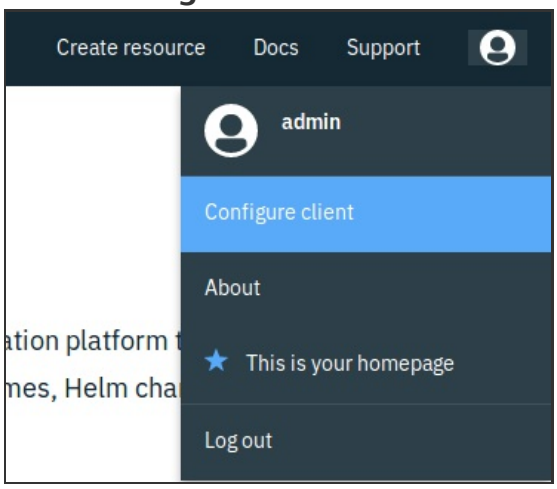
3. In your browser, get the IBM Cloud Private CLI plugin from the menu icon and select **Command Line Tools** > **Cloud Private CLI**.



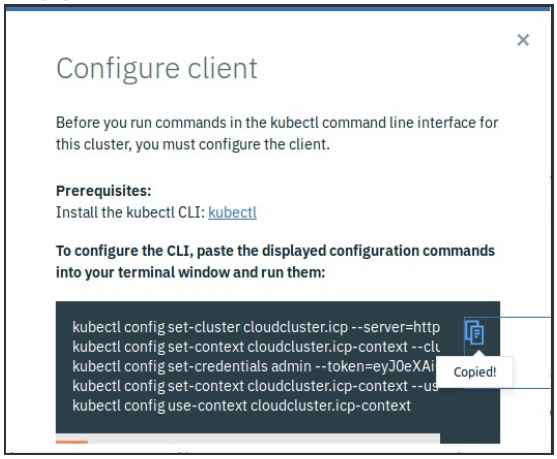
4. Click the Download link for **Linux (64-bit)** and click **Save file**. The file is downloaded to the `Downloads` folder.



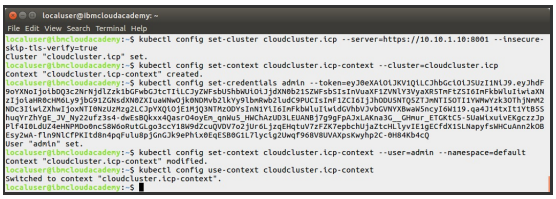
5. Set up the `kubectl` configuration. Click the "head" icon in the top-right corner of your browser window.
    - Click **Configure Client**.



    - In the configure client pop-up, click the Copy icon to copy the CLI commands.



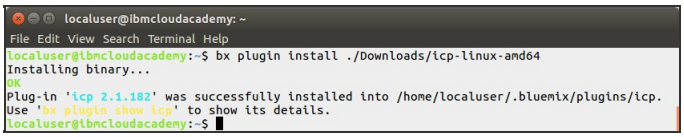
- Paste the commands into the terminal window to initialize kubectl.



6. Now you can invoke `kubectl`. Try to run `kubectl get pod -n kube-system` to list all pods related to the Kubernetes system.

7. Install the IBM Cloud CLI plugin.

   ```
   bx plugin install ./Downloads/icp-linux-amd64
   ```



8. Initialize helm using the IBM Cloud Private CLI.

   - Initialize the helm client.

     ```
     helm init --client-only
     ```

   - Login to the IBM Cloud Private CLI. Select the `cloudcluster` account.

     ```
     bx pr login -a https://master:8443 -u admin -p passw0rd
     --skip-ssl-validation
     ```

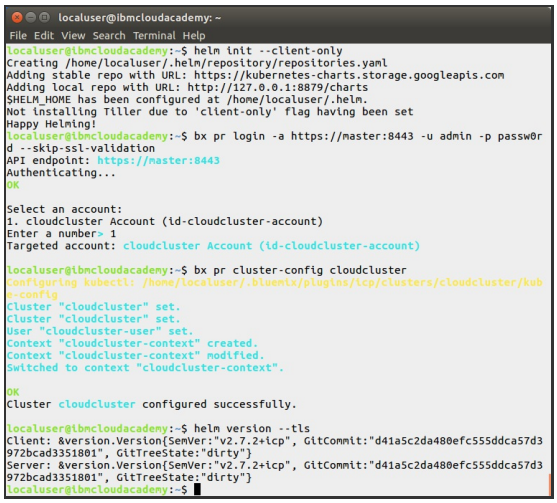
   - Set up helm credentials.

     ```
     bx pr cluster-config cloudcluster
     ```

     Note: compare the output here to the kubectl config commands you copied and pasted earlier to configure access to IBM Cloud Private.

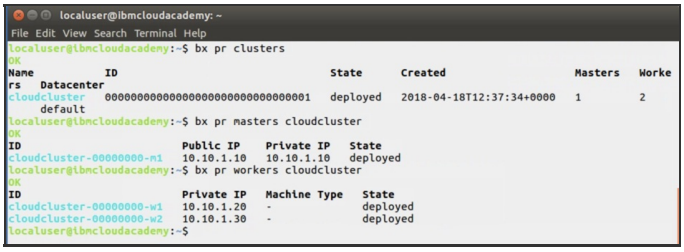
   - Check helm to tiller connectivity.

     ```
     helm version --tls
     ```



9. In your terminal window, use the following commands to view some details about your cluster.

```
bx pr clusters
bx pr masters cloudcluster
bx pr workers cloudcluster
```



*** End of exercises ***

Part 1 - Docker introduction lab

Prerequisites

This set of instructions requires that docker is already installed and docker commands can be run from a bash shell. You can get more information at the Docker website

Note: This demo assumes that you are running this from a "clean" environment. Clean means that you have not used docker with the images in this demo. This is important for someone who hasn't seen docker so they can see the activity as images are downloaded.

Working with docker

1. Launch a shell and confirm that docker is installed. The version number isn't particularly important.

```
$ docker -v
Docker version 17.06.1-ce, build 874a737
```

2. As with all new computer things, it is obligatory that we start with "hello-world"

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b04784fba78d: Pull complete
Digest:
sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26f
f74f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the
Docker Hub.
 3. The Docker daemon created a new container from that image which
runs the
```

```
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client,
which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container
with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide
```

Notice the message `Unable to find image 'hello-world:latest' locally` First you see that the image was automatically downloaded without any additional commands. Second the version `:latest` was added to the name of the image. We did not specify a version for this image.

3. Rerun "hello-world" Notice that the image is not pulled down again. It already exists locally, so it is run.

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working
correctly.

      [output truncated]
```

4. It already exists locally and `docker images` will show us that image.

```
$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
hello-world         latest    1815c82652c0   2 months ago    1.84kB
```

5. From where was the `hello-world` image pulled? Go to `https://hub.docker.com/_/hello-world/` and you can read about this image. Docker-hub is a repository that holds docker images for use. Docker-hub is not the only repository, IBM Cloud can serve as a docker repository.

6. This image is atypical; when an image is run it usually continues to run.

The running image is called a container. Let us run a more typical image; this image contains the noSQL database "couchDB".

```
$ docker run -d couchdb
Unable to find image 'couchdb:latest' locally
latest: Pulling from library/couchdb
ad74af05f5a2: Downloading [==>                        ]
2.702MB/52.61MB
ffdd0c835430: Download complete
d922980c187f: Downloading [===>                       ]
2.661MB/43.77MB
affbf57fdbcf: Verifying Checksum
0ddcd7e9244b: Download complete
34473f480310: Downloading [=============>           ]
2.26MB/8.236MB
78a52d457cb5: Waiting
```

The output above was captured while the image was still downloading from docker-hub. When the download is down you don't see anything from the container, like with hello-world. Instead you see a long hex id like `2169c6b42e5c590229c5c86f5ed3596b1b56c2366378914b082e5b000752bd34`. This is the id of the container.

6. Here's how you would see the running container. Notice only the first part of that long hex id is displayed. Typically this is more than enough to uniquely identify that container. `docker ps` provides information about when the container was created, how long it has been running, then name of the image as well as the name of the container. Note that each container must have a unique name. You can specify a name for each container as long as it is unique.

```
$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS        PORTS     NAMES
2169c6b42e5c  couchdb   "tini -- /docker-e..."   8 minutes ago   Up
8 minutes  5984/tcp  nervous_poincare
```

7. An image can be run multiple times. Launch another container for the couchdb image.

```
$ docker run -d couchdb
f9885aaf0a96742119462208dce611018ab2104737adf3485d6fc4e7642b104b
```

8. Now we have two containers running the couchdb database. Did you notice how quickly the second instance started? There was no need to download the image this time. The id of the container is show after is has started.

```
$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED         STATUS        PORTS
NAMES
f9885aaf0a96  couchdb   "tini -- /docker-e..."   2 minutes ago   Up 2 minutes  5984/tcp
brave_booth
2169c6b42e5c  couchdb   "tini -- /docker-e..."   22 minutes ago  Up 22 minutes 5984/tcp
nervous_poincare
```

9. The containers look similar, but they have unique names and unique ids. Stop the most recent container and then check to see what's running.

```
$ docker stop f9885aaf0a96
f9885aaf0a96

$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS        PORTS     NAMES
2169c6b42e5c  couchdb   "tini -- /docker-e..."   25 minutes ago  Up
25 minutes 5984/tcp  nervous_poincare
```

10. Stop the other container and see what is running.

```
$ docker stop 2169c6b42e5c
2169c6b42e5c

$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS        PORTS     NAMES
```

11. Notice the image still exists.

```
$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
couchdb           latest    7f8923b03b7f   5 weeks ago     225MB
hello-world       latest    1815c82652c0   2 months ago   1.84kB
```

11. Did you forget about the hello-world image? Go ahead and delete the couchdb image and double check that it is gone.

```
docker rmi couchdb
Error response from daemon: conflict: unable to remove repository reference "couchdb"
(must force) - container 2169c6b42e5c is using its referenced image 7f8923b03b7f
```

12. Oops, we can't delete that image until we delete the "couchdb" container. Note the `docker ps -a` will show us all the containers, not just the ones that are running.

```
$ docker ps -a
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS                     NAMES
f9885aaf0a96  couchdb     "tini -- /docker-e..." 22 minutes ago
Exited 2 minutes ago     brave_booth
9cd02a3f2eec2 couchdb     "tini -- /docker-e..." 22 minutes ago
Exited 4 minutes ago     silly_ritchie
b44146cfad65  hello-world "/hello"               An hour ago
Exited About an hour ago elated_engelbart
8d71894c865b  hello-world "/hello"               2 hours ago
Exited 2 hours ago       stoic_lamport
```

13. Delete the couchdb containers, delete the couchdb image, and make sure it is gone. You can leave hello-world.

```
$ docker rm f9885aaf0a96
f9885aaf0a96

$ docker rmi couchdb
Untagged: couchdb:latest
Untagged:
couchdb@sha256:eb463cca23b9e9370afbd84ae1d21c0274292aabd11b2e5b904d
4be2899141ff
Deleted:
sha256:7f8923b03b7f807ffbd51ff902db3b5d2e2bbbc440d72bc81969c6b05631
7c8a
Deleted:
sha256:d53bc50464e197cbe1358f44ab6d926d4df2b6b3742d64640a2523e46401
04c4
Deleted:
sha256:851748835e9443fa6b8d84fbfada336dffd1ba851a7ed51a0152de3e3115
b693
Deleted:
sha256:feb87fb4c017e2d01b5d22dee3f23db2b3f06a0111a941dda926139edc02
```

```
7c8e
Deleted:
sha256:e00c9e10766f6a4d24eff37b5a1000a7b41c501f4551a4790348b94ff179
ca53
Deleted:
sha256:b64ffebe7ca9cec184d6224d4546ccdfecce6ddf7f5429f8e82693a8372c
f599
Deleted:
sha256:f78934f92a8a0c822d4fc9e16a6785dc815486e060f60b876d2c4df19255
84d8
Deleted:
sha256:491c6d0078fa4421d05690c79ffa4baf3cdeb5ead60c151ab64af4fb6d4d
93dc
Deleted:
sha256:2c40c66f7667aefbb18f7070cf52fae7abbe9b66e49b4e1fd740544e7cea
ebdc
```

```
$ docker ps -a
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS                     NAMES
b44146cfad65  hello-world "/hello"               An hour ago
Exited About an hour ago elated_engelbart
8d71894c865b  hello-world "/hello"               2 hours ago
Exited 2 hours ago       stoic_lamport

$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
hello-world       latest    1815c82652c0   2 months ago   1.84kB
```

Note: Docker images and containers can be referenced by name or by id. This demonstration has shown only the rudimentary capabilities of docker.

Part 2 - WebSphere Liberty Docker lab

WebSphere Liberty running in docker

1. Launch a shell and confirm that docker is installed. The version number isn't particularly important.

```
$ docker -v
Docker version 17.06.1-ce, build 874a737
```

2. Pull the latest docker image for WebSphere Liberty. This may take a few

minutes to complete. You will notice that docker will download all of the layers of images that the Liberty image is built upon.

```
$ docker pull websphere-liberty
Using default tag: latest
latest: Pulling from library/websphere-liberty
  [output removed]

Digest:
sha256:60c00ecb8b3f74f7a2a4533bad57e9684a2df8f957919f332909bd55b248
4817
Status: Downloaded newer image for websphere-liberty:latest
```

3. When the image download has completed, you can verify that it has been downloaded by using the command `docker images`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
websphere-liberty   latest              01128080ee00        3 weeks
ago         437MB
```

4. You can also verify that no other images (containers) are running with the `docker ps` command.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
```

5. One thing you should look at in this demo is how little memory and CPU are required to run docker images. In this example you'll use uptime to look at system load to get a general idea how hard the CPU is working. Take a look at the load averages below. It isn't really important what you understand exactly what these numbers mean. This article explains what the load averages mean. From left to right the load average numbers are for 1, 5, and 15 minutes. What you will be checking is what the load averages look like before and after running multiple docker containers. You should write down the load averages, because one of the steps will clear your screen. Notice in the output here, that the one minute average load is slightly above three, and the five minute and fifteen minutes averages are a little less than three.

```
$ uptime
11:33  up 6 days, 20:24, 3 users, load averages: 3.28 2.91 2.83
```

6. In this step you will start up multiple instances of the WebSphere Liberty docker image. This command relies on having a bash shell to run. You will be launching five separate instances of the container. The liberty server will listen on port 9080. The `-p $i:9080` maps this port to another port. The first image will listen on port 80, the second on port 81, and so on. You can change these ports if they have a conflict on your machine. As each container is started, the unique ID of that container is returned. Notice that the container start rather quickly, another good feature of docker containers.

```
$ for i in 80 81 82 83 84; do docker run -d -p $i:9080  websphere-
liberty:webProfile7 ; done
webProfile7: Pulling from library/websphere-liberty
d5c6f90da05d: Already exists
1300883d87d5: Already exists
c220aa3cfc1b: Already exists
2e9398f099dc: Already exists
dc27a084064f: Already exists
155fe9cd6124: Already exists
974b2337a80b: Already exists
0ad69ad38c5e: Already exists
21f9c31bf2e9: Already exists
453240fee003: Already exists
a2c82cb1af29: Already exists
c5ae97216ae8: Already exists
c941b70b6812: Already exists
b3b2397ccd01: Already exists
Digest:
sha256:dca823c618a7d4a481eeee7acd59b5bdae5d5775ee69015a76aa53df1580
ce28
Status: Downloaded newer image for websphere-liberty:webProfile7
4fbf4b2d5440491fcd5b04ee8c8eec2a478a5db7913dea2e7715078182443c12
be78761830eacd254c0a252942b9b801381281af9deb9a1ef4f65b3d35756b07
7ce18cb57aef7b4c6b38e2c006300c5d008889bcefd53d616f60541c5bfe0c16
c74159638e585207857d08f4d093757e786835606653331472c7731a396f410f
251ed1f67a23b67a3893b3880d13fc622c3a307d14c6d91c6847b2d33658bc34
```

7. Docker provides the `docker stats` command so that you can see the resources used by each docker instance. Run this command quickly after all of the containers are launched so you can see the CPU and memory spike when the containers are first started. Notice that after a short period of time, the CPU and memory use drops significantly to low

usage. This should give you a better feeling for the efficiency of docker containers. The `docker stats` command loops continuously, so you will need to stop it. -c

```
$ docker stats
CONTAINER         CPU %       MEM USAGE / LIMIT     MEM %      NET
I/O        BLOCK I/O        PIDS
4fbf4b2d5440      68.96%      49.14MiB / 1.952GiB   2.46%      578B
/ 0B      0B / 766kB       35
be78761830ea      64.32%      48.07MiB / 1.952GiB   2.41%      578B
/ 0B      0B / 643kB       36
7ce18cb57aef      82.19%      62.59MiB / 1.952GiB   3.13%      578B
/ 0B      0B / 1.28MB      36
c74159638e58      86.43%      67.93MiB / 1.952GiB   3.40%      712B
/ 0B      0B / 2.1MB       37
251ed1f67a23      88.11%      63.91MiB / 1.952GiB   3.20%      822B
/ 0B      123kB / 2.1MB    38

CONTAINER         CPU %       MEM USAGE / LIMIT     MEM %      NET
I/O        BLOCK I/O        PIDS
4fbf4b2d5440      0.43%       114.1MiB / 1.952GiB   5.71%      986B
/ 0B      4.1kB / 6.8MB    42
be78761830ea      0.59%       106.2MiB / 1.952GiB   5.31%
1.06kB / 0B    0B / 8.59MB      45
7ce18cb57aef      0.98%       112.3MiB / 1.952GiB   5.62%
1.06kB / 0B    0B / 8.46MB      42
c74159638e58      0.52%       107.2MiB / 1.952GiB   5.36%
1.12kB / 0B    0B / 7.74MB      42
251ed1f67a23      0.39%       102.5MiB / 1.952GiB   5.13%
1.3kB / 0B     123kB / 7MB      43
```

8. You can rerun `uptime` to compare the load before there were five docker instances running. Notice in this example there is virtually no difference in the load.

```
$ uptime
11:37  up 6 days, 20:30, 3 users, load averages: 2.71 3.01 2.96
```

9. Open a browser and go to `http://localhost` you will see the liberty welcome page. You should also verify that `http://localhost:81` loads as well as ports 82, 83, and 84.

10. You can see the running containers with the `docker ps` command. This will also give you the container ID for each container.

```
$ docker ps
CONTAINER ID    IMAGE                              COMMAND
CREATED
4fbf4b2d5440    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
be78761830ea    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
7ce18cb57aef    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
c74159638e58    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
251ed1f67a23    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
```

11. Before moving on you should stop the five Liberty instances. You can use the `docker stop` command with all five of the container IDs on the same line. As each container is stopped, that ID is returned.

```
$ docker stop 4fbf4b2d5440 be78761830ea 7ce18cb57aef c74159638e58
251ed1f67a23
4fbf4b2d5440
be78761830ea
7ce18cb57aef
c74159638e58
251ed1f67a23
```

Deploying a Java Application to IBM Cloud Private

In this lab, you deploy a Java application in containers to IBM Cloud Private. The application you will deploy is one of the component microservices of a reference application called Light Blue Compute. Light Blue Compute is a simplified version of the Blue Compute reference application offered on the IBM Cloud Architecture web site. The application stores and maintains images and descriptions for the vintage computer products in an online store. Its data is stored in a MySQL database, which is also deployed in a container.

In this lab, you will be deploying the application in containers in a manual way, by compiling the code, packaging it in a container, and deploying that container to the Kubernetes cluster in IBM Cloud Private. Doing it this way will help you to understand the various steps that are carried out by the more automated methods of deployment you will be working with later.

This exercise assumes that the IBM Cloud Private command line interface plugin has been installed, and that you are generally familiar with your IBM Cloud Private lab environment.

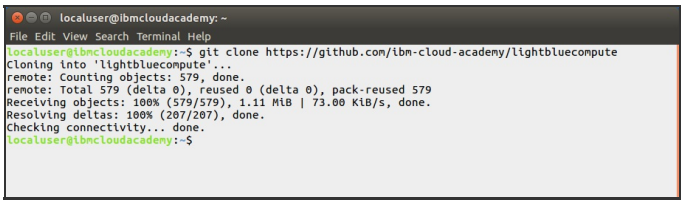
Note: In this exercise and others, there are various text files to be edited and configured. The command instructions in the lab assume the use of the `vi` editor. While `vi` is the prevalent historical text editor for Unix and Linux systems, it does have a learning curve and not all of our students will be familiar with it. If you are not comfortable with the `vi` editor, an alternative is to use `gedit`. You can use the `Files` application in the Application launcher to navigate to the file, then right-click to start the `gedit` editor.

Exercise 1: Deploying the MySQL Container

In this exercise, you create a container for a MySQL database. You deploy that container to your IBM Cloud Private cluster. You log in to the container and run a script to populate the MySQL database with the application data. Finally, you test to see that the data was successfully loaded.

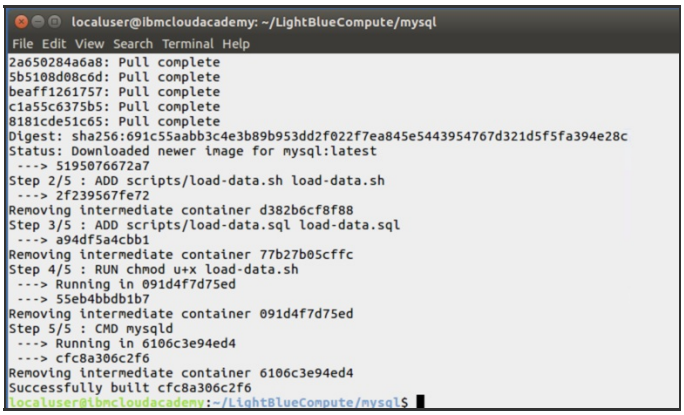
1. In your terminal window, clone the git repository that contains the code for the application you will be deploying.

   ```
   git clone https://github.com/ibm-cloud-academy/LightBlueCompute
   ```



2. Use the mysql folder contents and the commands below to build a docker image named `mysql`. The docker image is built using instructions in a file called `Dockerfile` in the `mysql` subdirectory. Edit the `Dockerfile` to see the commands if you haven't seen a Dockerfile before. When you go to build your container image, don't forget the dot (.) at the end of the `docker build` command.

```
cd ~/LightBlueCompute/mysql
docker build -t mysql .
```



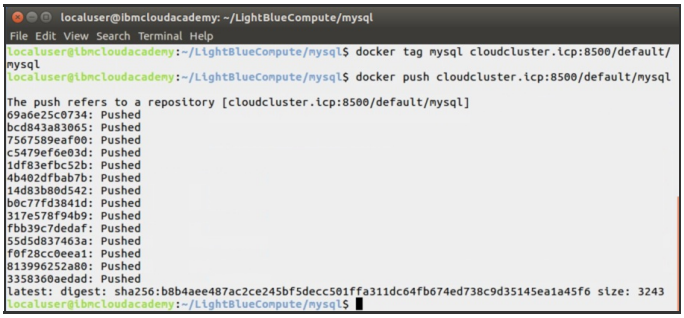
3. Log in to the IBM Cloud Private image registry for docker. This provides the local docker engine credentials to push images to the IBM Cloud Private image registry. Use Username `admin` and Password `passw0rd`. All communications with the IBM Cloud Private image registry use port 8500.

```
docker login cloudcluster.icp:8500
```

4. Tag and push the docker image to your IBM Cloud Private image registry. The `docker tag` command creates an alias name for the container image that associates it with the image repository and

namespace that you want to push it to. In this example we will use the `default` namespace that comes out of the box with IBM Cloud Private. You can also create other namespaces if you wish. Namespaces allow you to enforce access control to the images in your cluster.

```
docker tag mysql cloudcluster.icp:8500/default/mysql
docker push cloudcluster.icp:8500/default/mysql
```



5. Now that the image has been pushed to the IBM Cloud Private image registry, you should be able to see it in the User Interface. Go back to your browser session for IBM Cloud Private. In the menu on the top left, select `Menu > Catalog > Images`. You should see your image listed.

6. Deploy the mysql container to Kubernetes. This step uses a Kubernetes deployment yaml file. Using vi or the editor of your choice, edit the mysql.yml file in the kubernetes subdirectory

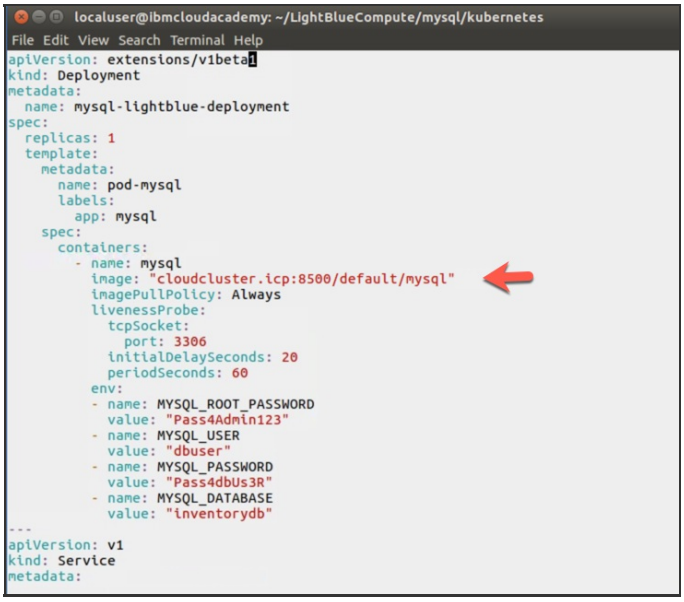
```
cd ~/LightBlueCompute/mysql/kubernetes
vi mysql.yml
```

Make the following modifications.

- Set your image source under `spec > template > spec > containers > image`. Make it match the tagged image name you just pushed to the IBM Cloud Private image registry.
- Answer the following questions:

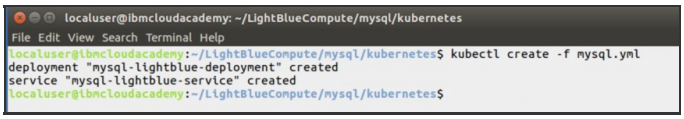
- How is the mysql health checked? _____
- What port will the mysql pod be contacted on? _____
- What is the database userID and password used? _____

- Save the yml file (`:wq <Enter>`)

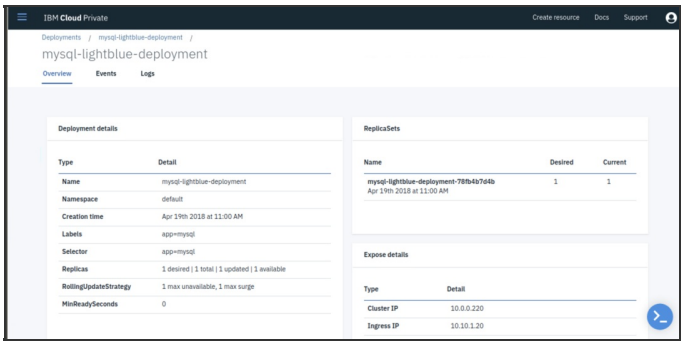


Deploy the yaml file using the following command:

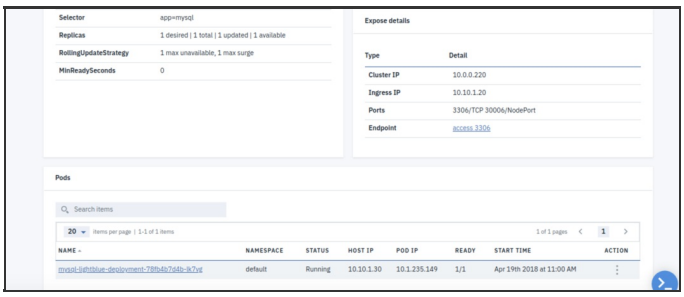
```
kubectl create -f mysql.yml
```



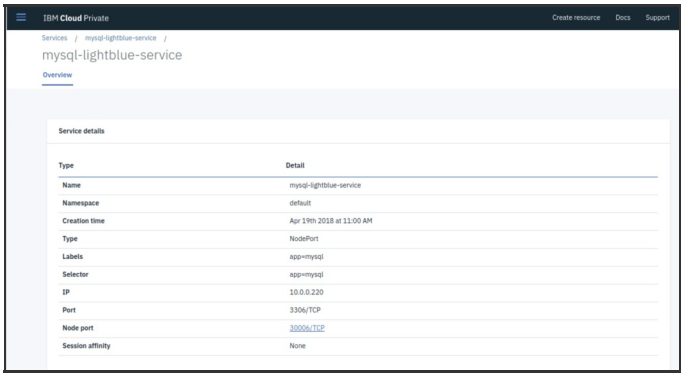
7. Now that you have created the deployment, and also created a Kubernetes service, as configured in the `mysql.yml` file, you should be able to see them in your IBM Cloud Private web user interface. Go to your browser session. Select `Menu > Workloads > Deployments`. You should see `mysql-lightblue-deployment` on the top of the list. Click that link to see more detailed information about this deployment.



Scroll down to see more information about the pod(s) the deployment has been deployed to.



8. You can also display details of the Kubernetes service that provides access to your MySQL database. In your IBM Cloud Private browser web interface, select `Menu > Network Access > Services`. You should see `mysql-lightblue-service` on top of the list. Click that link to see detailed information about your service.



9. Populate the table in the MySQL database

- Get the pod name. The pod name is based on the deployment name. To get the pod name, run the following command (all one line).

```
kubectl describe pod | grep mysql-lightblue-deployment | grep Name
```

- Open a shell session to the pod using the following command. Replace with the pod name returned by the previous command.

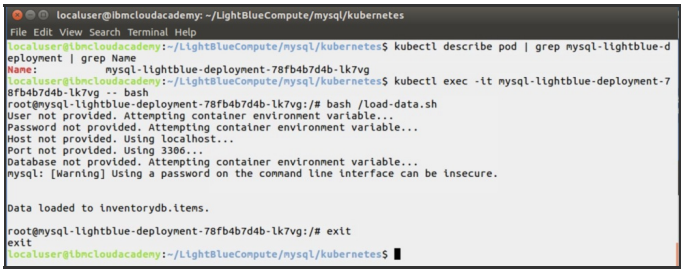
  `kubectl exec -it <podname> -- bash`

- Run the load-data.sh script, which populates the database:

  `bash /load-data.sh`

- Exit from the shell session:

  `exit`



10. Verify that the data is loaded and accessible. To do this, you will use the mysql client that is installed on your VM. In the mysql command below, substitute the mysql user, password and the nodePort found in the `mysql.yml` file.

The `<worker_publicIP>` is the public IP of your worker node. If you can't remember this value, use the command `bx pr workers cloudcluster` to retrieve it. You can use any of the worker nodes' IP addresses in your command.

The `mysql` command below does not tolerate spaces between the parameter and the value for some parameters. Therefore it is best to not use a space between any parameters and their values.

`mysql -u<mysqluser> -p<mysqlpassword> -h<worker_publicIP> -P<nodePort>`

```
          env:
          - name: MYSQL_ROOT_PASSWORD
            value: "Pass4Admin123"
          - name: MYSQL_USER
            value: "dbuser"
          - name: MYSQL_PASSWORD
            value: "Pass4dbUs3R"
          - name: MYSQL_DATABASE
            value: "inventorydb"
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-lightblue-service
  labels:
    app: mysql
spec:
  type: NodePort
  selector:
    app: mysql
  ports:
   - protocol: TCP
     port: 3306
     nodePort: 30006
```

Once connected, you can verify that the data has been loaded by running the following command:

```
select count(*) from inventorydb.items;
```

It should return a count of 12.

Type `quit` to exit.

```
localuser@ibmcloudacademy: ~/LightBlueCompute/mysql/kubernetes
File  Edit  View  Search  Terminal  Help
localuser@ibmcloudacademy:~/LightBlueCompute/mysql/kubernetes$ bx pr workers cloudcluster
OK
ID                          Private IP   Machine Type   State
cloudcluster-00000000-w1    10.10.1.20   -              deployed
cloudcluster-00000000-w2    10.10.1.30   -              deployed
localuser@ibmcloudacademy:~/LightBlueCompute/mysql/kubernetes$ mysql -udbuser -pPass4dbUs3R -h10.10.1.20
-P30006
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 29
Server version: 5.7.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select count(*) from inventorydb.items;
+----------+
| count(*) |
+----------+
|       12 |
+----------+
1 row in set (0.01 sec)

mysql> quit
Bye
localuser@ibmcloudacademy:~/LightBlueCompute/mysql/kubernetes$
```

Exercise 2: Deploying the Catalog Application

1. Explore the application.

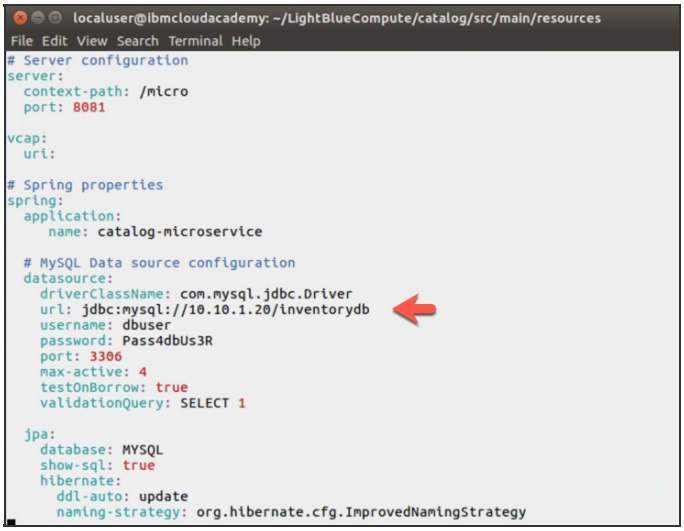
   ```
   cd ~/LightBlueCompute/catalog
   ```

   Check the existing application configuration in the application.yml file (`src/main/resources/application.yml`). What are the correct variable values for:

   - mysql URL: `_____`
   - mysql port: `_____`
   - mysql username: `_____`
   - mysql password: `_____`
   - application port: `_____`

   These values should match the values for the mysql container deployment you just did in the last section. Hint: The mysql URL should include the Worker IP address.

2. Modify the application.yml file to match the configuration of your mysql deployment.



   Note: While it is good practice to have the application.yml file match the target environment for the application, when you are deploying the application to kubernetes, as you will be, some of the parameters in this file will be superceded by parameters in a yaml file for the kubernetes service (in this example the file will be called `catalog.yml`) which you will be configuring in a future step. The configuration

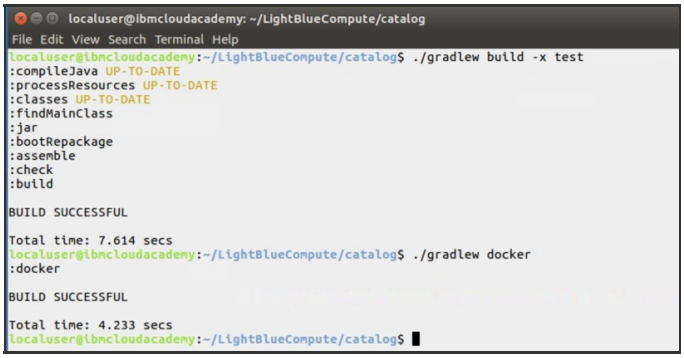
change you just made to the mysql URL parameter in this example is therefore optional.

3. Look at the CatalogController.java file. (`src/main/java/catalog/CatalogController.java`). Check the paths that the application will respond on. There is nothing for you to modify in this file. Looking at it is just for your own interest.

```
| Path                 | Method       |
|----------------------|--------------|
|                      |              |
|                      |              |
|                      |              |
|--------------------------------------|
```

4. Use the following commands to build the executable and copy the application jar file into the docker directory.

```
cd ~/LightBlueCompute/catalog
./gradlew build -x test
./gradlew docker
```

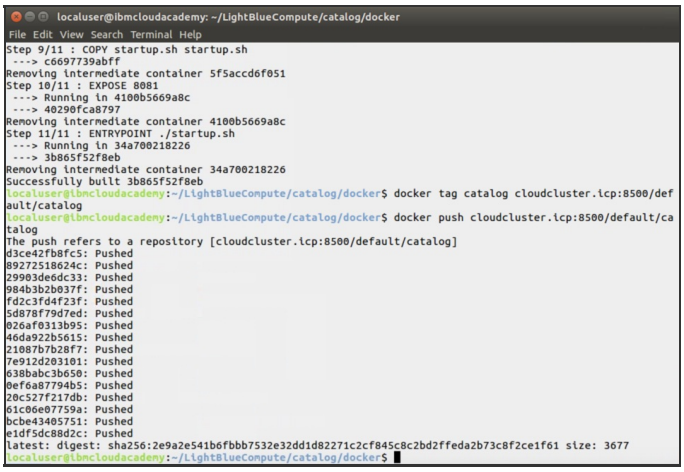


It is likely that your output will be more lengthy than that shown above. If newer versions of dependencies are available, they will be downloaded and this will lengthen the output. As long as you have a `BUILD SUCCESSFUL` message at the end, you are OK.

5. Build and upload the docker image. Don't forget the dot (.) at the end of the `docker build` command.

```
cd docker
docker build -t catalog .
```

```
docker tag catalog cloudcluster.icp:8500/default/catalog
docker push cloudcluster.icp:8500/default/catalog
```

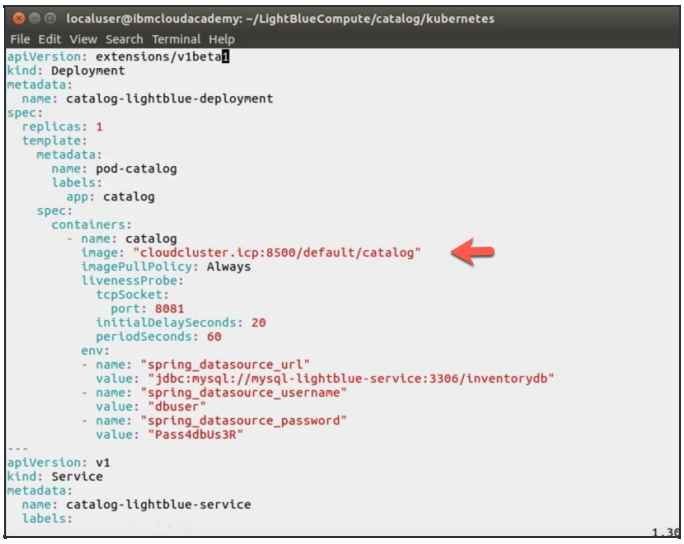


6. Using vi or the editor of your choice, edit the `catalog.yml` file in the kubernetes subdirectory. Modify the file so that the definition matches the name of your container image in the IBM Cloud Private registry.

   ```
   cd ~/LightBlueCompute/catalog/kubernetes
   vi catalog.yml
   ```

   Answer the following questions:

   - How does Kubernetes test this pod's health? `_____`
   - How does the catalog application connect to the mysql instance? `_____`
   - What port is the catalog application exposed at? `_____`
   - What is the service name for the catalog application? `_____`



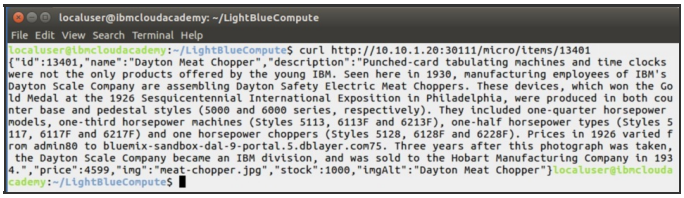
Deploy the yaml file using the command:

```
kubectl create -f catalog.yml
```

7. Now that you have sent the catalog deployment and service to your IBM Cloud Private cluster, you can visit the same pages in the web interface that you did for the mysql deployment to see the details.

8. Test the application. The following command should return the description of one of the products in the catalog.It uses the worker node IP Address and the Node Port of the catalog application service.

```
curl http://10.10.1.20:30111/micro/items/13401
```



If you see output of a product description like the example above, you can successfully deployed your application and completed this lab exercise.

END OF EXERCISE

Setting Up a Jenkins environment in IBM Cloud Private

These exercises set up a Jenkins environment in IBM Cloud Private and verify its operation. The exercises in this module are:

- Exercise 1: Setting up Jenkins
- Exercise 2: Working with Jenkins
- Exercise 3: Building a Sample Web Application

Exercise 1: Setting up Jenkins

Perform the following steps to set up a Jenkins environment in IBM Cloud Private:

1. Set up NFS. The master node is the NFS server. The master node already has NFS server software installed.
    - Open a terminal window.
    - SSH to the master node. Log in as **root** with a password of **passw0rd**.

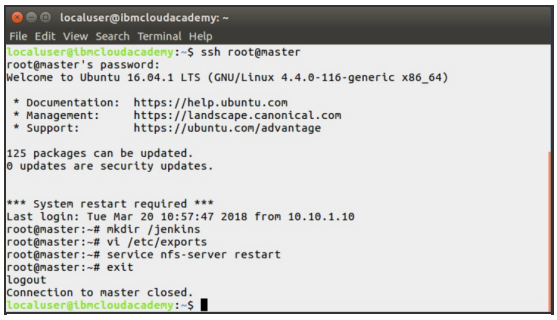
      `ssh root@master`
    - Create the /jenkins directory.

      `mkdir /jenkins`
    - Edit `/etc/exports` and add the following line:

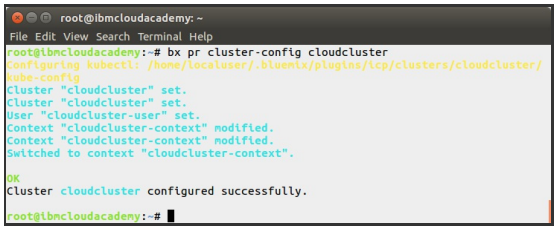
      `/jenkins *(rw,sync,no_root_squash)`
    - Restart the NFS server.

      `service nfs-server restart`
    - Close the connection to the master node.

      `exit`



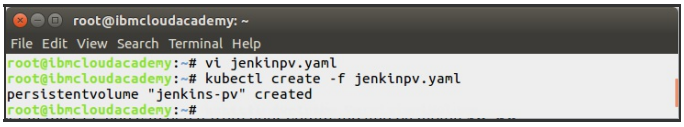
2. Prepare the PersistentVolume object.
    - Set up kubectl using the `bx pr cluster-config cloudcluster` command.



    - Create a file called **jenkinpv.yaml** in your current directory. The file should have the following contents.

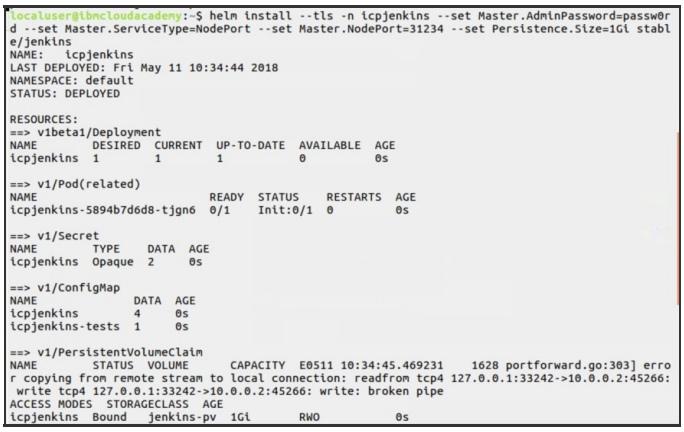
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkins-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    server: 10.10.1.10
    path: "/jenkins"
```

3. Create the PersistentVolume using the command `kubectl create -f jenkinpv.yaml`. This volume will be used to store the Jenkins configuration.



4. Install Jenkins in IBM Cloud Private. You will be using the Jenkins chart that is available from the kubernetes-charts site. There are many existing charts there that could be used. You can use the command `helm search` to see all the charts available, including the Jenkins one. Run the following command to deploy the Jenkins chart:

```
helm install --tls -n icpjenkins\
  --set Master.AdminPassword=passw0rd \
  --set Master.ServiceType=NodePort \
  --set Master.NodePort=31234 \
  --set Persistence.Size=1Gi \
  stable/jenkins
```

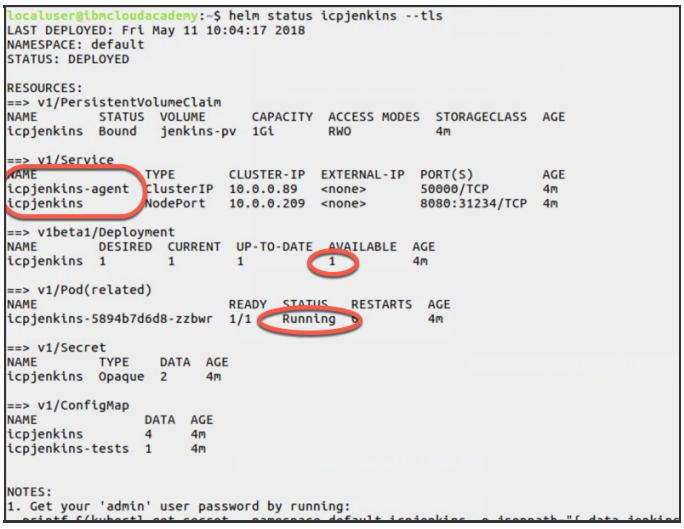


5. Check that Jenkins is deployed successfully using the command:

   `helm status icpjenkins --tls`

   Make sure that the Status of the Pod is `Running` and that there is a non-zero number of running instances.

   Note: If you are not seeing an available instance immediately, wait a few minutes and check again. It takes a few minutes for the Jenkins image to be downloaded and installed. Also, take note of the names of the 2 services that get created. You will need to configure Jenkins for these service names in a later step.



Exercise 2: Working with Jenkins

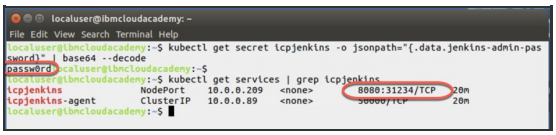
Perform the following steps to start working with your Jenkins environment:

1. Collect configuration information from your Helm release:
    - Get the Jenkins admin password (command is all one line):

```
kubectl get secret icpjenkins -o jsonpath="{.data.jenkins-admin-password}" | base64 --decode
```

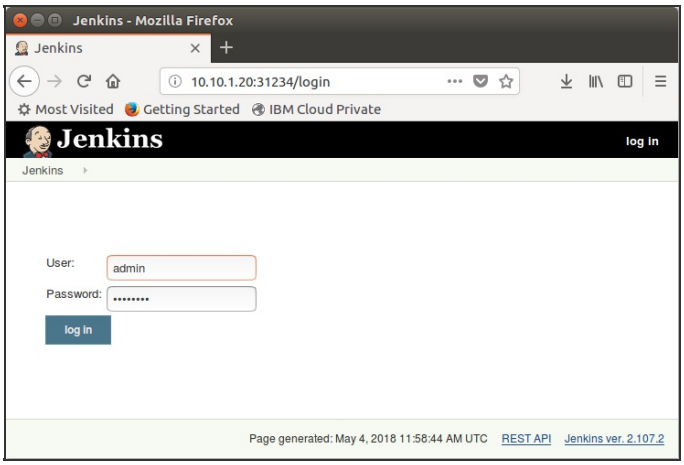
    - Get the Jenkins' port:

```
kubectl get services | grep icpjenkins
```



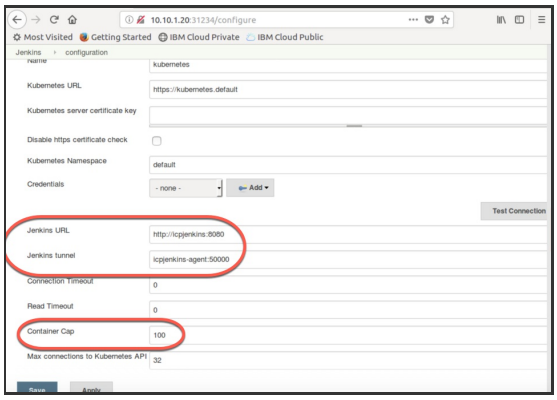
    The password should be `passw0rd` and the port should be `31234` as specified in the deployment parameters.

2. Open your browser to http://10.10.1.20:31234 . Log in as `admin` with the password that you retrieved from step 1.



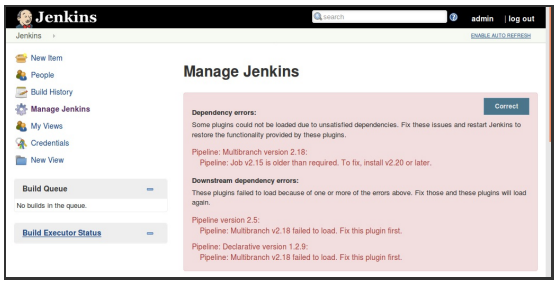
3. Configure the Jenkins server to match your installation and to increase the maximum number of containers.

    - Go to `http://10.10.1.20:31234/configure` and scroll to the **Kubernetes** area
    - Update the values for `Jenkins URL` and `Jenkins tunnel` to match the names of your services.
    - Change the Container Cap to be `100` and click **Save**.

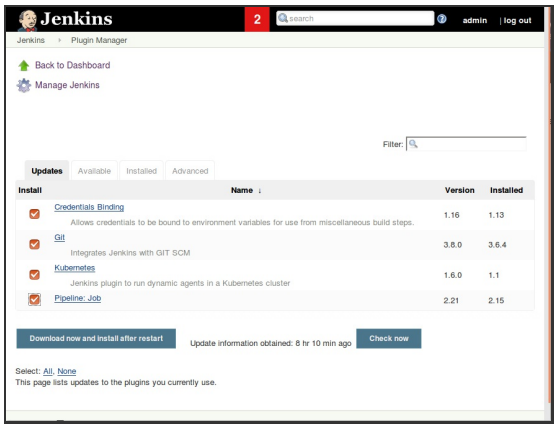


4. Update the Jenkins plugins:

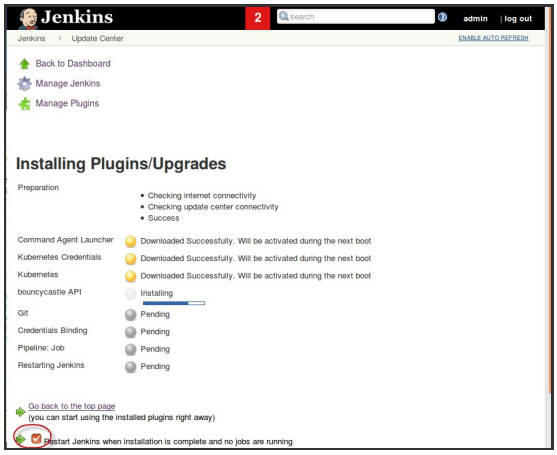
    - Click **Manage Jenkins** on the Jenkins home page.



- Click the **Correct** button on the top right of the page.
- Check all of the plugins and then click **Download now and Install after restart**.



- Check `Restart Jenkins when the installation is complete`.



- Refresh the screen and log back in to Jenkins if necessary

5. Define a secret to access the IBM Cloud Private user:

- Encode the user and password in base64:

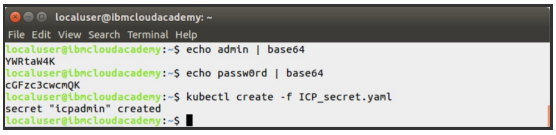
  `echo admin | base64`
  `echo passw0rd | base64`

- Create a ICP_secret.yaml file in your current directory with the following contents:

```
apiVersion: v1
kind: Secret
metadata:
  name: icpadmin
type: Opaque
data:
  username: <user-encoded>
  password: <password-encoded>
```

- Load the secret to ICP

  `kubectl create -f ICP_secret.yaml`



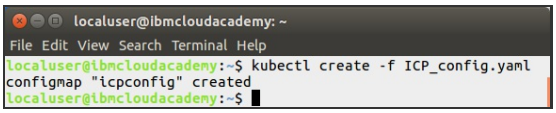
6. Set up IBM Cloud Private registry parameters as a ConfigMap (namespace, imagePullSecret and registry).

   - Create a ICP_config.yaml file in your current directory with the following contents:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: icpconfig
data:
  namespace: default
  registry: cloudcluster.icp:8500
```

   - Load the ConfigMap to ICP

     `kubectl create -f ICP_config.yaml`



7. What is the main difference between a Secret and a ConfigMap in Kubernetes?

   _____

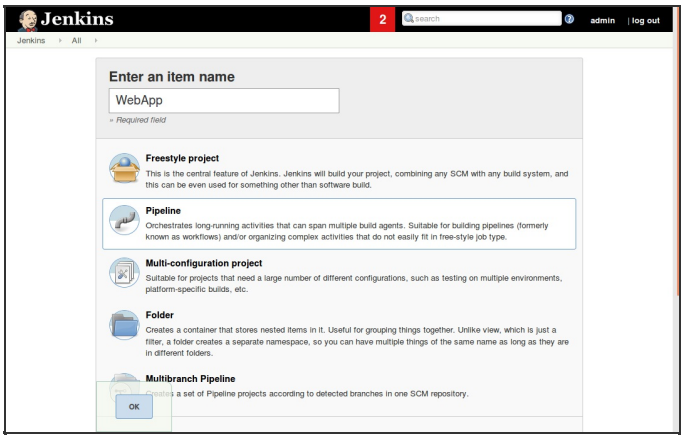
Exercise 3: Building a Sample Web Application

This exercise builds and deploys a Web application using Jenkins to verify that the Jenkins processes are working. The Web application will not be fully operational until you complete the last exercise in this course.

1. Login to Jenkins from the Jenkins Web UI. In your browser, go to `http://10.10.1.20:31234`. Log in as `admin` with a password of `passw0rd`. You should be in the Jenkins dashboard.

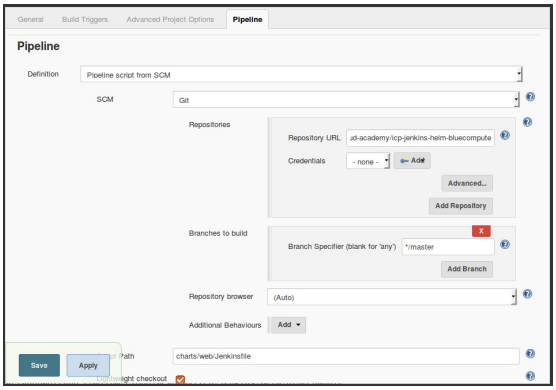
2. Click **New item** from the Jenkins menu on the left toolbar.

3. Enter a name of `WebApp`, select **Pipeline** and click **OK**.

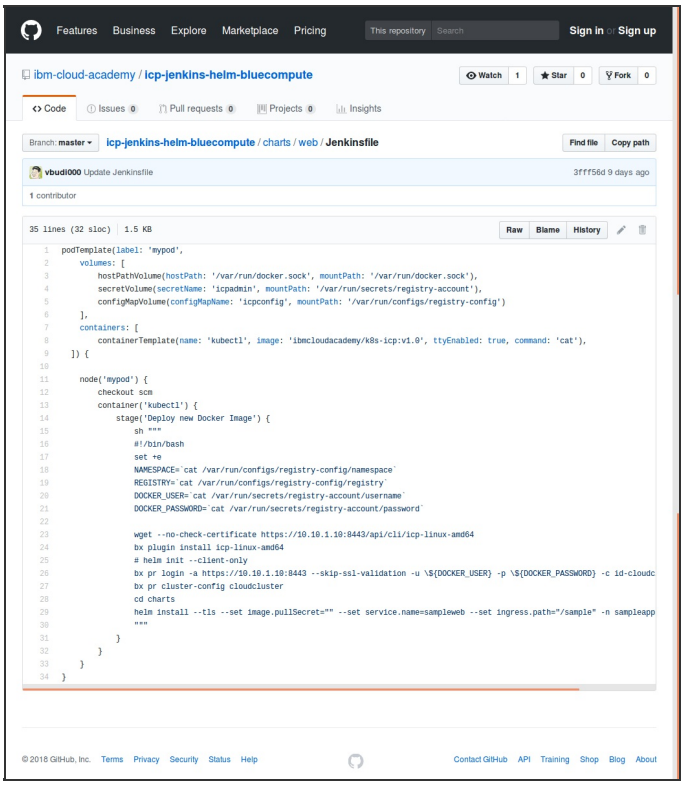


4. Click the **Pipeline** tab. Scroll down to the Pipeline section and specify the following definitions:

    - Definition: **Pipeline script from SCM**
    - SCM: **Git**
    - Repository URL: **https://github.com/ibm-cloud-academy/icp-jenkins-helm-bluecompute**
    - Branch Specifier: ***/master**
    - Script Path: **charts/web/Jenkinsfile**
    - Click **Save**

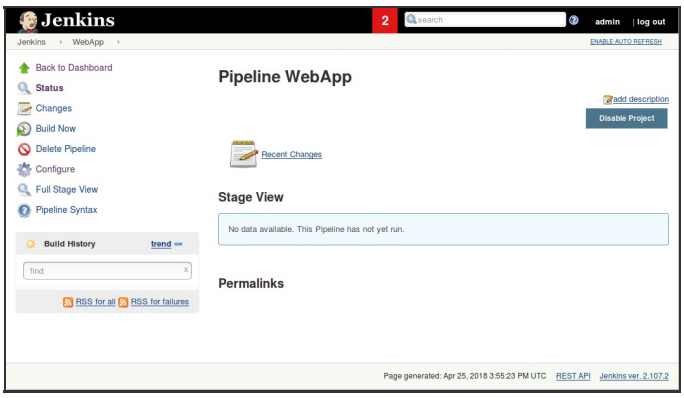


5. Open a new browser tab and navigate to the Jenkinsfile at `https://github.com/ibm-cloud-academy/icp-jenkins-helm-bluecompute`. Click to go to charts/web/Jenkinsfile.

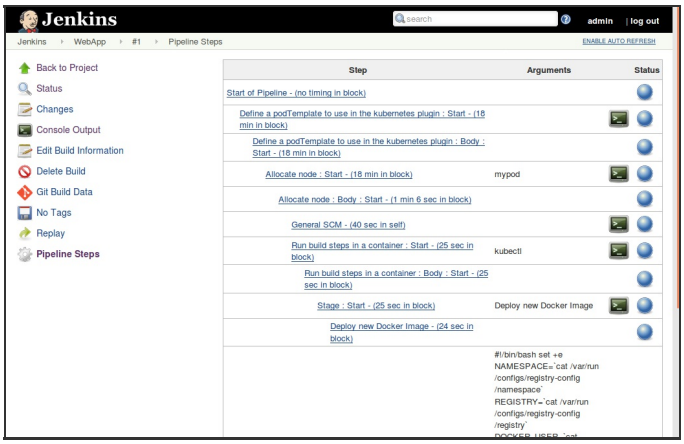


- How many stages are there in the pipeline? _____
- How does the pipeline script access the configMap and secret? _____

6. Back on the Jenkins browser tab, on the WebApp pipeline page, click **Build now**.



7. Once the pipeline is running as indicated on the lower left side, click on the Build number (`#1`), then select **Pipeline Steps**. These steps demonstrate the pipeline invocation hierarchy.

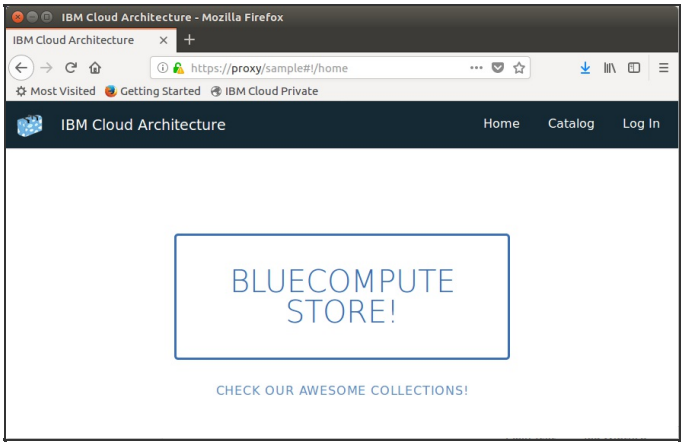


8. Again from the left navigation pane, select **Console Output**. The console should show the helm chart being deployed at the end and the pipeline finished successfully.

9. Having the helm chart deployed does not necessarily mean that the application is correctly deployed. You must check whether the actual application pod is running. This may take a couple of minutes depending on the network speed to load the container. Run `kubectl get pod | grep sampleapp` command or `helm status --tls sampleapp` command and wait until the pod status is `Running`.

```
localuser@ibmcloudacademy:~$ helm status --tls sampleapp
LAST DEPLOYED: Wed Apr 25 11:05:22 2018
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/Deployment
NAME           DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
sampleapp-web  1        1        1           1          20m

==> v1beta1/Ingress
NAME           HOSTS  ADDRESS     PORTS  AGE
sampleapp-web  *      10.10.1.20  80     20m

==> v1/Pod(related)
NAME                              READY  STATUS   RESTARTS  AGE
sampleapp-web-6dc67d7988-9pjft    1/1    Running  0         20m

==> v1/ConfigMap
NAME                DATA  AGE
sampleapp-web-config  2     20m

==> v1/Service
NAME           TYPE      CLUSTER-IP  EXTERNAL-IP  PORT(S)       AGE
sampleapp-web  NodePort  10.0.0.51   <none>       80:30626/TCP  20m
```

10. Test the Web application using the URL `http://10.10.1.20/sample`. Note that none of the links in that page is working. This application is part of a larger microservice application. Since the other microservices are not deployed, the links are not active.



11. You can remove this Web application using the following command.

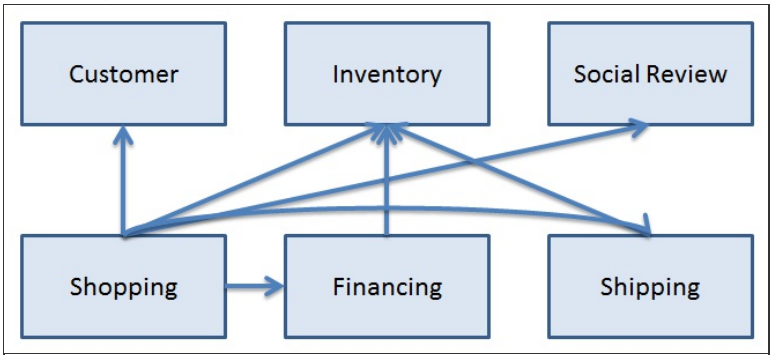
    `helm delete sampleapp --tls --purge`.

*** End of exercises ***

Designing Microservices lab

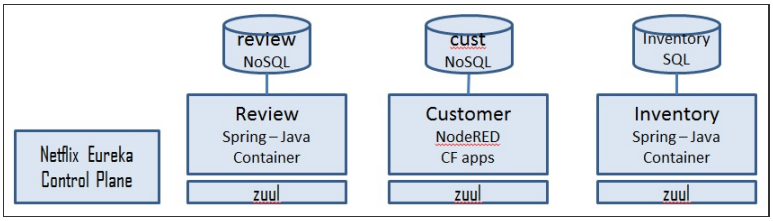
This lab is designed to be an thought exercise, with no real structure prescribed. The total length of session is around 1 hr with no actual coding. The practical hands on can be performed using the BlueCompute application. This exercise guide should be used by the instructor and expose the design step-by-step to finally arrive on the application structure. Student will not see this material directly, instead should be guided by the instructor to show parts of this instructions.

1. Illustrating goal: **online store application**
    - The application in its infancy will build a bare minimum system
    - Some of the given are:
        - the inventory is stored in the corporate SQL database and wont be changed in the near future.
        - user review of the product is one of the important feature that is expected to be available in the first iteration of the product.
    - Discuss some of the design considerations and boundary:
        - Some of the goods are quite expensive financing is an option
        - Shipping can take a large portion of the cost
2. Define microservices:
    - What are the possible breakdown of the services?
    - Discuss how the microservices can be dependent on each other
    - Discuss how the API would look like roughly
    - The following diagram is the suggessted structure that this guide is following. There can be other implementations

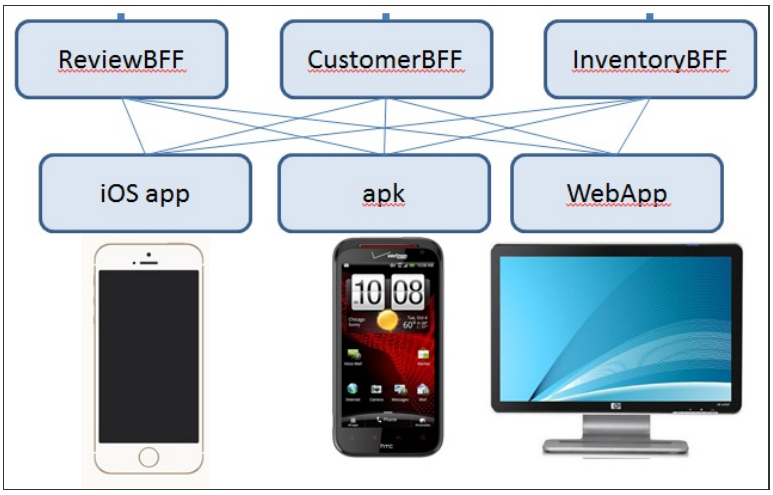


    - Inventory:
        - `GET /inventory/<itemId>`
        - `PUT /inventory/<itemId>`
        - `POST /inventory` ← JSON input
        - `GET /inventory`

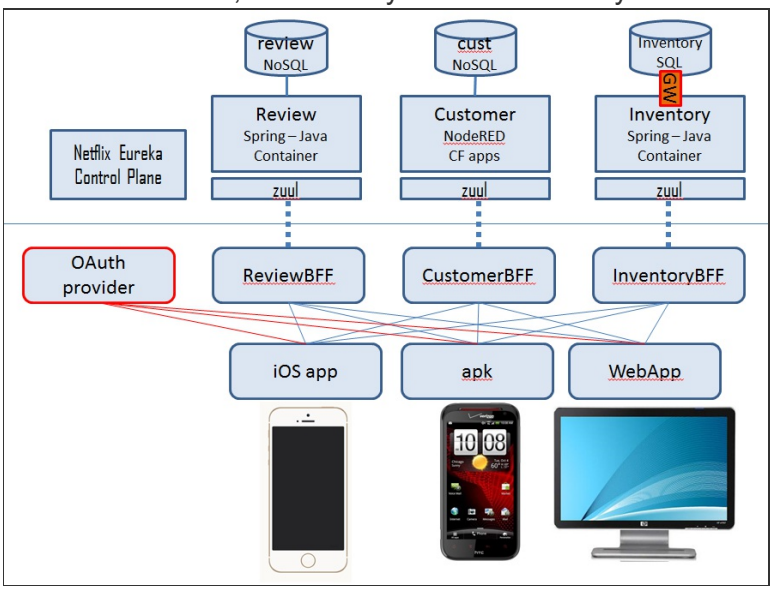
        - `DELETE /inventory/<itemId>`
    - Social review
        - `POST /flight/list/<origin>/<dest>/<date>` ← array of Flight info
            - Flight info = { itinieraryId , fare , [orig, origin-time, dest, dest-time, Flightnum], numseat: int }
        - `POST /flight/book/<origin>/<dest>/<date>` ← flight info
    - Ordering
        - `GET /booking/<custId>`
        - `POST /booking/<custId>` ← flightInfo + #numBook this will
        - `PUT /booking/<custId>/<bookingId>`
        - `GET /booking/<custId>/<bookingId>`
        - `DELETE /booking/<custId>/<bookingId>`
    - Financing
    - Customer
3. Add framework and compute options
    - Service discovery and proxy: Amalgam8 - Eureka
    - Circuit Breaker: Hystrix
    - Compute technology: Container - CF apps
    - Programming language (polyglot?): Node.js - Java - PHP
    - Persistent layer: API call - NoSQL - SQL



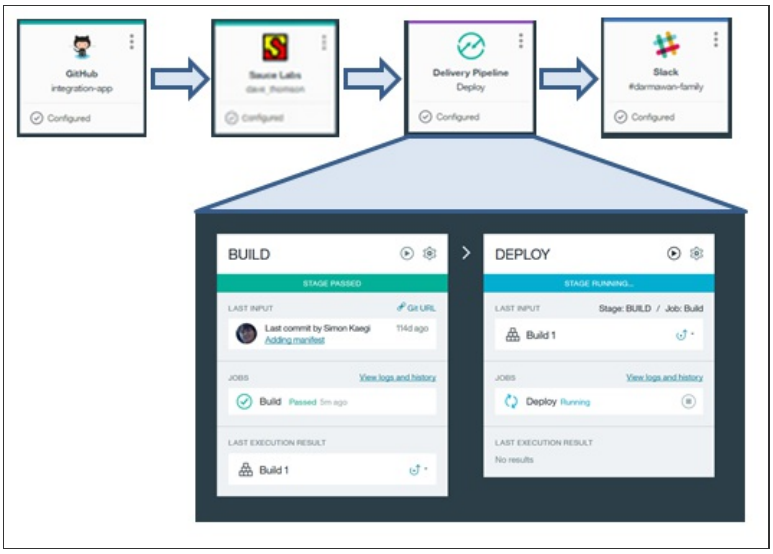
4. Define front-end components (platform, language, requirements): Discuss the following:
    - BFF: why do you need a BFF? what the BFF do?
    - Mobile: ios - android
    - Web based
    - API management needed? benefit/limitation



5. Security considerations
    - logging in: OAuth - SSO
    - intra component communication: JWT
    - Backend interface, connectivity: Secure Gateway - VPN



6. DevOps thinking: Discuss the following topics to adhere to 12-factors and other design topics
    - GIT repository
    - Pipeline development
    - Slack notification
    - Test suite (Sauce lab - App Security - homegrown)



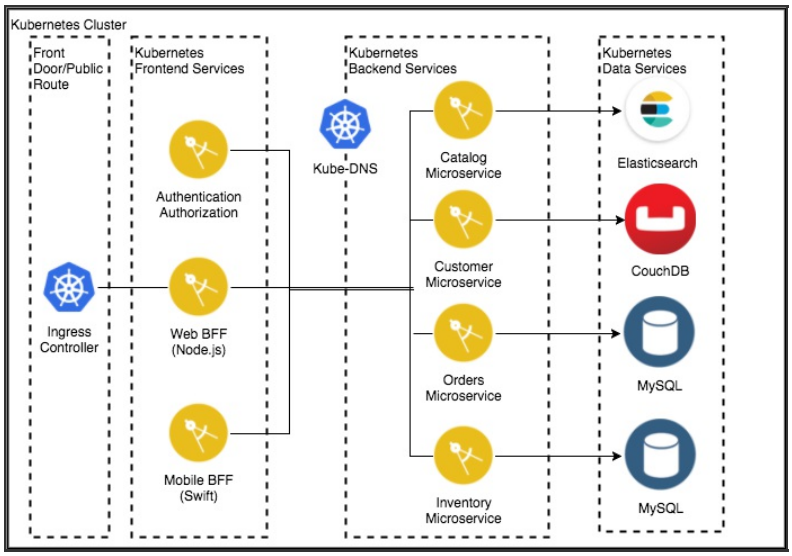
7. Resiliency: Discuss the techniques for doing resiliency across multiple data centers, database replication and load balancing (Dyn, Akamai etc)
    - Failover - load balancing
    - Disaster recovery - site switching

Deploying the BlueCompute Microservices Application

These exercises deploy the BlueCompute application. This microservices application is created as a showcase of developing IBM Cloud capabilities. You will use Jenkins to deploy the application components.

Exercise 1: Designing the Microservices Deployment

The application is shown in the following diagram.



You will perform this section of the exercises together.

1. As discussed in the refactoring exercise, the BlueCompute application contains a frontend layer, a backend layer (microservices) and a data layer (backing services).

2. Let's go through this exercise to design how you will group and deploy the application. The deployment should starting with the data layer, followed by the backend layer and the frontend layer.

3. For the following data services resource, which Kubernetes resources should you define?

    - ElasticSearch: _____

- CouchDB: _____
- Orders MySQL: _____
- Inventory MySQL: _____

4. For the following backend services microservice applications, which Kubernetes resources should you define?

   - Catalog microservice: _____
   - Customer microservice: _____
   - Orders microservice: _____
   - Inventory microservice: _____

5. For the following frontend services application, which Kubernetes resources should you define? Assume for now that the Ingress resource will be deployed with the Web application and that you will ignore the swift Web BFF.

   - Authentication: _____
   - Web BFF: _____

6. Now that you have been through the design part, let's consider the following:

   - What resource(s) is/are common for all components?
     _____
   - What resource(s) is/are common for the data layer?
     _____
   - When would you have an ingress resource?
     _____

Exercise 2: Setting Up and Verifying Backing Services

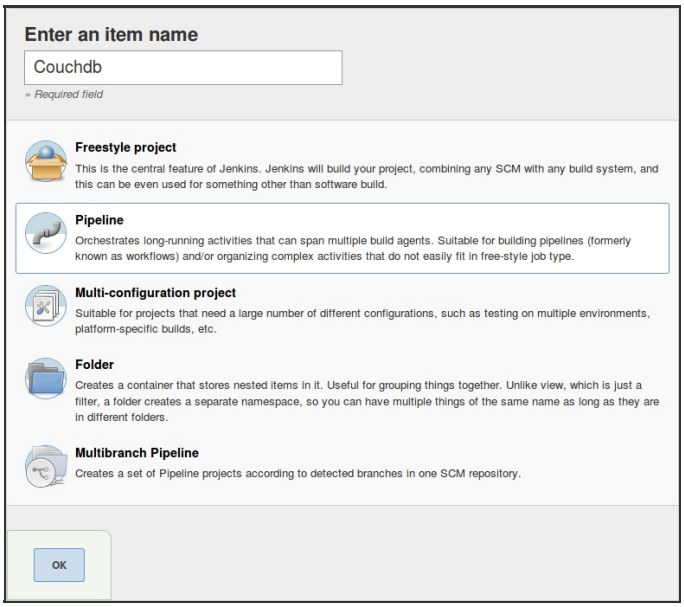
The following instructions allow you to build and deploy the backend services required by the BlueCompute application.

Note: Better than just following these steps, you can learn much more by looking at the Jenkinsfile and helm charts you will be working with. This will help you understand what actually happens in the process.

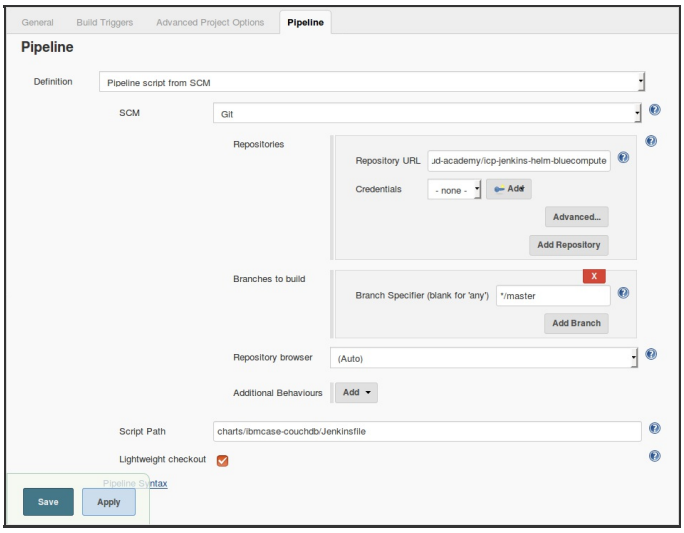
1. In your browser, go to the Jenkins Web UI at `http://proxy:31234`. Log in as `admin` with a password of `passw0rd`. You should be in the Jenkins dashboard. The IP address for `proxy` is `10.10.1.20`.

2. Click **New item** from the Jenkins menu on the left toolbar.

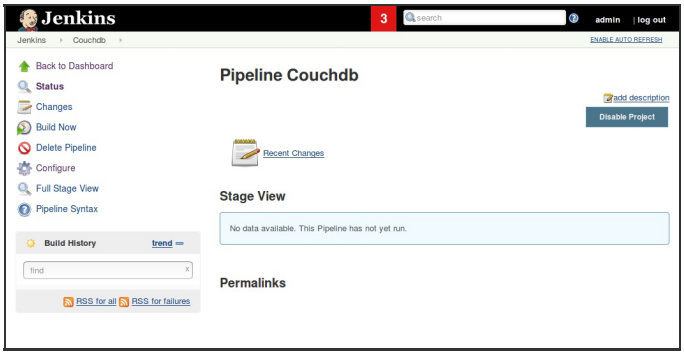
3. Enter a name of `CouchDB`, select **Pipeline** and click **OK**.



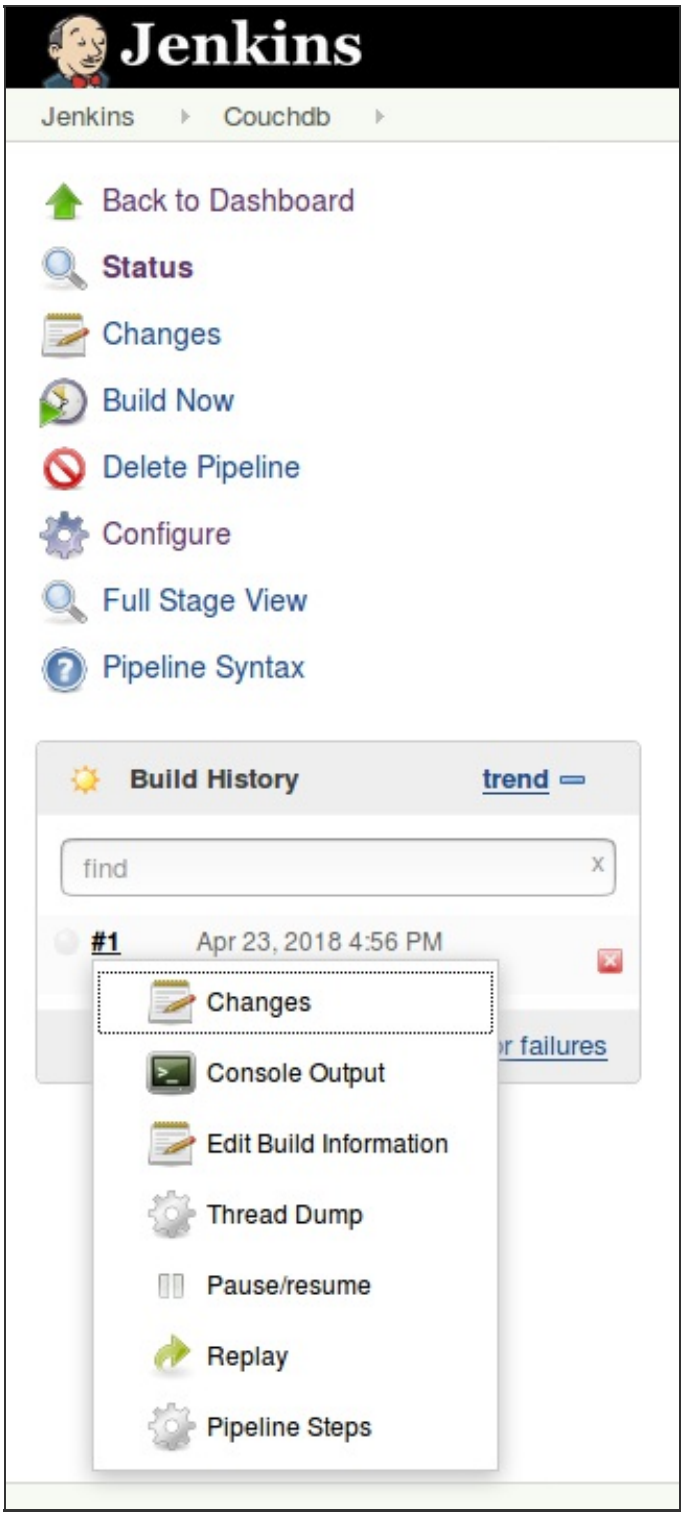
4. Scroll down to the Pipeline section and specify:
    - Definition: `Pipeline script from SCM`
    - SCM: `Git`
    - Repository URL: `https://github.com/ibm-cloud-academy/icp-jenkins-helm-bluecompute`
    - Branch Specifier: `*/master`
    - Script Path: `charts/ibmcase-couchdb/Jenkinsfile`
    - Click **Save**



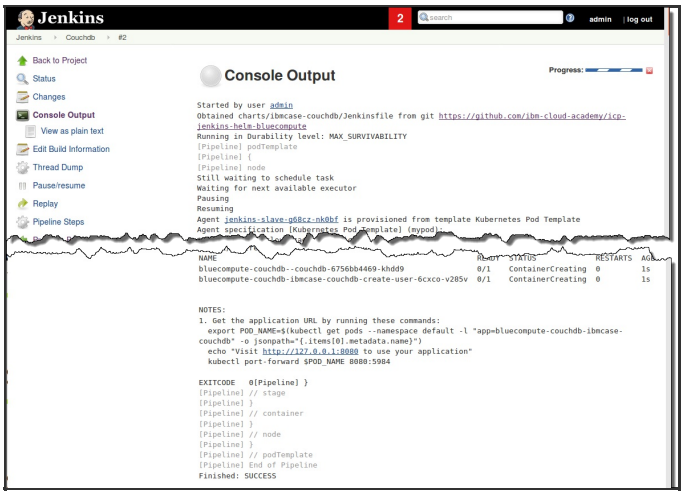
5. On the couchDB pipeline page, click **Build now**.



6. Once the pipeline is running as indicated on the lower left side, open the drop down menu next to the run number and select **Console Output**.

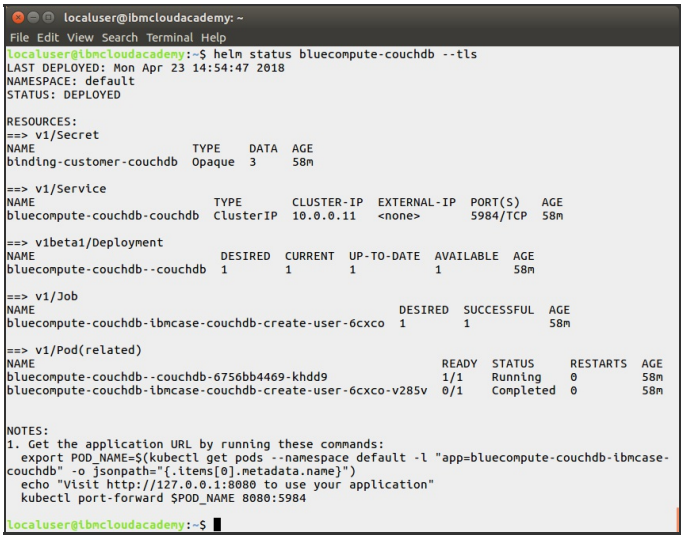


7. The console should show the helm chart being deployed at the end and the pipeline having finished successfully.



8. Having the helm chart deployed does not necessarily mean that the application is correctly deployed. You must check whether the actual application pod is running. This may take a couple of minutes depending on the network speed to load the container. Run `kubectl get pod` commands and wait until the pod for couchdb is running.

9. Check the status of the `bluecompute-couchdb` release, by running `helm status --tls bluecompute-couchdb`.



Can you identify the chart's components?

- Secret: `binding-customer-couchdb`

- Service: `bluecompute-couchdb-couchdb`

- Deployment: `bluecompute-couchdb--couchdb`

- Job: `bluecompute-couchdb-ibmcase-couchdb-create-user-"*"`

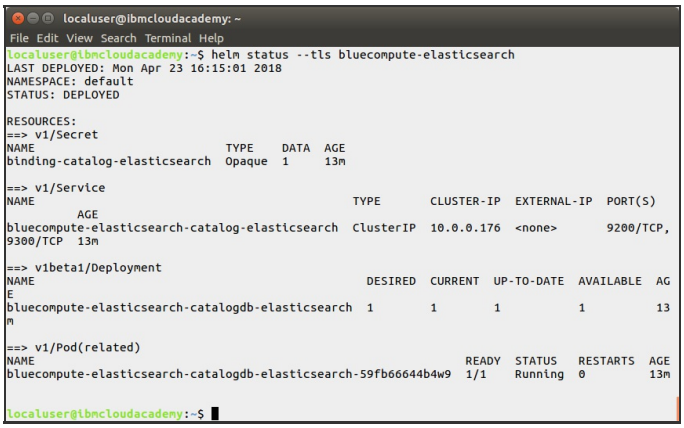
- Pods: `bluecompute-couchdb--couchdb-"*"` and `bluecompute-couchdb-ibmcase-couchdb-create-user-"*"`

Note: The deployment of the other backend components (elasticsearch, inventory-mysql and orders-mysql) is actually very similar. You create the Jenkins pipelines from SCM (GIT) and run them individually. These pipelines are not automated (triggered using code changes) as this is meant to be a stable backend. In a real production environment, you would want to add a PersistentVolumeClaim to physically host the data instead of storing it in volatile containers as this example describes.

10. Deploy `elasticsearch` by creating a new pipeline, with the following parameters:

    - Name: `Elasticsearch`
    - Repository URL: `https://github.com/ibm-cloud-academy/icp-jenkins-helm-bluecompute`
    - Branch Specifier: `*/master`
    - Script Path: `charts/ibmcase-elasticsearch/Jenkinsfile`
    - Run the pipeline using the **Build now** link.
    - Make sure the `bluecompute-elasticsearch` pod is running.

11. Check the status of the `bluecompute-elasticsearch` release, by running `helm status --tls bluecompute-elasticsearch`.



Can you identify the chart's components?

- Secret: `binding-catalog-elasticsearch`

  _____

- Service: `bluecompute-elasticsearch-catalog-elasticsearch`

  _____

- Deployment: `bluecompute-elasticsearch-catalogdb-elasticsearch`

  _____

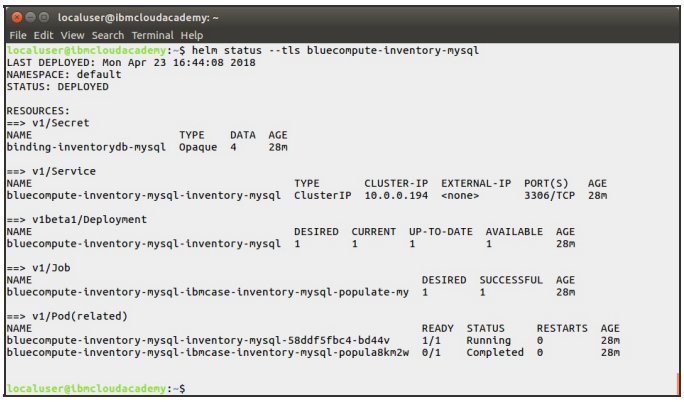
- Pod: `bluecompute-elasticsearch-catalogdb-elasticsearch`-

  _____

12. Deploy `inventory-mysql` by creating a new pipeline, with the following parameters:

    - Name: `Inventory-mysql`
    - Repository URL: `https://github.com/ibm-cloud-academy/icp-jenkins-helm-bluecompute`
    - Branch Specifier: `*/master`
    - Script Path: `charts/ibmcase-inventory-mysql/Jenkinsfile`
    - Run the pipeline using the **Build now** link.
    - Make sure the `bluecompute-inventory-mysql` pod is running

13. Check the status of the `bluecompute-inventory-mysql` release, by running `helm status --tls bluecompute-inventory-mysql`.



Can you identify the chart's components?

- Secret: `binding-inventorydb-mysql`

  _____

- Service: `bluecompute-inventory-mysql-inventory-mysql`

  _____

- Deployment: `bluecompute-inventory-mysql-inventory-mysql`

  _____

- Job: `bluecompute-inventory-mysql-ibmcase-inventory-mysql-populate*`

  _____

- Pods: `bluecompute-inventory-mysql-inventory-mysql-*` and `bluecompute-inventory-mysql-ibmcase-inventory-mysql-popu*`

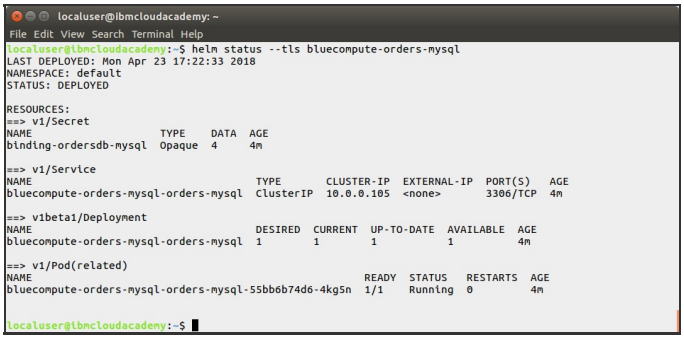
  _____

14. Deploy `orders-mysql` by creating a new pipeline, with the following parameters:

- Name: `Orders-mysql`
- Repository URL: `https://github.com/ibm-cloud-academy/icp-jenkins-helm-bluecompute`
- Branch Specifier: `*/master`

- Script Path: `charts/ibmcase-orders-mysql/Jenkinsfile`
- Run the pipeline using the **Build now** link.
- Make sure the `bluecompute-orders-mysql` pod is running

15. Check the status of the `bluecompute-orders-mysql` release, by running `helm status --tls bluecompute-orders-mysql`.



Can you identify the chart's components?

- Secret: `binding-ordersdb-mysql`

  _____

- Service: `bluecompute-orders-mysql-orders-mysql`

  _____

- Deployment: `bluecompute-orders-mysql-orders-mysql`

  _____

- Pod: `bluecompute-orders-mysql-orders-mysql-*`

  _____

16. Answer the following questions:

- How would you get the URL and credentials to access the backend services?

  _____

  _____

- Why do some charts contain job(s) while other don't?

  _____

- What other resource is typically present for a backend service?

  _____

Exercise 3: Deploying Microservices Components

The set of charts in this exercise is different that in the previous set. The main difference is that the docker images are built by the pipeline and loaded to the IBM Cloud Private local registry.

Note:

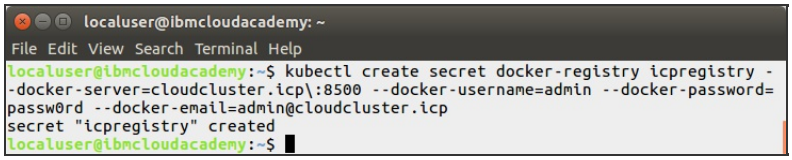
These instructions load the Jenkinsfile directly from a forked repository that has been pre-configured. The actual steps should have been:

- Fork the repository.
- Create a Jenkinsfile.
- Poll the repository for commit changes and create a post-commit hook.

You would never need to build the pipeline manually.

1. There are 4 microservices built into the BlueCompute application: customer, inventory, catalog and orders. You will deploy them similar to deploying the backend resources, but using different source git repositories. You must first create an **imagePullSecret** to authenticate to a private registry using the following command (all one line):

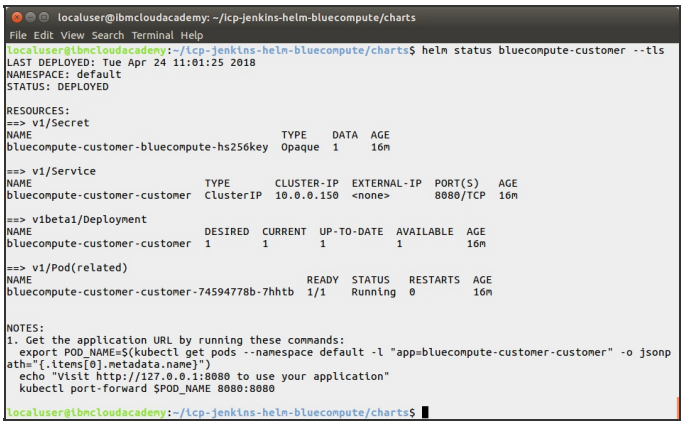
```
kubectl create secret docker-registry icpregistry \
  --docker-server=cloudcluster.icp:8500 \
  --docker-username=admin --docker-password=passw0rd \
  --docker-email=admin@cloudcluster.icp
```



2. Create a new Jenkins pipeline using the following specifications for the `customer` application.
    - Name: `Customer`
    - Repository URL: `https://github.com/ibm-cloud-academy/refarch-cloudnative-micro-customer`

- Branch Specifier: `*/master`
- Script Path: `Jenkinsfile`
- Run the pipeline using the **Build now** link.
- Make sure the `customer` pod is running

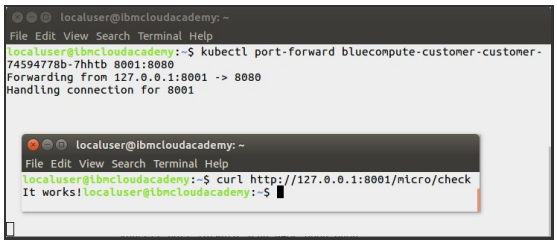
3. Check the status of the `bluecompute-customer` release, by running `helm status --tls bluecompute-customer`.



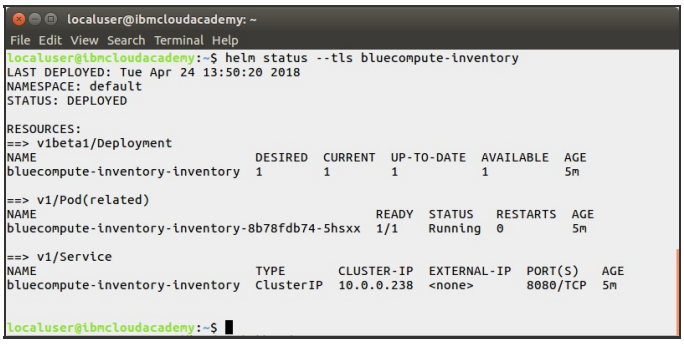
4. Test the `customer` application's health:
   - Open a terminal window
   - Run the command `kubectl port-forward <customerpod> 8080:8080` The command maps the pod 8080 to the localhost.
   - Open another terminal window and check the microservice. Note that most of the usual microservice REST calls for the `customer` application are protected by JWT. Therefore, you can only easily test the check call which returns the string `It works`.

```
curl http://127.0.0.1:8080/micro/check
```

   - Stop the port forwarding by typing `Ctrl-C`.



5. Create a new Jenkins pipeline using the following specifications for the `inventory` application.
    - Name: `Inventory`
    - Repository URL: `https://github.com/ibm-cloud-academy/refarch-cloudnative-micro-inventory`
    - Branch Specifier: `*/master`
    - Script Path: `inventory/Jenkinsfile`
    - Run the pipeline using the **Build now** link.
    - Make sure the `inventory` pod is running
6. Check the status of the `bluecompute-inventory` release, by running `helm status --tls bluecompute-inventory`.

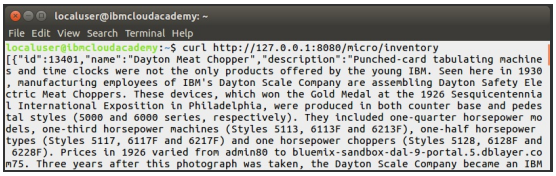


7. Test the `inventory` application's health:
    - Open a terminal window
    - Run the command `kubectl port-forward <inventorypod> 8080:8080` The command maps the pod 8080 to the localhost port 8080.

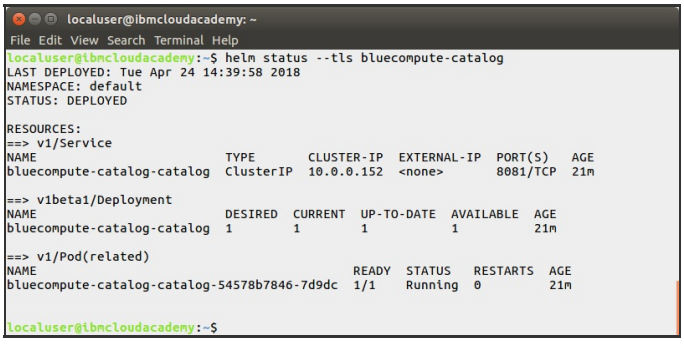
- Open another terminal window and check the microservices. Note that this microservice is not secured (how do you know that? _____)

```
curl http://127.0.0.1:8080/micro/inventory
```

- Stop the port forwarding by typing `Ctrl-C`.



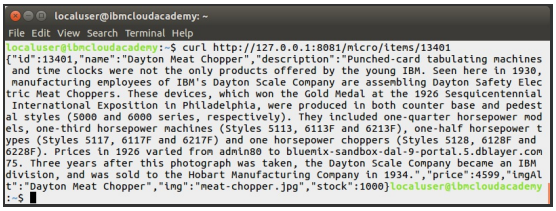
8. Create a new Jenkins pipeline using the following specifications for the `catalog` application. Note that the repository below is correct. The `catalog` application is in the `inventory` repository.
    - Name: `Catalog`
    - Repository URL: `https://github.com/ibm-cloud-academy/refarch-cloudnative-micro-inventory`
    - Branch Specifier: `*/master`
    - Script Path: `catalog/Jenkinsfile`
    - Run the pipeline using the **Build now** link.
    - Make sure the `catalog` pod is running
9. Check the status of the `bluecompute-catalog` release, by running `helm status --tls bluecompute-catalog`.



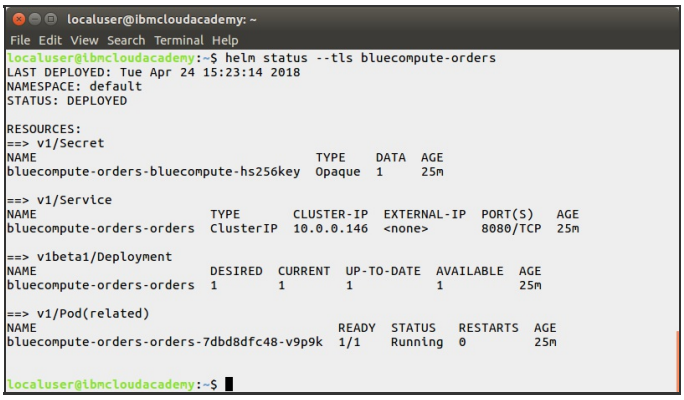
10. Test the `catalog` application's health:
    - Open a terminal window
    - Run the command `kubectl port-forward <catalogpod> 8081:8081` The command maps the pod 8081 to the localhost port 8081.
    - How do you know that this application uses port 8081?

      _____
    - Open another terminal window and check the microservice. Note that this microservice is not secured

      ```
      curl http://127.0.0.1:8081/micro/items/13401
      ```
    - Stop the port forwarding by typing `Ctrl-C`.



11. Create a new Jenkins pipeline using the following specifications for the `orders` application.
    - Name: `Orders`
    - Repository URL: `https://github.com/ibm-cloud-academy/refarch-cloudnative-micro-orders`
    - Branch Specifier: `*/master`
    - Script Path: `Jenkinsfile`
    - Run the pipeline using the **Build now** link.
    - Make sure the `orders` pod is running
12. Check the status of the `bluecompute-orders` release, by running `helm status --tls bluecompute-orders`.



13. Test the `orders` application's health:
    - Open a terminal window
    - Run the command `kubectl port-forward <orderspod> 8080:8080` The command maps the pod 8080 to the localhost port 8080.
    - Is this microservice being secured using JWT? ____

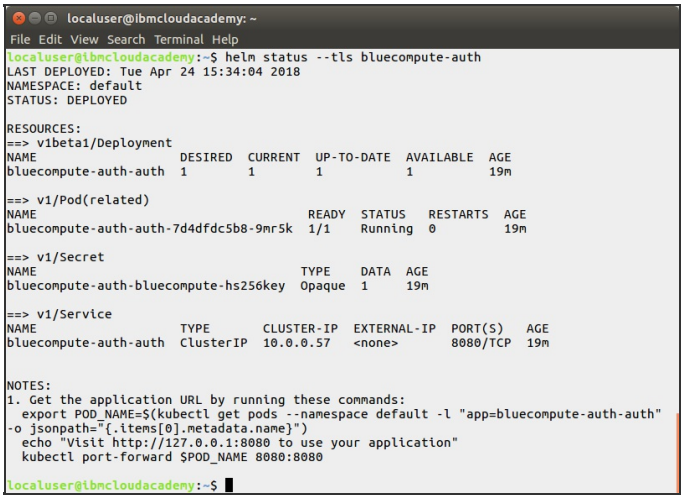
      ```
      curl http://127.0.0.1:8080/micro/check
      ```

    - Stop the port forwarding by typing `Ctrl-C`.

14. Create a new Jenkins pipeline using the following specifications for the `authentication` application.
    - Name: `Auth`
    - Repository URL: `https://github.com/ibm-cloud-academy/refarch-cloudnative-auth`
    - Branch Specifier: `*/master`
    - Script Path: `Jenkinsfile`
    - Run the pipeline using the **Build now** link.

- Make sure the `auth` pod is running

15. Check the status of the `bluecompute-auth` release, by running `helm status --tls bluecompute-auth`.

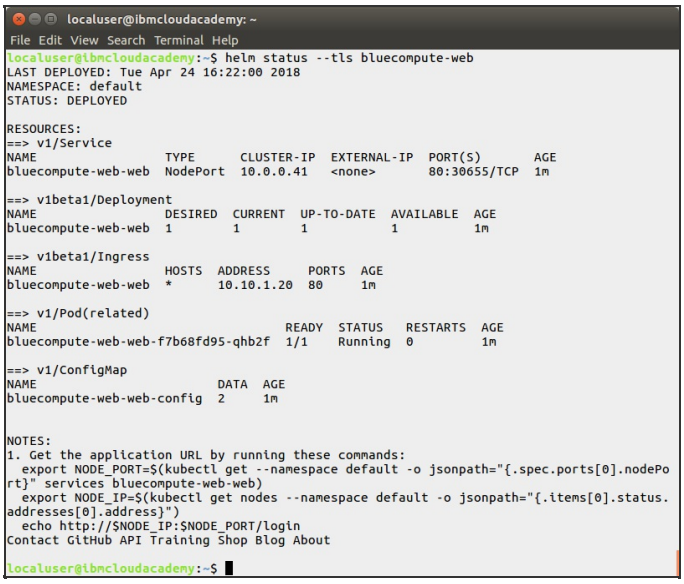


16. You will test the oauth authentication later after the Web application has been deployed.

17. Create a new Jenkins pipeline using the following specifications for the `Web` application.

    - Name: `Web`
    - Repository URL: `https://github.com/ibm-cloud-academy/refarch-cloudnative-bluecompute-web`
    - Branch Specifier: `*/master`
    - Script Path: `Jenkinsfile`
    - Run the pipeline using the **Build now** link.
    - Make sure the `bluecompute-web` pod is running

18. Check the status of the `bluecompute-web` release, by running `helm status --tls bluecompute-web`.



19. What resources are created in the Web application? What are the usage of those resources?

    - ConfigMap:

      _____

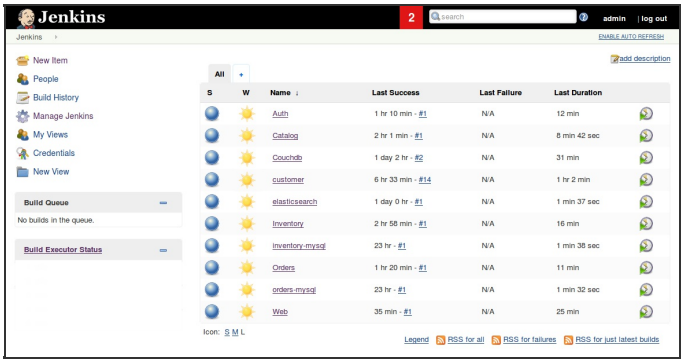
      _____

    - Ingress:

      _____

      _____

20. The final state of the Jenkins dashboard should be similar to the following:

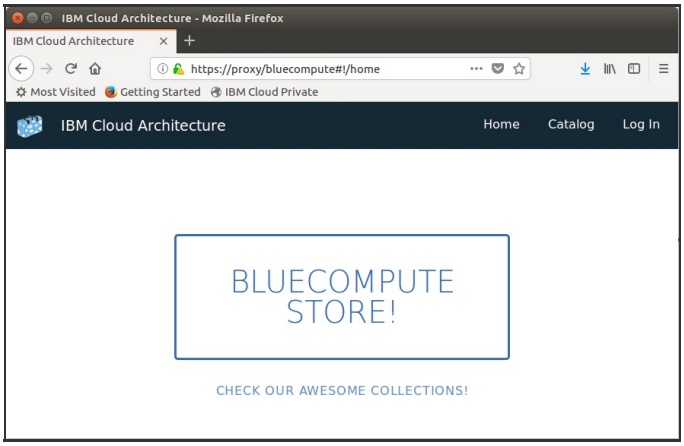


Exercise 4: Validating the Application and Resources

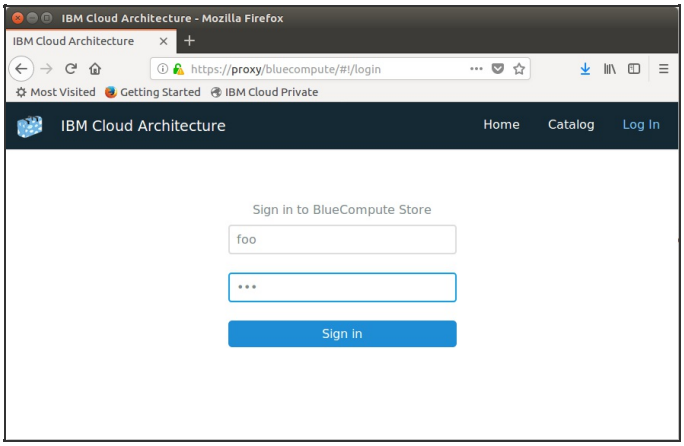
Now that you have completed all the deployments successfully, it's time to test the

overall microservice application.

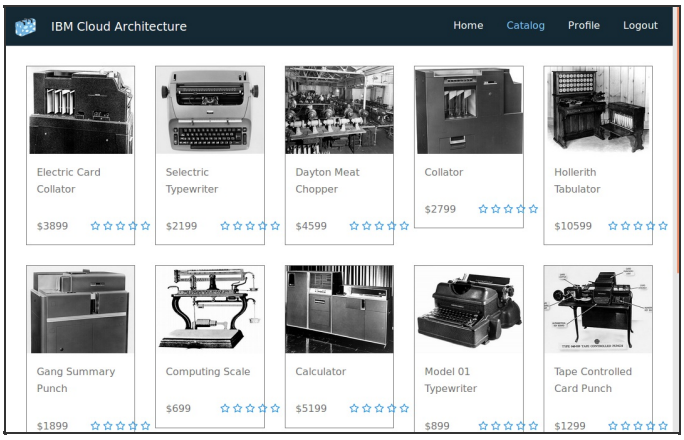
1. Open the URL to get to the BlueCompute web application: `http://proxy/bluecompute`



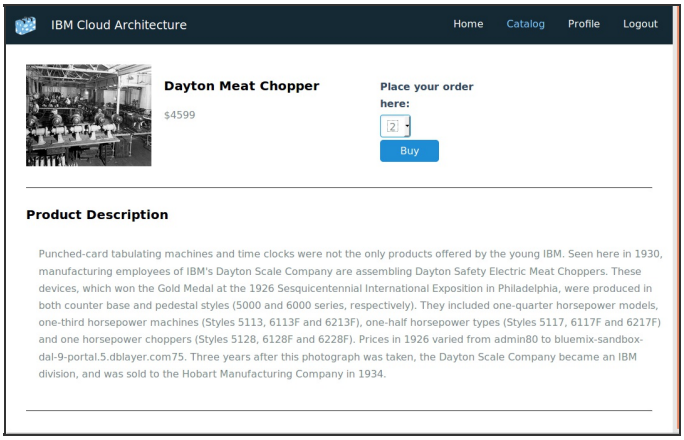
2. Click the `Log in` link and log in with as user `foo` and a password of `bar`.



3. After login, the Catalog view is shown. This verifies that the `catalog` microservice is working.

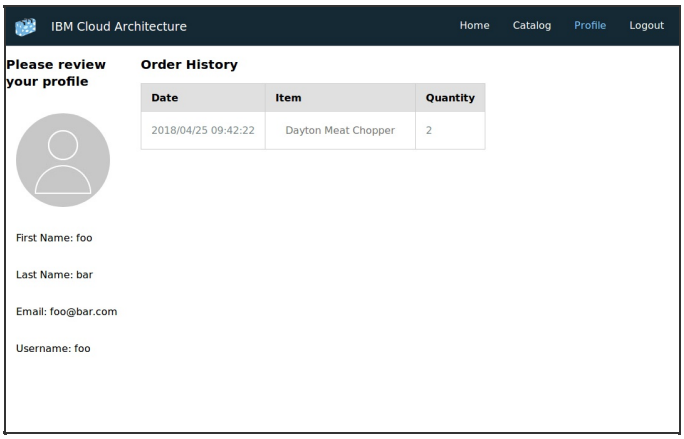


4. Select an item. The example below shows the Dayton Meat Chopper.



5. From the **Place your order here** dropdown, select a number and click **Buy**. This checks whether the `Orders` microservice is working. You should be able to retrieve your orders in the **Profile** page.

6. Click the **Profile** link. Verify that it can retrieve your user profile and the orders you have placed.



*** End of exercises ***

Working with istio

This document lists the exercises for installing and using istio 1.0.0. The units for this exercise are:

- Installing istio
- Activating bookinfo application
- Working with mixer configurations
- Configuring istio networking
- Configuring external service
- Working with RBAC definitions

Installing istio

This exercise assumes that you have already:

- Installed the appropriate CLI for your Kubernetes environment
- Installed and initialized your environment for `helm` command, including having the `tiller` installed
- Setup `kubectl` command line for accessing your cluster

These steps will install istio 1.0.0 into your Kubernetes environment:

1. Login and setup the environment for working with you cluster.
    - Use the `ibmcloud pr login` command to login to IBM Cloud Private CLI using `admin` / `passw0rd` and select the mycluster Account.

    - Login to IBM Kubernetes Services using `ibmcloud login` command and use the `ibmcloud ks cluster-config <clustername>` to set the KUBECONFIG environment variable

2. Download the istio zip file for some definitions and executables that are needed to setup istio later. (**Note**: This download is only valid if you are using a Linux client, for other platform, use the download from `https://istio.io` page. The filename and path may differ slightly.)

```
 docker run --rm -v /home/localuser:/home ibmcom/istioctl:1.0.0 tar
-xzf /root/istio-1.0.0.tar.gz -C /home
```

3. If you are usig IBM Cloud Private below 3.1, setup the Kubernetes cluster for istio (this is only needed for helm below 2.10).

```
 kubectl apply -f istio-
1.0.0/install/kubernetes/helm/istio/templates/crds.yaml
```

```
localuser@ibmcloudacademy:~/istio-chart-master/ibm-istio/templates$ kubectl apply -f crds.yaml
customresourcedefinition "virtualservices.networking.istio.io" created
customresourcedefinition "destinationrules.networking.istio.io" created
customresourcedefinition "serviceentries.networking.istio.io" created
customresourcedefinition "gateways.networking.istio.io" created
customresourcedefinition "envoyfilters.networking.istio.io" created
customresourcedefinition "policies.authentication.istio.io" created
customresourcedefinition "meshpolicies.authentication.istio.io" created
customresourcedefinition "httpapispecbindings.config.istio.io" created
customresourcedefinition "httpapispecs.config.istio.io" created
customresourcedefinition "quotaspecbindings.config.istio.io" created
customresourcedefinition "quotaspecs.config.istio.io" created
customresourcedefinition "rules.config.istio.io" created
customresourcedefinition "attributemanifests.config.istio.io" created
customresourcedefinition "bypasses.config.istio.io" created
customresourcedefinition "circonuses.config.istio.io" created
customresourcedefinition "deniers.config.istio.io" created
customresourcedefinition "fluentds.config.istio.io" created
customresourcedefinition "kubernetesenvs.config.istio.io" created
customresourcedefinition "listcheckers.config.istio.io" created
customresourcedefinition "memquotas.config.istio.io" created
customresourcedefinition "noops.config.istio.io" created
customresourcedefinition "opas.config.istio.io" created
customresourcedefinition "prometheuses.config.istio.io" created
customresourcedefinition "rbacs.config.istio.io" created
customresourcedefinition "redisquotas.config.istio.io" created
customresourcedefinition "servicecontrols.config.istio.io" created
customresourcedefinition "signalfxs.config.istio.io" created
customresourcedefinition "solarwindses.config.istio.io" created
customresourcedefinition "stackdrivers.config.istio.io" created
customresourcedefinition "statsds.config.istio.io" created
customresourcedefinition "stdios.config.istio.io" created
customresourcedefinition "apikeys.config.istio.io" created
customresourcedefinition "authorizations.config.istio.io" created
customresourcedefinition "checknothings.config.istio.io" created
customresourcedefinition "kuberneteses.config.istio.io" created
customresourcedefinition "listentries.config.istio.io" created
customresourcedefinition "logentries.config.istio.io" created
customresourcedefinition "edges.config.istio.io" created
customresourcedefinition "metrics.config.istio.io" created
customresourcedefinition "quotas.config.istio.io" created
customresourcedefinition "reportnothings.config.istio.io" created
customresourcedefinition "servicecontrolreports.config.istio.io" created
customresourcedefinition "tracespans.config.istio.io" created
customresourcedefinition "rbacconfigs.rbac.istio.io" created
customresourcedefinition "serviceroles.rbac.istio.io" created
customresourcedefinition "servicerolebindings.rbac.istio.io" created
customresourcedefinition "adapters.config.istio.io" created
customresourcedefinition "instances.config.istio.io" created
customresourcedefinition "templates.config.istio.io" created
customresourcedefinition "handlers.config.istio.io" created
localuser@ibmcloudacademy:~/istio-chart-master/ibm-istio/templates$ cd ../charts/certmanager/te
mplates/
localuser@ibmcloudacademy:~/istio-chart-master/ibm-istio/charts/certmanager/templates$ kubectl
apply -f crds.yaml
customresourcedefinition "clusterissuers.certmanager.k8s.io" created
customresourcedefinition "issuers.certmanager.k8s.io" created
customresourcedefinition "certificates.certmanager.k8s.io" created
localuser@ibmcloudacademy:~/istio-chart-master/ibm-istio/charts/certmanager/templates$
```

4. Create the istio-system namespace.

```
kubectl create namespace istio-system
```

5. Setup istio helm chart repository, we are using the IBM flavor of the helm chart.

```
helm repo add ibm-charts
https://raw.githubusercontent.com/IBM/charts/master/repo/stable/
```

6. Deploy istio using helm, enable the tracing and grafana as you will use them in this exercise. (If you are using IBM Kubernetes Services, you should omit the `--tls` option)

```
helm install ibm-charts/ibm-istio --tls --name istio --namespace
istio-system --set tracing.enabled=true --set grafana.enabled=true
--set pilot.resource.request.cpu=100m --set
pilot.resource.request.memory=1024Mi
```

7. Check result of the deployment.

```
kubectl get pod -n istio-system
```

Make sure that the istio system pods are running.

```
localuser@ibmcloudacademy:~/istio-chart-master$ kubectl get pod  -n istio-system
NAME                                      READY     STATUS    RESTARTS   AGE
grafana-64b7b844cc-9frz8                  1/1       Running   0          4h
istio-citadel-c8fb4f667-6vftc             1/1       Running   0          5h
istio-egressgateway-f64f49d9c-4gv4n       1/1       Running   0          5h
istio-galley-57b749c55d-vd5w8             1/1       Running   0          5h
istio-ingressgateway-57d6c54b44-wkh7t     1/1       Running   0          5h
istio-pilot-59dd4576c6-h7v9d              2/2       Running   0          4h
istio-policy-5465f45b49-8vvn2             2/2       Running   0          5h
istio-sidecar-injector-74cb6c675f-gd45n   1/1       Running   0          5h
istio-statsd-prom-bridge-77754cdb7b-47qxj 1/1       Running   0          5h
istio-telemetry-69c7b9d67b-4bf6l          2/2       Running   0          2m
istio-telemetry-69c7b9d67b-wtpc2          2/2       Running   0          5h
istio-tracing-5fbd6f6ddb-ksnsd            1/1       Running   0          5h
prometheus-94794746c-2vnql                1/1       Running   0          5h
```

8. Check the created services for istio:

```
kubectl get service -n istio-system
```

```
localuser@ibmcloudacademy:~/istio-chart-master$ kubectl get service -n istio-system
NAME                       TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)
                                                                     AGE
grafana                    ClusterIP      10.0.0.15    <none>        3000/TCP
                                                                     5h
istio-citadel              ClusterIP      10.0.0.129   <none>        8060/TCP,9093/TCP
                                                                     5h
istio-egressgateway        ClusterIP      10.0.0.54    <none>        80/TCP,443/TCP
                                                                     5h
istio-galley               ClusterIP      10.0.0.178   <none>        443/TCP,9093/TCP
                                                                     5h
istio-ingressgateway       LoadBalancer   10.0.0.60    <pending>     80:31380/TCP,443:31390/TCP,31400:31400/
TCP,15011:30164/TCP,8060:30485/TCP,15030:32469/TCP,15031:32018/TCP   5h
istio-pilot                ClusterIP      10.0.0.101   <none>        15010/TCP,15011/TCP,8080/TCP,9093/TCP
                                                                     5h
istio-policy               ClusterIP      10.0.0.85    <none>        9091/TCP,15004/TCP,9093/TCP
                                                                     5h
istio-sidecar-injector     ClusterIP      10.0.0.244   <none>        443/TCP
                                                                     5h
istio-statsd-prom-bridge   ClusterIP      10.0.0.252   <none>        9102/TCP,9125/UDP
                                                                     5h
istio-telemetry            ClusterIP      10.0.0.22    <none>        9091/TCP,15004/TCP,9093/TCP,42422/TCP
                                                                     5h
jaeger-agent               ClusterIP      None         <none>        5775/UDP,6831/UDP,6832/UDP
                                                                     5h
jaeger-collector           ClusterIP      10.0.0.48    <none>        14267/TCP,14268/TCP
                                                                     5h
jaeger-query               ClusterIP      10.0.0.25    <none>        16686/TCP
                                                                     5h
prometheus                 ClusterIP      10.0.0.148   <none>        9090/TCP
                                                                     5h
tracing                    ClusterIP      10.0.0.26    <none>        16686/TCP
                                                                     5h
zipkin                     ClusterIP      10.0.0.177   <none>        9411/TCP
                                                                     5h
```

What ports are mapped to the istio-ingressgateway?

- Port 80: _____
- Port 443: _____

9. Add `istioctl` to program path

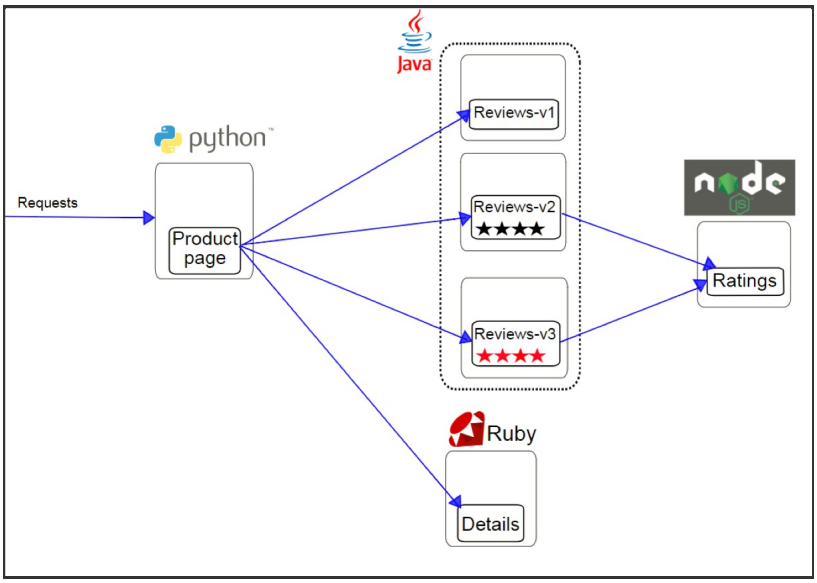
```
sudo mv ~/istio-1.0.0/bin/istioctl /usr/local/bin/istioctl
```

Check it using the command `istioctl version`.

```
localuser@ibmcloudacademy:~$ istioctl version
Version: 1.0.0
GitRevision: 3a136c90ec5e308f236e0d7ebb5c4c5e405217f4
User: root@71a9470ea93c
Hub: gcr.io/istio-release
GolangVersion: go1.10.1
BuildStatus: Clean
```

Activating bookinfo application

We will use the bookinfo sample application. The application structure is as shown here:



1. The bookinfo sample application comes with the istio distribution, here are the steps to install it.

```
cd ~/istio-1.0.0
kubectl create -f <(istioctl kube-inject -f
samples/bookinfo/platform/kube/bookinfo.yaml)
```

```
localuser@ibmcloudacademy:~/istio-1.0.0$ kubectl create -f <(istioctl kube-inject
 -f samples/bookinfo/platform/kube/bookinfo.yaml)
service "details" created
deployment "details-v1" created
service "ratings" created
deployment "ratings-v1" created
service "reviews" created
deployment "reviews-v1" created
deployment "reviews-v2" created
deployment "reviews-v3" created
service "productpage" created
deployment "productpage-v1" created
```

2. Check the deployed pods:

```
kubectl get pod
```

Make sure that the pods are running before continuing with the lab.

```
localuser@ibmcloudacademy:~$ kubectl get pod
NAME                              READY     STATUS    RESTARTS   AGE
details-v1-558d5fc956-95qzx       2/2       Running   0          31m
productpage-v1-576c55ddb8-qnz26   2/2       Running   0          31m
ratings-v1-556c44f648-tbqtz       2/2       Running   0          31m
reviews-v1-5ff97b656b-rkzcf       2/2       Running   0          31m
reviews-v2-948df8f54-lr9dw        2/2       Running   0          31m
reviews-v3-f9b6c94f8-5xxqs        2/2       Running   0          31m
```

3. Check whether the istio-proxy sidecar are running with the pod. As you deploy them in a single command, it would be sufficient to check only one of them. This example check the deployment of the productpage pod.

```
kubectl describe pod $(kubectl get pod | grep productpage | awk '{print $1}')
```

4. Check the services created and note the port for the productpage.

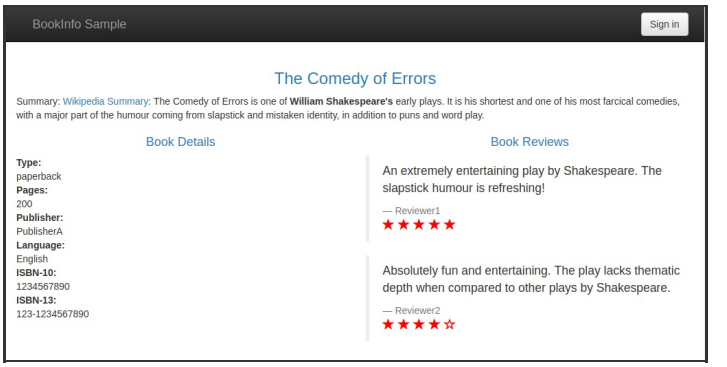
```
kubectl get services
```

```
localuser@ibmcloudacademy:~$ kubectl get services
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
details       ClusterIP   10.0.0.46     <none>        9080/TCP   38m
kubernetes    ClusterIP   10.0.0.1      <none>        443/TCP    11d
productpage   ClusterIP   10.0.0.159    <none>        9080/TCP   38m
ratings       ClusterIP   10.0.0.108    <none>        9080/TCP   38m
reviews       ClusterIP   10.0.0.113    <none>        9080/TCP   38m
```

5. To access the application without istio, you can create a port-forwarding with `kubectl`

```
kubectl port-forward $(kubectl get pod | grep productpage | awk '{print $1}') 9000:9080
```

6. Make sure that you can access the forwarded port using a Web browser to `http://localhost:9000/productpage`; try to refresh the page several time, you should be able to see that the reviews part has 3 different flavors (representing the different versions of the reviews application which all are wired to the same service). Refresh the pages several times.



7. Stop the port-forward command using `Ctrl-C`.
8. Try accessing the details page using curl and port-forward.
    - Run the port-forwarding for the details pod:

```
kubectl port-forward <details-pod> 9080:9080
```

    - Run curl command to get the details:

```
curl http://localhost:9080/details/0
```

    - Stop the port-forwarding using `Ctrl-C`

Now the bookinfo application is running without any management function from istio. You did install the istio sidecar with the pod, however, as there are no management directives yet it is not performing anything.

Work with mixer definitions

Start working with mixer definition.

1. Mixer definitions are defined against the istio-system namespace. The

Out-of-the-box installation of istio already have the necessary metric collected. This first part of the exercise would check this collection.

- What feed into the grafana dashboard?

  _____

- What istio record types needed to create that feed?

  _____

- How can you check that those feeds exist?

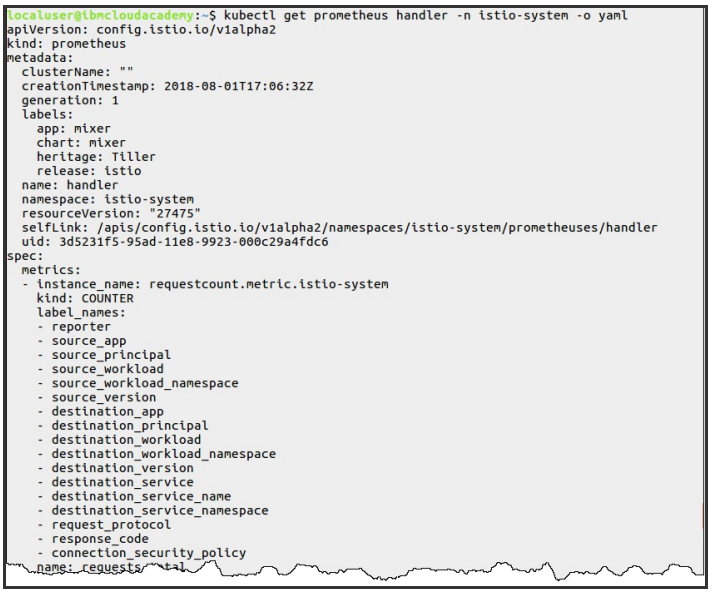
  _____

2. Perform the following commands:

```
kubectl get prometheus -n istio-system
```

Use the name field from above:

```
kubectl get prometheus <name> -n istio-system -o yaml
```



Those metrics and collections are the one to be loaded to prometheus, but what generates the metrics?

3. Perform the following commands:

```
kubectl get rule -n istio-system
```

- How many rules do you get? _____
- Which rules do you think is targeting the prometheus handler? _____

4. Check the promhttp rule using the command:

```
kubectl get rule promhttp -n istio-system -o yaml
```

What metrices are collected from into the prometheus handler?

_____

```
localuser@ibmcloudacademy:~$ kubectl get rule promhttp -n istio-system -o yaml
apiVersion: config.istio.io/v1alpha2
kind: rule
metadata:
  clusterName: ""
  creationTimestamp: 2018-08-01T17:06:32Z
  generation: 1
  labels:
    app: mixer
    chart: mixer
    heritage: Tiller
    release: istio
  name: promhttp
  namespace: istio-system
  resourceVersion: "27479"
  selfLink: /apis/config.istio.io/v1alpha2/namespaces/istio-system/rules/promhttp
  uid: 3d5665d9-95ad-11e8-9923-000c29a4fdc6
spec:
  actions:
  - handler: handler.prometheus
    instances:
    - requestcount.metric
    - requestduration.metric
    - requestsize.metric
    - responsesize.metric
  match: context.protocol == "http" || context.protocol == "grpc"
```

5. Lets look at the first instance, the requestcount.

```
kubectl get metric requestcount -n istio-system -o yaml
```

Does the collected metrics matched with the loaded items?

_____

```
localuser@ibmcloudacademy:~$ kubectl get metric requestcount -n istio-system -o yaml
apiVersion: config.istio.io/v1alpha2
kind: metric
metadata:
  clusterName: ""
  creationTimestamp: 2018-08-01T17:06:32Z
  generation: 1
  labels:
    app: mixer
    chart: mixer
    heritage: Tiller
    release: istio
  name: requestcount
  namespace: istio-system
  resourceVersion: "27474"
  selfLink: /apis/config.istio.io/v1alpha2/namespaces/istio-system/metrics/requestcount
  uid: 3d50d8c5-95ad-11e8-9923-000c29a4fdc6
spec:
  dimensions:
    connection_security_policy: conditional((context.reporter.kind | "inbound") ==
      "outbound", "unknown", conditional(connection.mtls | false, "mutual_tls", "none"))
    destination_app: destination.labels["app"] | "unknown"
    destination_principal: destination.principal | "unknown"
    destination_service: destination.service.host | "unknown"
    destination_service_name: destination.service.name | "unknown"
    destination_service_namespace: destination.service.namespace | "unknown"
    destination_version: destination.labels["version"] | "unknown"
    destination_workload: destination.workload.name | "unknown"
    destination_workload_namespace: destination.workload.namespace | "unknown"
    reporter: conditional((context.reporter.kind | "inbound") == "outbound", "source",
      "destination")
    request_protocol: api.protocol | context.protocol | "unknown"
    response_code: response.code | 200
    source_app: source.labels["app"] | "unknown"
    source_principal: source.principal | "unknown"
    source_version: source.labels["version"] | "unknown"
    source_workload: source.workload.name | "unknown"
    source_workload_namespace: source.workload.namespace | "unknown"
  monitored_resource_type: '"UNSPECIFIED"'
  value: "1"
```

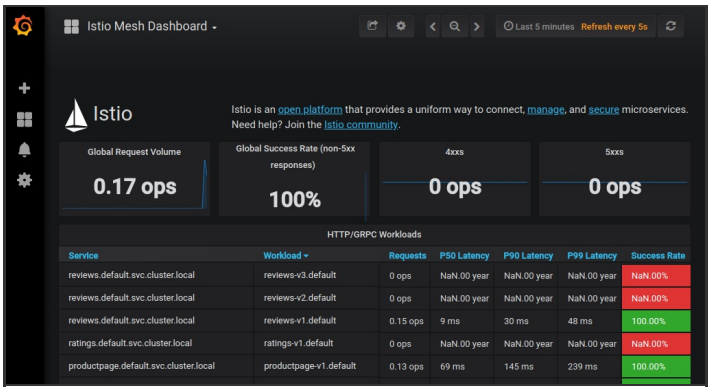
6. Check grafana dashboards, first lets do the port forwarding to allow access to grafana. Get the grafana pod name and port number:

```
kubectl get pod -n istio-system | grep grafana
kubectl get service -n istio-system | grep grafana
```

Use the pod name and port number to run port forwarding:

```
kubectl port-forward <podname> -n istio-system 10000:<portnumber>
```

7. Check the grafana dashboard at `http://localhost:10000`. Navigate through the various dashboard it has (you can change dashboard using the  icon on the top left.



Stop the port forwarding when you are done.

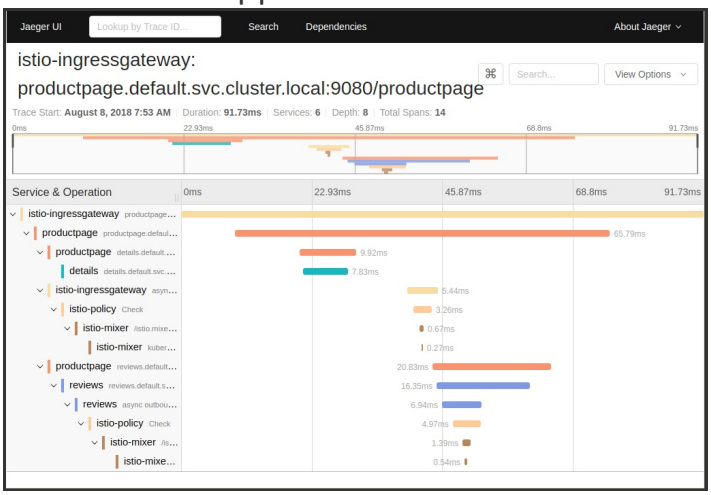
8. Look at the tracing data, similar to the grafana, get the pod name and port number.

```
kubectl get pod -n istio-system | grep tracing
kubectl get service -n istio-system | grep tracing
```

Use the pod name and port number to run port forwarding:

```
kubectl port-forward <podname> -n istio-system 10000:<portnumber>
```

9. Open the jaeger dashboard at `http://localhost:10000`. Select the service `productpage` and then click **Find Traces** at the bottom of the left pane. It will show a bubble chart for the found instances. click on one of the instance. You can see the time progression of your microservice application.



Navigate and expand some of the components to understand the use of this application. Also note that the istio overhead are shown for the istio-policy and istio-mixer is quite small.

Configuring istio networking

The application is currently running using the internal service in Kubernetes, there is no real direct access to the application. This section creates the necessary istio resources for bookinfo.

Note: You can download most of the yaml files for the rest of the exercises from `git clone https://github.com/ibm-cloud-academy/istio-yaml`

1. Create a definition file called bookinfo-gw.yaml which contains a

Gateway object that is using the default istio-ingressgateway; and serve port 80.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gw
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

2. Create another definition called bookinfo-vs.yaml which contains a virtual service object that is a member of bookinfo-gw. The application would accept the paths `/productpage`, `/login` and `/logoff`. The virtual service refers to the productpage host in port 9080.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gw
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        exact: /login
    - uri:
        exact: /logout
    route:
    - destination:
        host: productpage
        port:
          number: 9080
```

3. Activate both definitions:

```
istioctl create -f bookinfo-gw.yaml
istioctl create -f bookinfo-vs.yaml
```

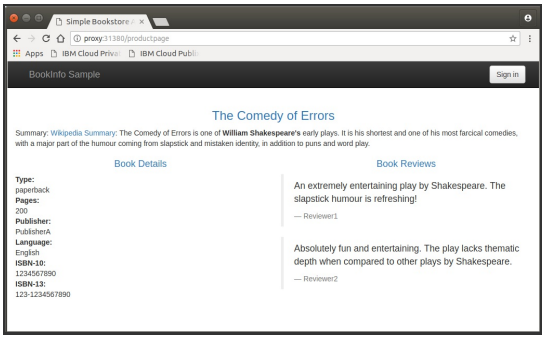
```
localuser@ibmcloudacademy:~$ vi bookinfo-gw.yaml
localuser@ibmcloudacademy:~$ vi bookinfo-vs.yaml
localuser@ibmcloudacademy:~$ istioctl create -f bookinfo-gw.yaml
Created config gateway/default/bookinfo-gw at revision 38888
localuser@ibmcloudacademy:~$ istioctl create -f bookinfo-vs.yaml
Created config virtual-service/default/bookinfo-vs at revision 38900
localuser@ibmcloudacademy:~$
```

Note: that once the definition is created, when you must modify the yaml file, you can run `istioctl replace -f` command.

4. Try to access the bookinfo application, remember the port mapping for http for istio-ingressgateway service.

```
http://proxy:<ingressport>/productpage
```

    1. You should get the same page as the one from the port-forward command.



    2. Try to **Sign in** using user `foo` with the password of `bar`

    3. Try signing out

5. Since you have not activate any security or other networking definitions, the application behaves exactly as it would behave with just Kubernetes implementation. These next steps introduce different actions that you can do in the networking sense.

6. Create definition (Virtual Service and Destination Rule) for the Reviews page. The first few scenarios would use this `reviews-dr.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
```

```
kind: DestinationRule
metadata:
  name: reviews-dr
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  - name: v3
    labels:
      version: v3
```

1. Set that only reviews-v1 would show in the productpage every time:
   `reviews-vs.yaml`

   ```
   apiVersion: networking.istio.io/v1alpha3
   kind: VirtualService
   metadata:
     name: reviews-vs
   spec:
     hosts:
     - reviews
     http:
     - route:
       - destination:
           host: reviews
           subset: v1
   ```

   Activate the definition using:

   `istioctl create -f reviews-vs.yaml`

   `istioctl create -f reviews-dr.yaml`

   Check the product page that now the reviews never show the star rating information, even when you refresh it multiple times.

2. Modify the definitions to only show reviews-v2; This modification is performed on reviews-vs.yaml to have the `subset: v2`. Apply to definition:

`istioctl replace -f reviews-vs.yaml`

Check that now the ratings is shown with black stars.

3. Modify and perform traffic splitting so that it will show 50% of reviews-v2 and 50% of reviews-v3. Modify reviews-vs.yaml to include the following route definition:

```
http:
      - route:
        - destination:
            host: reviews
            subset: v2
          weight: 50
        - destination:
            host: reviews
            subset: v3
          weight: 50
```

Check again in the productpage, that the ratings shows with black or red stars alternately.

4. Perform traffic steering and set that only user `foo` will be sent to reviews-v3 and all other users to reviews-v2 while if the user is not logged in, go to reviews-v1. Change the reviews-vs.yaml to include the following route definition.

```
http:
      - match:
        - headers:
            end-user:
              exact: foo
        route:
        - destination:
            host: reviews
            subset: v3
      - match:
        - headers:
            end-user:
              regex: ".*\\S+.*"
        route:
        - destination:
            host: reviews
            subset: v2
```

```
      - route:
        - destination:
            host: reviews
            subset: v1
```

Test the definition using various user names in the productpage; there is no restriction of the user or password that you can use for the page.

5. Remove all the conditional routing and introduce latency for the user foo for 5 seconds. The reviews-vs.yaml rules becomes:

```
http:
      - match:
        - headers:
            end-user:
              exact: foo
        fault:
          delay:
            percent: 100
            fixedDelay: 5s
        route:
        - destination:
            host: reviews
            subset: v2
      - route:
        - destination:
            host: reviews
            subset: v3
```

6. Change from the delay into returning error 501 for foo to get reviews:

```
http:
      - match:
        - headers:
            end-user:
              exact: foo
        fault:
          abort:
            percent: 100
            httpStatus: 501
        route:
        - destination:
            host: reviews
```

```
                subset: v2
          - route:
            - destination:
                host: reviews
                subset: v3
```

7. Reset the all the fault injection on the reviews-vs.yaml file and maintain traffic steering for user foo to go to v2 subset:

```
    http:
          - match:
            - headers:
                end-user:
                  exact: foo
            route:
            - destination:
                host: reviews
                subset: v2
          - route:
            - destination:
                host: reviews
                subset: v3
```

   Test to make sure that all fault injection has been removed and the steering works as expected.

7. Add ratings virtualservice and destinationrule with a 3sec latency for foo.

   - ratings-vs.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings-vs
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: foo
    fault:
      delay:
```

```
            percent: 100
            fixedDelay: 3s
        route:
        - destination:
            host: ratings
            subset: v1
      - route:
        - destination:
            host: ratings
            subset: v1
```

- ratings-dr.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ratings-dr
spec:
  host: ratings
  subsets:
  - name: v1
    labels:
      version: v1
```

Test the productpage and verify that the delays only happen for the user foo.

8. In the reviews-dr.yaml, define a circuit breaker that is only have a single connection in the connectionPool. Set the outlierDetection to test every 5 seconds and for any error, eject the node for 5 minutes.

```
 trafficPolicy:
   connectionPool:
     http:
       http1MaxPendingRequests: 1
       maxRequestsPerConnection: 1
     tcp:
       maxConnections: 1
   outlierDetection:
     baseEjectionTime: 5m
     consecutiveErrors: 1
     interval: 5s
     maxEjectionPercent: 100
```

- With single connection, when you submit multiple

requests in successions (use a couple of browser tabs); the first few requests would return after a couple of seconds. However, afterwards the request would just be failed quickly; which indicates that the circuit breaker in the reviews process is tripped.

- After the ejection time has passed, try to load the product page again to check whether it returns to the slow response time again.
- Remove the traffic policy from reviews-dr.yaml and remove the fault from ratings-vs.yaml, reapply the files using `istioctl replace -f <file>`.

9. Finish creating the VirtualServices and DestinationRules for the details and productpage components as follows, you will need these definition for the RBAC testing later.

    - productpage-vs.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: productpage-vs
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1
```

    - productpage-dr.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: productpage-dr
spec:
  host: productpage
  subsets:
  - name: v1
    labels:
      version: v1
```

- details-vs.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: details-vs
spec:
  hosts:
  - details
  http:
  - route:
    - destination:
        host: details
        subset: v1
```

- details-dr.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: details-dr
spec:
  host: details
  subsets:
  - name: v1
    labels:
      version: v1
```

Load these definitions using `istioctl create -f` command.

Working with external service

The external service that you are working with is represented with a simple curl command. This method simplify the testing without the need to build a specialized application that access an external API endpoint.

1. Deploy the sleep pod:

```
 cd istio-1.0.0
 kubectl apply -f <(istioctl kube-inject -f
samples/sleep/sleep.yaml)
```

Check that the pod get deployed correctly using

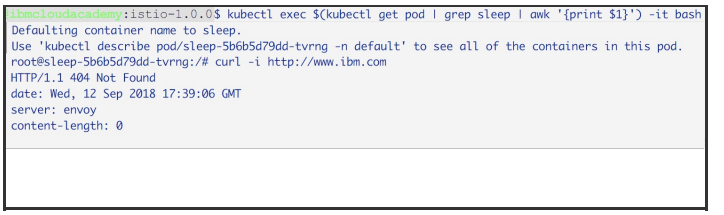
`kubectl get pod | grep sleep`.

2. Open another terminal session to connect into the sleep pod:

```
 kubectl exec $(kubectl get pod | grep sleep | awk '{print $1}') -
it bash
```

3. Check whether you can access any external web-site from the sleep pod, lets use `http://www.google.com`

```
curl -i http://www.ibm.com
```



You notice that it cannot find it and envoy is the one that returns 404 code.

4. Create a serviceEntry for www.ibm.com in googlese.yaml.

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: ibm-se
spec:
  hosts:
  - www.ibm.com
  ports:
  - number: 80
    name: http
    protocol: http
  resolution: DNS
  location: MESH_EXTERNAL
```

Apply it using `istioctl create -f ibmse.yaml`

- Test from the bash session your connection to www.google.com:

```
curl -i http://www.ibm.com
```

You should get some output and a HTTP response of 301, which redirect to the secure https site.

- Now try to connect to the secure https://www.ibm.com

```
curl -i https://www.ibm.com
```

You should got an error indicating SSL certificate problem.

5. Create a VirtualService to route to www.google.com using tls in file called googlevs.yaml.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ibm-vs
spec:
  hosts:
  - www.ibm.com
  tls:
  - match:
    - port: 443
      sni_hosts:
      - www.ibm.com
    route:
    - destination:
        host: www.ibm.com
        port:
          number: 443
      weight: 100
```

Activate using the command `istioctl create -f ibmvs.yaml` and test the `curl -i https://www.ibm.com` to make sure that you can connect to the destination. In the testing that we performed, it redirects you (HTTP 303) to another page, but the communication works.

```
root@sleep-5b6b5d79dd-tvrng:/# curl -i https://www.ibm.com
HTTP/1.1 303 See Other
Server: AkamaiGHost
Content-Length: 0
Location: https://www.ibm.com/it-en/
Date: Wed, 12 Sep 2018 22:06:42 GMT
Connection: keep-alive
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy: upgrade-insecure-requests
Strict-Transport-Security: max-age=31536000
```

Define RBAC security

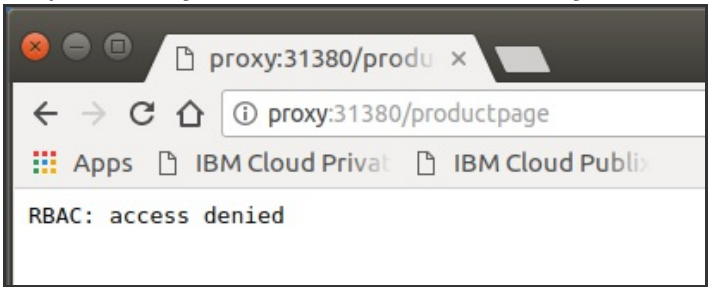
Here you will work with RBAC security for the bookinfo application.

1. Create the global RbacConfig to enable RBAC for the whole default namespace. Edit a file called rbacconfig.yaml and define the following:

```
apiVersion: rbac.istio.io/v1alpha1
kind: RbacConfig
metadata:
  name: default
spec:
  mode: 'ON_WITH_INCLUSION'
  inclusion:
    namespaces: ["default"]
```

   This definition activates RBAC in the default namespace, apply this definition: `istioctl create -f rbacconfig.yaml`

2. Check now whether you can access the productpage? It will reject the request as you have not defined any role and access to the role.

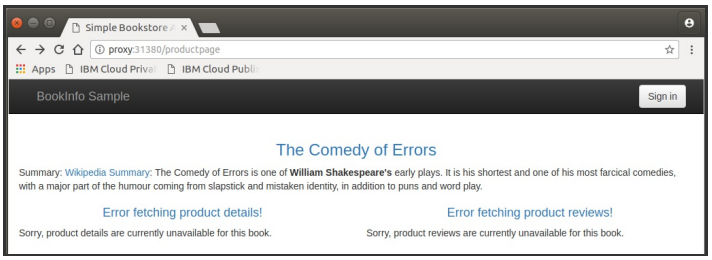


3. Create access rule for productpage viewer (GET action). Create a file called productpage-viewer.yaml as follows:

```
apiVersion: rbac.istio.io/v1alpha1
kind: ServiceRole
metadata:
```

```
    name: productpage-viewer
    namespace: default
  spec:
    rules:
    - services: ["productpage.default.svc.cluster.local"]
      methods: ["GET"]
  ---
  apiVersion: rbac.istio.io/v1alpha1
  kind: ServiceRoleBinding
  metadata:
    name: bind-productpage-viewer
    namespace: default
  spec:
    subjects:
    - user: "*"
    roleRef:
      kind: ServiceRole
      name: "productpage-viewer"
```

4. Activate the definition using the command `istioctl create -f productpage-viewer.yaml` and check whether you can access the product page. Did you see any error(s) in the page? Why? _____



5. Fix the error by allowing all access to the details and reviews page (you will restrict that later). Create the file details-viewer.yaml, ratings-viewer.yaml and reviews-viewer.yaml similar to the productpage-viewer.yaml.

```
cp productpage-viewer.yaml details-viewer.yaml
cp productpage-viewer.yaml reviews-viewer.yaml
cp productpage-viewer.yaml ratings-viewer.yaml
```

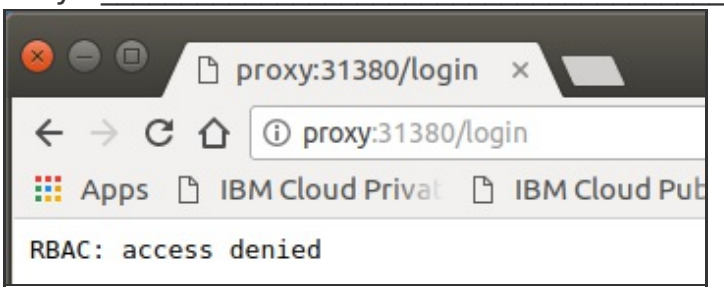
Modify the names of the object; and the service names for each of the yaml files. Activate the definitions:

```
istioctl create -f details-viewer.yaml
```

```
istioctl create -f reviews-viewer.yaml
istioctl create -f ratings-viewer.yaml
```

Verify that the product page can be accessed without error.

6. Check whether you can **Sign in** to the product page. Specify a username and click **Sign in**.
Why? _____



7. Modify the productpage-viewer.yaml to include the `POST` method for the ServiceRole record and activate it using `istioctl replace -f productpage-viewer.yaml`.

```
apiVersion: rbac.istio.io/v1alpha1
kind: ServiceRole
metadata:
  name: productpage-viewer
  namespace: default
spec:
  rules:
  - services: ["productpage.default.svc.cluster.local"]
    methods: ["GET", "POST"]
```

Check that now you should be able to **Sign in**.

8. The current status is that all permissions are wide-open. For the details-viewer.yaml to only allow access from the productpage, answer the following questions:

    - What record do you update? _____
    - What property can you use for checking access? _____
    - Can you do that now? _____

Note: You should update the ServiceRoleBinding. The ServiceRole defines the destination and the constraints that label a specific destination, while the binding defines which source can access the

destination including a specific identifying properties. The main property to identify a source is the user or source.principal; and no, you cannot do it yet as without mTLS, you do not really know which source generate the request except for its IP address.

9. Working with mutual TLS and service account. First, enable a policy record that would enable mTLS for the default namespace. Create a yaml file called default-policy.yaml:

```
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: default-namespace-policy
  namespace: default
spec:
  peers:
  - mtls: {}
```

10. Load the policy using the command `istioctl create -f default-policy.yaml`.
Can you access the product page? _____

    The product page now requires mTLS access, however, your request from the gateway has not specified any TLS attribute, hence it failed.

11. Activate mTLS in the destination rules, for all the destination rules (productpage-dr.yaml, reviews-dr.yaml, ratings-dr.yaml and details-dr.yaml), add the following stanza:

```
trafficPolicy:
  tls:
    mode: ISTIO_MUTUAL
```

    Activate the definition using `istioctl replace -f <file>` command.
    Can you access the productpage now? _____

12. Create service account for bookinfo components. Create the bookinfo-sa.yaml as follows:

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: productpage-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: details-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: reviews-sa
```

Load the definition using `kubectl create -f bookinfo-sa.yaml`; do you think we missed one service account? _____

13. Modify the serviceAccount for the bookinfo deployment units. Retrieve the definition for bookinfo deployments using the command: `kubectl get deployment -o yaml > bookinfo.yaml`

14. Edit bookinfo.yaml and add the appropriate `serviceAccountName` stanzas under `spec` just before `containers` for each of the deployments for bookinfo; use the related serviceAccounts for each components. Load the definition using the command `kubectl apply -f bookinfo.yaml`. Make sure that after the modification the application is still running.

15. Now lets add restriction on which application can access the bookinfo components. Modify the details-viewer.yaml and modify to the ServiceRoleBinding the following stanza:

```
subjects:
- user: "*"
  properties:
    source.principal: "cluster.local/ns/default/sa/details-sa"
```

Load the definition `istioctl replace -f details-viewer.yaml`. check the productpage, is it working? why? _____

Note: Which application accesses details? it is the productpage, if you modified the yaml correctly then the productpage is supposedly run using `productpage-sa` user, not `details-sa`.

16. Fix any resulting error, the source principal should be `cluster.local/ns/default/sa/productpage-sa` and replace details-viewer.yaml file. Check that you can load the productpage correctly

now.

End of exercise

IBM Training