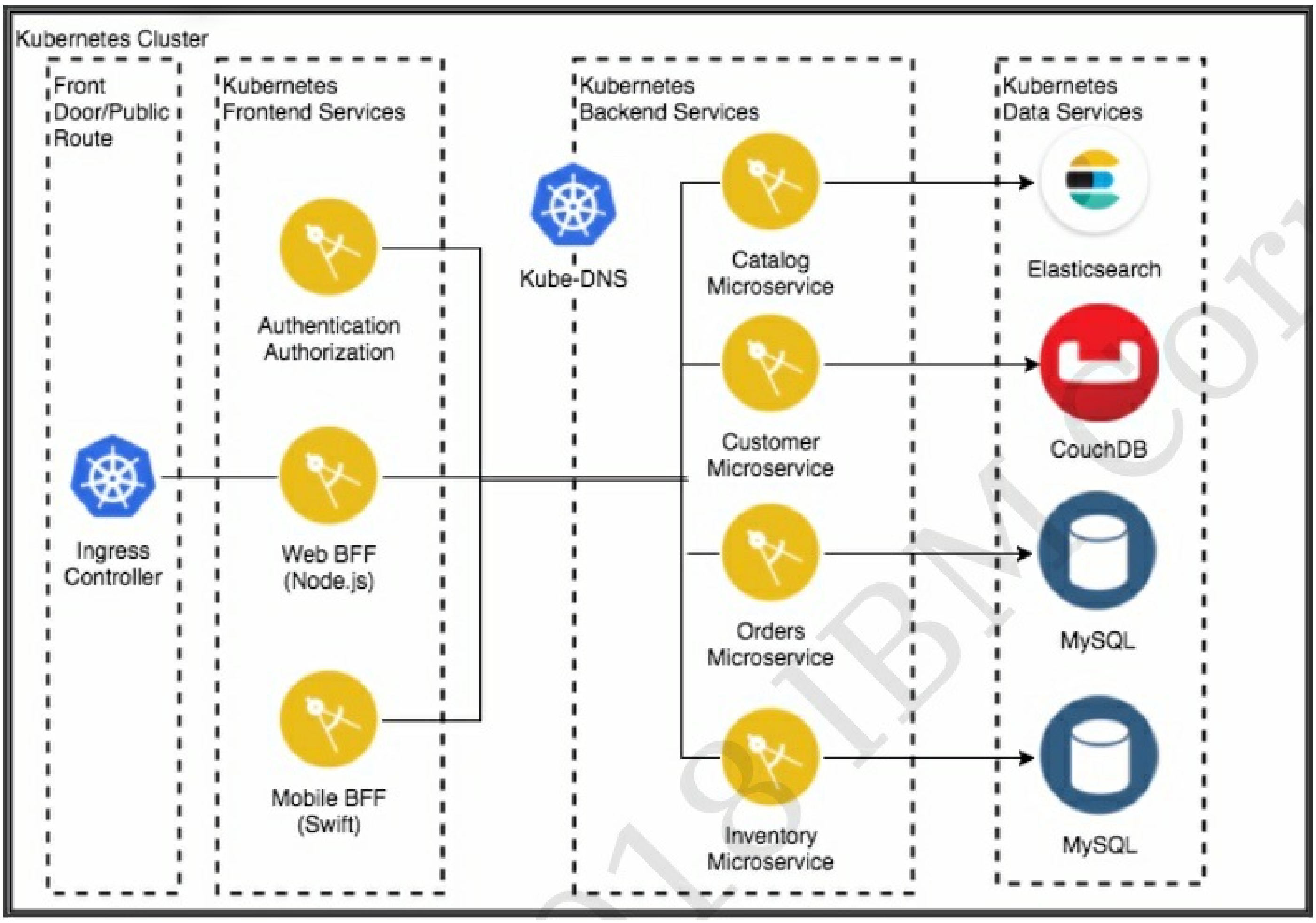
Deploying the BlueCompute Microservices Application

These exercises deploy portions of 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. The result is not a fully working application as this is just a demonstration of how to use Jenkins to build and deploy microservices.

Exercise 1: Designing the Microservices Deployment

The application is shown in the following diagram.



1. 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 start 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 one of 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. Use a terminal session on the Boot VM to define a secret to access the IBM Cloud Private user:
    - Encode the user and password that is used to connect to ICP in base64:

      `echo admin | 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: YWRtaW4K
          password: YWRtaW4K
```

    - Load the secret to ICP

      `kubectl create -f ICP_secret.yaml -n jenkins`

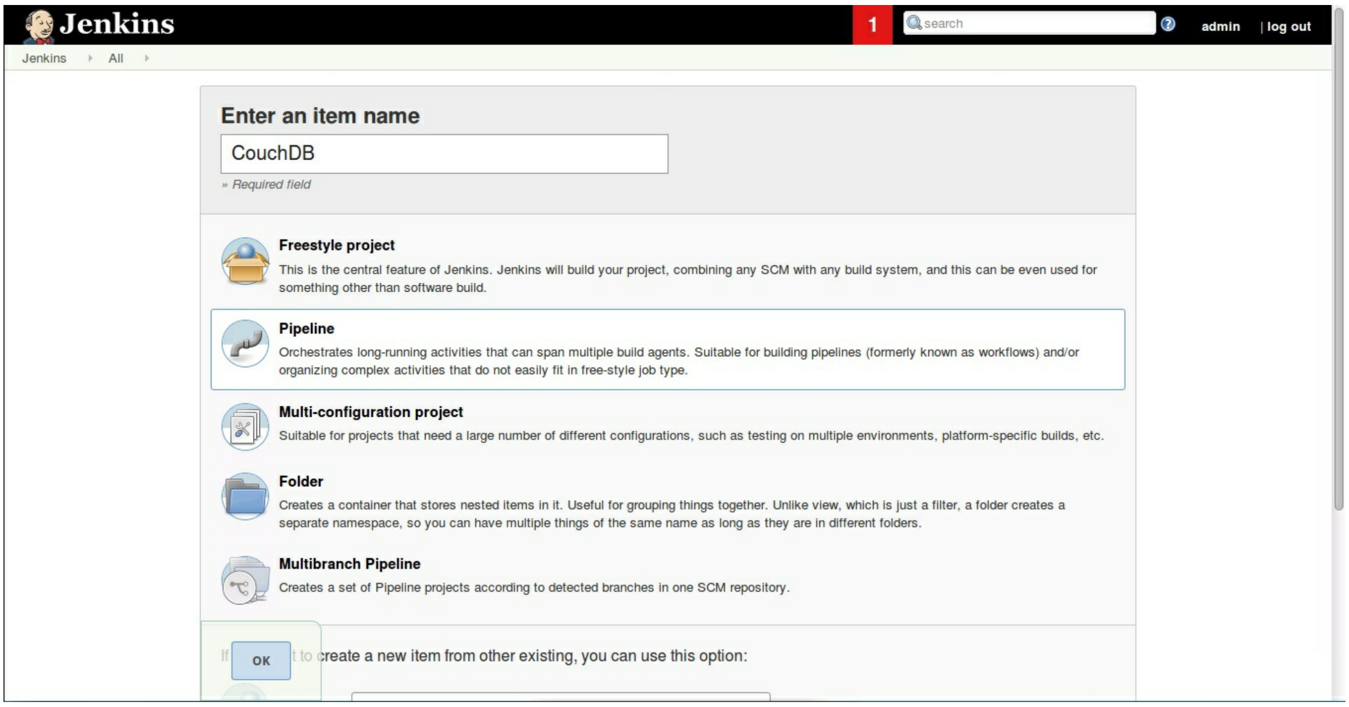
2. Set up IBM Cloud Private registry parameters as a ConfigMap (namespace, imagePullSecret and registry).
    - Create a ICP_config.yaml file in your current directory with the following contents:

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

- Load the ConfigMap to ICP

  `kubectl create -f ICP_config.yaml -n jenkins`

3. In your browser, go to the Jenkins Web UI at `http://10.10.1.4/jenkins`. Log in as `admin` with a password of `admin_0000`. You should be in the Jenkins dashboard.
4. Click New item from the Jenkins menu on the left toolbar.
5. Enter a name of `CouchDB`, select Pipeline and click OK.



Enter an item name

CouchDB

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

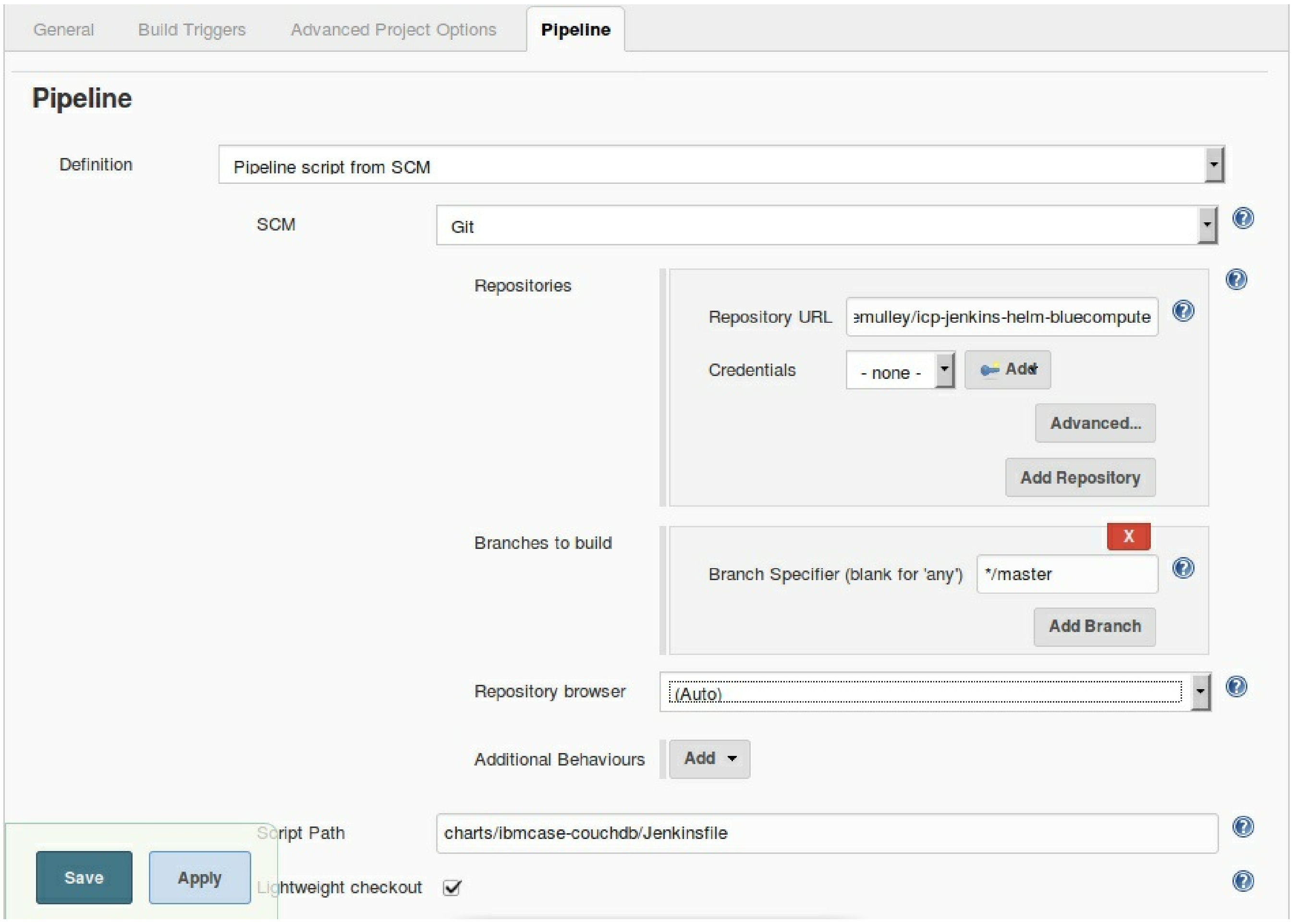
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

create a new item from other existing, you can use this option:

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


General | Build Triggers | Advanced Project Options | Pipeline

Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: ɘmulley/icp-jenkins-helm-bluecompute

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

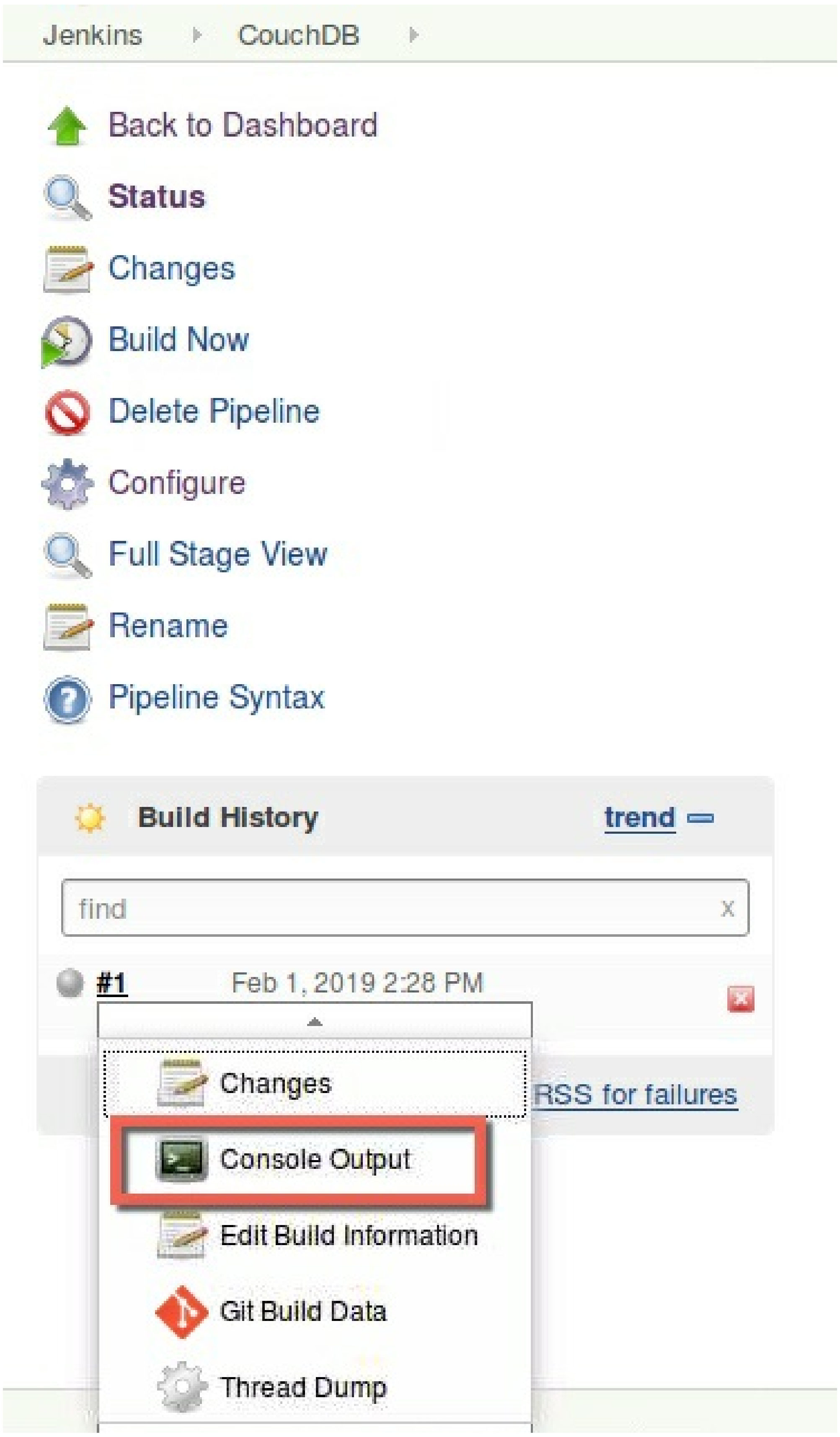
Repository browser: (Auto)

Additional Behaviours

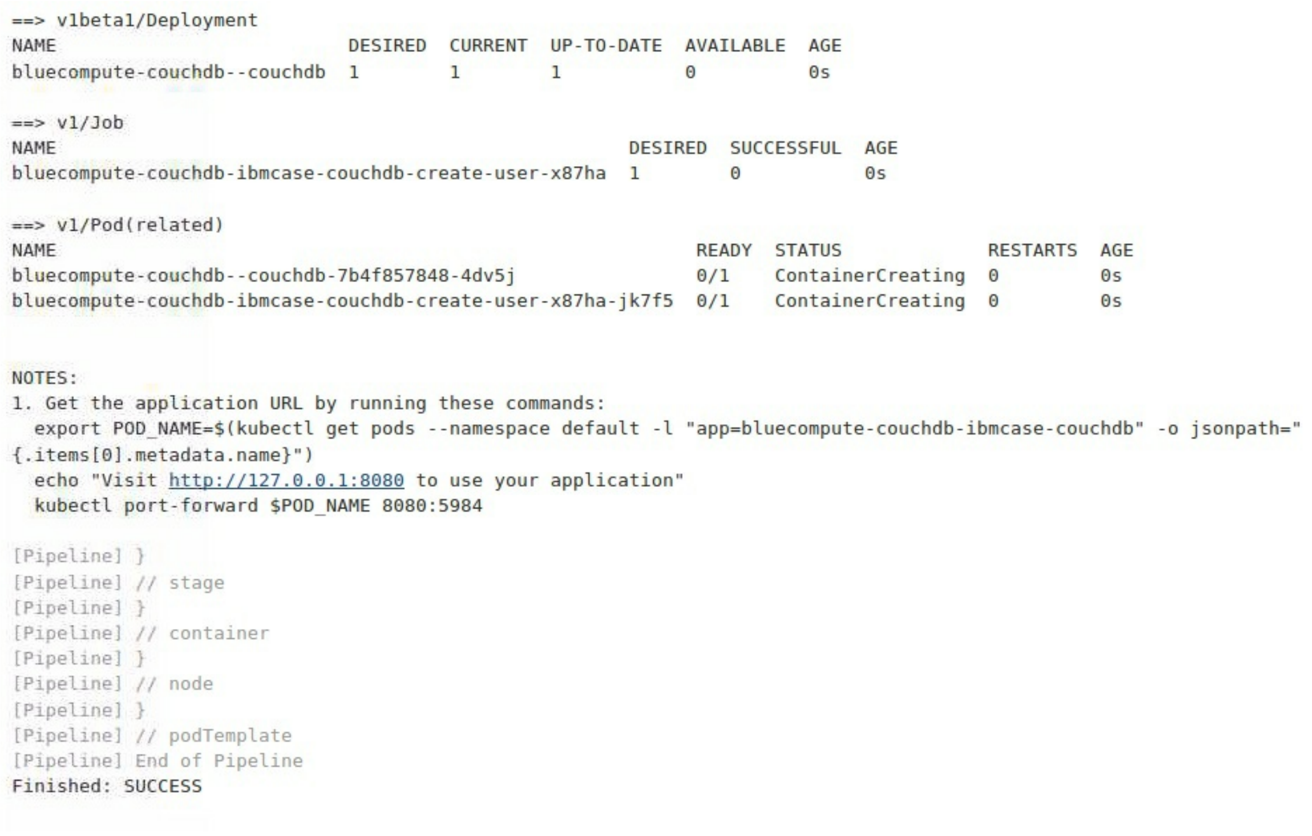
Script Path: charts/ibmcase-couchdb/Jenkinsfile

Lightweight checkout: ✓

7. On the couchDB pipeline page, click Build now.
8. 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.

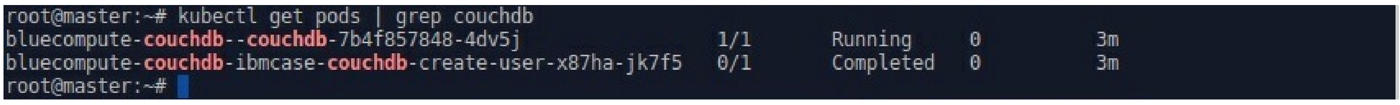


9. The console should show the helm chart being deployed at the end and the pipeline having finished successfully.



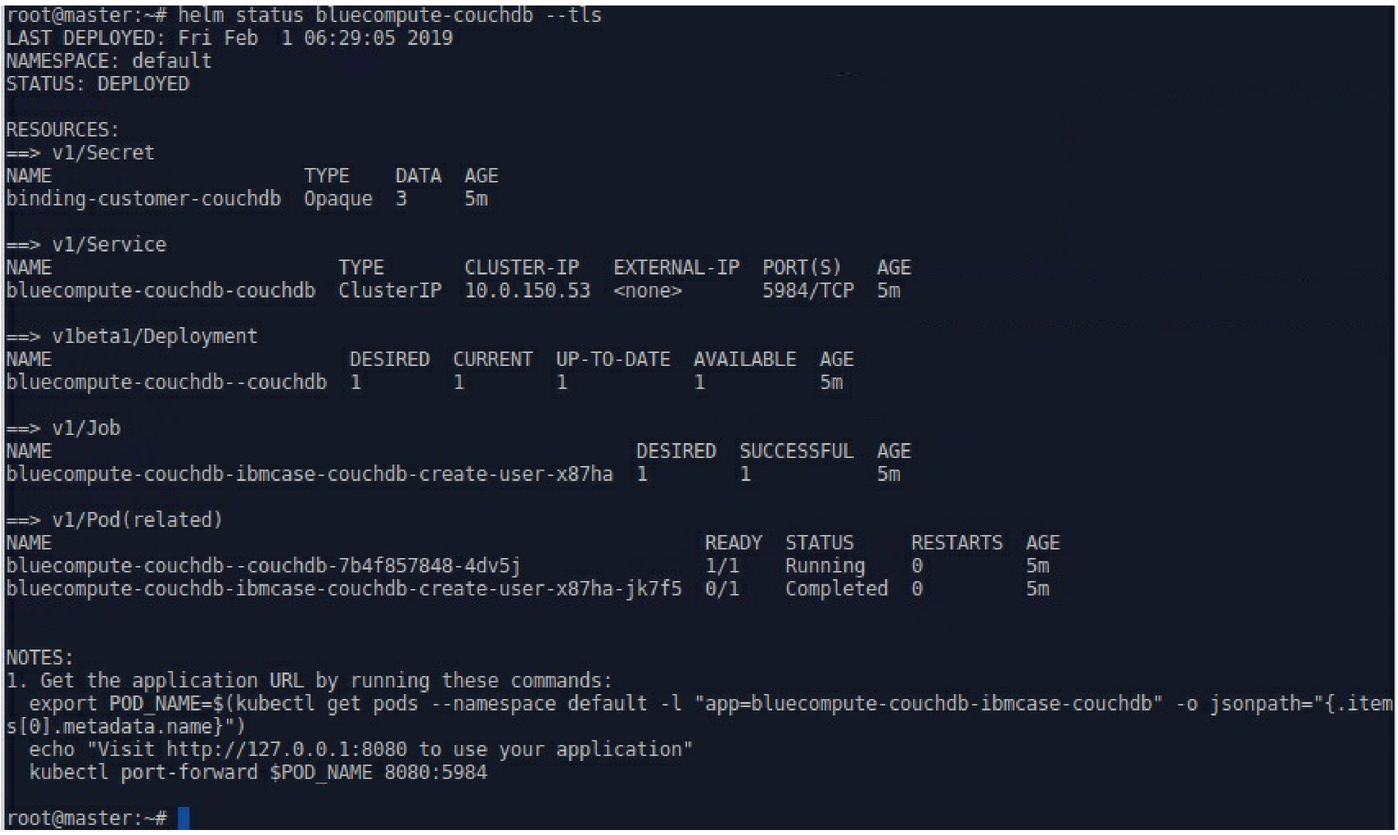
```
==> v1beta1/Deployment
NAME                          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
bluecompute-couchdb--couchdb  1        1        1           0          0s

==> v1/Job
NAME                                                  DESIRED  SUCCESSFUL  AGE
bluecompute-couchdb-ibmcase-couchdb-create-user-x87ha  1        0           0s

==> v1/Pod(related)
NAME                                                        READY  STATUS             RESTARTS  AGE
bluecompute-couchdb--couchdb-7b4f857848-4dv5j               0/1    ContainerCreating  0         0s
bluecompute-couchdb-ibmcase-couchdb-create-user-x87ha-jk7f5  0/1    ContainerCreating  0         0s


NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app=bluecompute-couchdb-ibmcase-couchdb" -o jsonpath="
{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl port-forward $POD_NAME 8080:5984

[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // container
[Pipeline] }
[Pipeline] // node
[Pipeline] }
[Pipeline] // podTemplate
[Pipeline] End of Pipeline
Finished: SUCCESS
```

10. 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.



```
root@master:~# kubectl get pods | grep couchdb
bluecompute-couchdb--couchdb-7b4f857848-4dv5j                1/1      Running     0          3m
bluecompute-couchdb-ibmcase-couchdb-create-user-x87ha-jk7f5  0/1      Completed   0          3m
root@master:~#
```

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



```
root@master:~# helm status bluecompute-couchdb --tls
LAST DEPLOYED: Fri Feb  1 06:29:05 2019
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                     TYPE    DATA  AGE
binding-customer-couchdb  Opaque  3     5m

==> v1/Service
NAME                        TYPE       CLUSTER-IP   EXTERNAL-IP  PORT(S)   AGE
bluecompute-couchdb-couchdb  ClusterIP  10.0.150.53  <none>       5984/TCP  5m

==> v1beta1/Deployment
NAME                         DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
bluecompute-couchdb--couchdb  1        1        1           1          5m

==> v1/Job
NAME                                                 DESIRED  SUCCESSFUL  AGE
bluecompute-couchdb-ibmcase-couchdb-create-user-x87ha  1        1           5m

==> v1/Pod(related)
NAME                                                       READY  STATUS     RESTARTS  AGE
bluecompute-couchdb--couchdb-7b4f857848-4dv5j              1/1    Running    0         5m
bluecompute-couchdb-ibmcase-couchdb-create-user-x87ha-jk7f5  0/1    Completed  0         5m


NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app=bluecompute-couchdb-ibmcase-couchdb" -o jsonpath="{.item
s[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl port-forward $POD_NAME 8080:5984

root@master:~#
```

```
  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.

12. Open a browser and navigate to `https://github.com/davemulley/icp-jenkins-helm-bluecompute/tree/master/charts/ibmcase-couchdb` . Review the `Jenkinsfile`

    Can you determine what the Jenkinsfile is doing?

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.

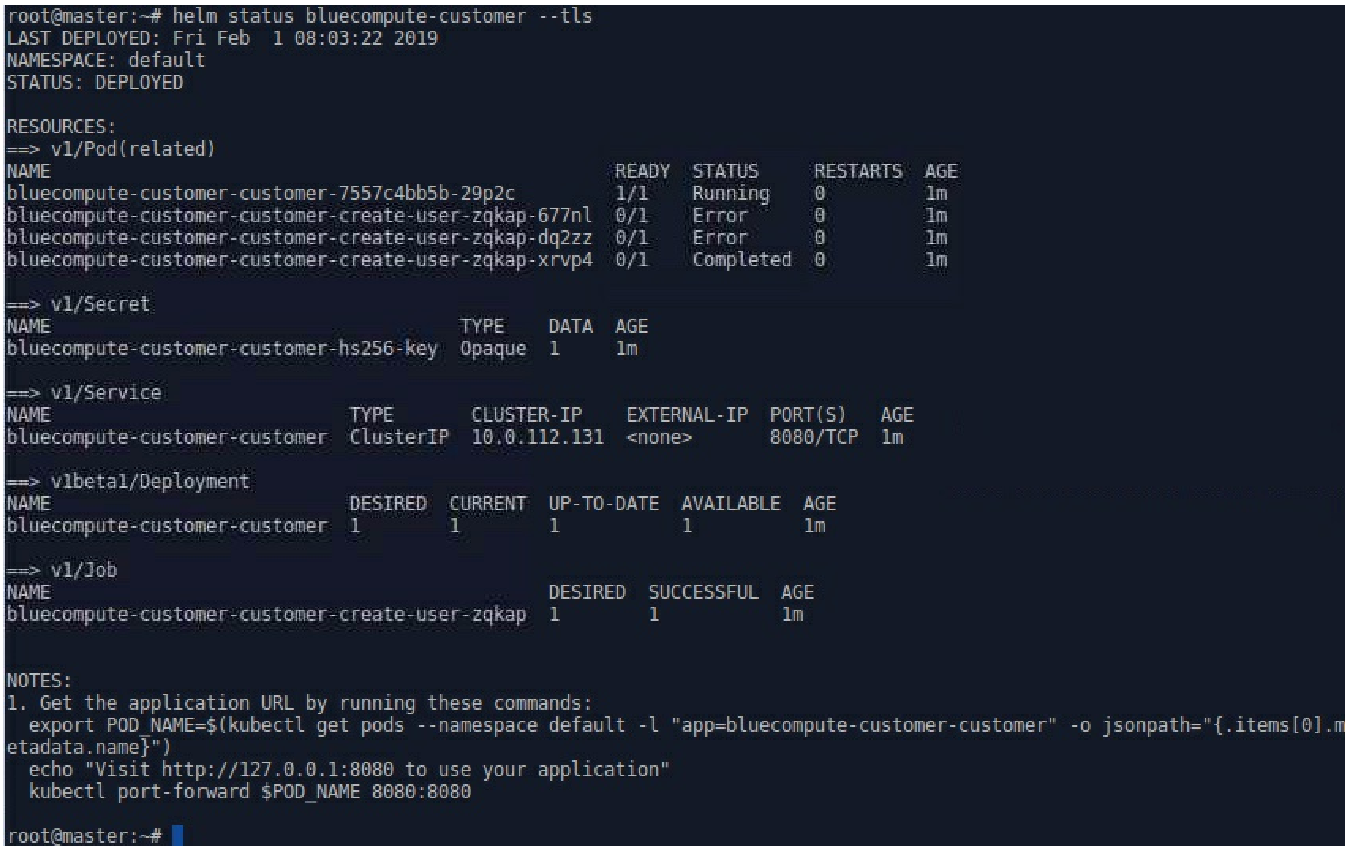
You would never need to build the pipeline manually.

1. There are 4 microservices built into the BlueCompute application: customer, inventory, catalog and orders. You will deploy one of them here for demonstrations purposes

2. Create a new Jenkins pipeline using the following specifications for the `customer` application.

   - Name: `Customer`
   - Repository URL: `https://github.com/davemulley/refarch-cloudnative-micro-customer`
   - Branch Specifier: `*/master`
   - Script Path: `Jenkinsfile`
   - Run the pipeline using the Build now link.
   - View the `console` output of the build job
   - Make sure the `customer` pod is running

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



```
root@master:~# helm status bluecompute-customer --tls
LAST DEPLOYED: Fri Feb  1 08:03:22 2019
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Pod(related)
NAME                                                READY  STATUS     RESTARTS  AGE
bluecompute-customer-customer-7557c4bb5b-29p2c      1/1    Running    0         1m
bluecompute-customer-customer-create-user-zqkap-677nl  0/1    Error      0         1m
bluecompute-customer-customer-create-user-zqkap-dq2zz  0/1    Error      0         1m
bluecompute-customer-customer-create-user-zqkap-xrvp4  0/1    Completed  0         1m

==> v1/Secret
NAME                                   TYPE    DATA  AGE
bluecompute-customer-customer-hs256-key  Opaque  1     1m

==> v1/Service
NAME                           TYPE       CLUSTER-IP    EXTERNAL-IP  PORT(S)   AGE
bluecompute-customer-customer  ClusterIP  10.0.112.131  <none>       8080/TCP  1m

==> v1beta1/Deployment
NAME                           DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
bluecompute-customer-customer  1        1        1           1          1m

==> v1/Job
NAME                                             DESIRED  SUCCESSFUL  AGE
bluecompute-customer-customer-create-user-zqkap  1        1           1m


NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app=bluecompute-customer-customer" -o jsonpath="{.items[0].m
etadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl port-forward $POD_NAME 8080:8080

root@master:~#
```

4. Open a browser navigate to `https://github.com/davemulley/refarch-cloudnative-micro-customer` and

review the `Jenkinsfile`

How is this Jenkinsfile building the Docker Image?

How is this Jenkinsfile deploying the application?

Conclusion

In this exercise you have used Jenkins to perform tasks including deploying a helm chart, compiling code, building a Docker Image and also pushing a Docker Image to the ICP Registry. Jenkins dynamically created the `slaves` to perform the following steps which are the types of steps that you will automate for microservices as well as applications that have been modernized to run in IBM Cloud Private.

*** End of exercises ***