ASSIGNMENT 4

Assignment Date	**22/10/2022**
Student Name	**S.Sivaranjani**
Student Roll No	**960519104081**
Maximum Marks	**2 Marks**

1.Pull an Image from docker and run it in docker playground.

Pull Image

Pulling an image from DockerHub is easy. First of all, you need to go to your profile and get the repository name. I got the name docker-demo. Then you just type below command to your terminal to get latest image from DockerHub.

docker pull coderkan/docker-demo:latest

You can see the installed images with docker images. You will see the details like below.

```
➜  docker-demo git:(main) docker pull coderkan/docker-demo:latest
latest: Pulling from coderkan/docker-demo
0ecb575e629c: Already exists
7467d1831b69: Already exists
feab2c490a3c: Already exists
f15a0f46f8c3: Already exists
26cb1dfcbebb: Already exists
5b224ce6d4ea: Already exists
c932fe81bb40: Already exists
b079b2033d71: Already exists
4f4fb700ef54: Already exists
Digest: sha256:c561be4395468e6da7d4a6397520eb13ba92d599abdb7176834ed46f962aa37d
Status: Downloaded newer image for coderkan/docker-demo:latest
docker.io/coderkan/docker-demo:latest
➜  docker-demo git:(main) docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
coderkan/docker-demo   latest              84b6698cb652        5 minutes ago       531MB
```

Running Application

You need to run this installed image with run command.

docker run -p 8080:8080 coderkan/docker-demo

You will see the Started DockerDemoApplication in xxx seconds.. This means that your application is started successfully. Now, you can check if the application is running or not with Browser.

```
➜  docker-demo git:(main) docker run -p 8080:8080 coderkan/docker-demo

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.2)

2021-02-14 12:00:28.422  INFO 1 --- [           main] c.c.dockerdemo.DockerDemoApplication     : Starting DockerDemoApplication v0.0.1-SNAPSHOT
 using Java 1.8.0_282 on c98e9e6c623c with PID 1 (/usr/src/my-sample-app.jar started by root in /usr/src)
2021-02-14 12:00:28.435  INFO 1 --- [           main] c.c.dockerdemo.DockerDemoApplication     : No active profile set, falling back to default
 profiles: default
2021-02-14 12:00:31.838  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2021-02-14 12:00:31.882  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-02-14 12:00:31.883  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.41
]
2021-02-14 12:00:32.047  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationCon
text
2021-02-14 12:00:32.048  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization com
pleted in 3449 ms
2021-02-14 12:00:32.631  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskE
xecutor'
2021-02-14 12:00:33.326  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with co
ntext path ''
2021-02-14 12:00:33.354  INFO 1 --- [           main] c.c.dockerdemo.DockerDemoApplication     : Started DockerDemoApplication in 6.402 seconds
 (JVM running for 7.929)
2021-02-14 12:00:48.786  INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatc
herServlet'
2021-02-14 12:00:48.786  INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2021-02-14 12:00:48.789  INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 3 ms
```

2.Create a docker file for the jobportal application and deploy it in Docker desktop application.

To create a Docker container, you need to create a Dockerfile on your project. Using that file, you can create a Docker container which can run on any platform without installing any libraries on the actual machine.

Docker allows you to package an application with its environment and all of its dependencies into an encapsulated “box”, called a container. Usually, a container consists of an application running in a stripped-to basics version of a Linux operating system. An image is the blueprint for a container, a container is a running instance of an image.

Creating a simple Node.js application

First, create a new directory and create a `package.json` file inside:

```
{
"name": "nodejs app",
"version": "1.0.0",
"description": "create a dockerfile on Nodejs project",
"author": "sarasa Gunawardhana",
"main": "server.js",
"scripts": {
"start": "node server.js"
},
"dependencies": {
"express": "^4.16.1"
}
```

```
}
```

Run `npm install`.

Then, create a `server.js` file that defines a web app using the Express.js framework:

```
'use strict';const express = require('express');// Constants
const PORT = 3000;const app = express();
app.get('/', (req, res) => {
res.send('Docker and Nodejs');
});app.listen(PORT, HOST);
console.log(`Running on ${PORT}`);
```

Creating a Dockerfile

create a file as D**ockerfile**

```
touch Dockerfile
# OR
nano Dockerfile
```

The first thing we need to do is define from what image we want to build. Here we will use the latest Node, version 11.

```
FROM node:11
```

Next, we create a directory to hold the application code inside the image, this will be the working directory for your application:`# Create app directory`

```
WORKDIR /usr/src/app
```

This image comes with Node.js and NPM already installed so the next thing we need to do is to install your app dependencies using the `npm` binary. Please note that if you are using `npm` version 4 or earlier a `package-lock.json` file will *not* be generated.

```
# Install app dependencies
# A wildcard is used to ensure both package.json AND package
lock.json are copied
# where available (npm@5+)
```

```
COPY package*.json ./
RUN npm install
# If you are building your code for production
# RUN npm ci --only=production
```

Note that, rather than copying the entire working directory, we are only copying the `package.json` file. This allows us to take advantage of cached Docker layers. bitJudo has a good explanation of this here. Furthermore, the `npm ci` command, specified in the comments, helps provide faster, reliable, reproducible builds for production environments. You can read more about this here.

To bundle your app's source code inside the Docker image, use the `COPY` instruction:

```
# Bundle app source
COPY . .
```

Your app binds to port `8080` so you'll use the `EXPOSE` instruction to have it mapped by the `docker` daemon:

```
EXPOSE 3000
```

Last but not least, define the command to run your app using `CMD` which defines your runtime. Here we will use the basic `npm start` which will run `node server.js` to start your server:

```
CMD [ "npm", "start" ]
```

Your `Dockerfile` should now look like this:

```
FROM node:8
# Create app directory
WORKDIR /usr/src/app
# Install app dependencies
# A wildcard is used to ensure both package.json AND package
lock.json are copied
# where available (npm@5+)
COPY package*.json ./
```

```
RUN npm install

# If you are building your code for production

# RUN npm ci --only=production

# Bundle app source

COPY . .

EXPOSE 3000

CMD [ "npm", "start" ]
```

Building your image

Go to the directory that has your `Dockerfile` and run the following command to build the Docker image. The `-t` flag lets you tag your image so it's easier to find later using the `docker images` command:

```
$ docker build -t <your username>/node-web-app .
```

Your image will now be listed by Docker:

```
$ docker images# Example

REPOSITORY TAG ID

node 8 1934b0b038d1 5 days
ago
<your username>/node-web-app latest d64d3505b0d2 1 minute
ago
```

Run the image

Running your image with `-d` runs the container in detached mode, leaving the container running in the background. The `-p` flag redirects a public port to a private port inside the container. Run the image you previously built:

```
$ docker run -p 49160:8080 -d <your username>/node-web-app
```

Print the output of your app:

```
# Get container ID
$ docker ps# Print app output
$ docker logs <container id># Example
Running on http://localhost:8080
```

If you need to go inside the container you can use the `exec` command:

```
# Enter the container
$ docker exec -it <container id> /bin/bash
```

Test

To test your app, get the port of your app that Docker mapped:

```
$ docker ps# Example
ID IMAGE COMMAND ...
PORTS
```

```
ecce33b30ebf <your username>/node-web-app:latest npm start ...
49160->8080
```

In the example above, Docker mapped the `8080` port inside of the container to the port `49160` on your machine.Now you can call your app using `curl` (install if needed via: `sudo apt get install curl`):

```
$ curl -i localhost:49160HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-M6tWOb/Y57lesdjQuHeB1P/qTV0"
Date: Mon, 13 Nov 2017 20:53:59 GMT
Connection: keep-aliveDocker on nodejs
```

Deploy your app:

Now, that we have configured a CI/CD pipleline, let's look at how we can deploy the application. Docker supports deploying containers on Azure ACI and AWS ECS. You can also deploy your application to Kubernetes if you have enabled Kubernetes in Docker Desktop.

Docker and Azure ACI

applications in Azure Container Instances (ACI) when building cloud-native applications. The new experience provides a tight integration between Docker Desktop and Microsoft Azure allowing developers The Docker Azure Integration enables developers to use native Docker commands to run to quickly run applications using the Docker CLI or VS Code extension, to switch seamlessly from local development to cloud deployment.

Docker and AWS ECS

The Docker ECS Integration enables developers to use native Docker commands in Docker Compose CLI to run applications in Amazon EC2 Container Service (ECS) when building cloud native applications.

The integration between Docker and Amazon ECS allows developers to use the Docker Compose CLI to set up an AWS context in one Docker command, allowing you to switch from a local context to a cloud context and run applications quickly and easily simplify multi-container application development on Amazon ECS using Compose files.

For detailed instructions, see Deploying Docker containers on ECS.Kubernetes

Docker Desktop includes a standalone Kubernetes server and client, as well as Docker CLI integration that runs on your machine. When you enable Kubernetes, you can test your workloads on Kubernetes.

To enable Kubernetes:

1. From the Docker menu, select Preferences (Settings on Windows).
2. Select Kubernetes and click Enable Kubernetes.

This starts a Kubernetes single-node cluster when Docker Desktop starts.

3.Create IBM container registery and deploy helloworld app or jobportalapp.

Objectives :

- Learn the similarities between Code Engine and Cloud Foundry.
- Learn the general process of deploying apps in Code Engine.
- Deploy an application from code on your local system with Code Engine.

Prerequisites :

Before you can get started with Code Engine, you need to set up your account and install the CLI.

• All Code Engine users are required to have a Pay-as-you-Go account.

• While you can use Code Engine through the console, the examples in this documentation focus on the command line.

$ibmcloud plugin
install code-engine

Step 1: Log in to IBM Cloud

1. Log in to the IBM Cloud CLI.

$ibmcloud login

2. Target a resource group by running the following command. To get a list of your resource groups, run **ibmcloud resource groups.**

$ibmcloud target -g
<resource_group>

Example output :

Targeted resource group default

Step 2: Creating a project

• A Code Engine "project" is similar to a Cloud Foundry "space" in that it groups related workloads into a logical collection that is meaningful to the developer.

• You can group workloads into different projects based on whatever criteria that makes sense to you, for example, company organization structure, dependencies between the workloads, or development versus test versus production environments.

1. Create a project in Code Engine called sample.

$ibmcloud ce project
create --name sample

Example output :

Creating project 'sample'…
ID for project 'sample' is 'abcdabcd-abcd-abcd-abcd-abcd12e3456f7'.Waiting for project 'sample' to be active…
Now selecting project 'sample'.
OK

Step 3: Creating a directory and source code

1. Create a directory on your local workstation that is named **myapp** and navigate into it. In this directory, save all the files that are required to build the image and to run your app

$Mkdir myapp && cd myapp

2. Create a file called **server.js** and copy the following source code into it.

```
Const http = require('http');
http.createServer(function (request, response) {
response.writeHead(200, {'Content-Type': 'text/plain'});
response.end( "Hello world\n" );
}).listen(8080);
```

Step 4: Deploying your application

• Push your code to Code Engine by using the **application create** command.

• The following example creates an application called **myapp** that uses the **buildpack** strategy and provides the location for the source code in the current directory **(.)**.

$Ibmcloud ce app create –name myapp –build-source . –strategy buildpacks

Example output :

Creating application 'myapp'…
Packaging files to upload from source path '.'…
Submitting build run 'myapp-run-220999-210706331'…
Creating image 'private.us.icr.io/ce—6ef04-khxrbwa0lci/app-myapp:220418-0207-askql'…

Waiting for build run to complete…
Build run status: ‘Running’
Build run completed successfully.
Run ‘ibmcloud ce buildrun get -n myapp-run-220000-210706331’ to check the build run status.
Waiting for application ‘myapp’ to become ready.
Configuration ‘myapp’ is waiting for a Revision to become ready.
Ingress has not yet been reconciled.Waiting for load balancer to be ready.
Run ‘ibmcloud ce application get -n myapp’ to check the application status.
OK

Reference link :

https://myapp.abcdbwa0lci.us-south.codeengine.appdomain.cloud

Let’s take a deeper look at the previous app create command. Notice that the output of the app create command provides information about the progression of the build run before the app is created and deployed.

• Code Engine receives a request to create an application from source code (instead of pulling directly from an image).
• Code Engine checks for an IAM service ID and API key that is associated with the selected project. This service ID must be authorized to read and write to IBM Cloud® Container Registry. If no service ID exists, Code Engine creates one for you. Note that this service ID is used for subsequent Code Engine build requests that are run from the same project.
• This example builds code from a local source **(--build-source .)**. The source code is packed into an archive file and uploaded to a managed namespace within the IBM Cloud® Container Registry instance in your account. Note that you can target only IBM Cloud® Container Registry for your local builds.
• Code Engine builds your source code into an image. The source image is created in the same namespace as your source archive file.
• After the build completes, your application is deployed. You can access your application from the provided URL.

Clean up :

After you finish this tutorial, you can clean up the resources that you created with the following commands.

• Delete your application

$ibmcloud ce app delete --name myapp

4.Create a Kubernets cluster in IBM cloud and deploy hello world image or jobportal image and also expose the same app to run in nodeport.

Install the IBM Cloud Developer Tools

Install the IBM Cloud CLI.

curl -sL https://ibm.biz/idt-installer | bash

Verify your installation

ibmcloud dev help

Connect to the proper IBM API endpoint for your IBM Cloud location. Example:

ibmcloud api https://api.ng.bluemix.net

Log in to IBM Cloud using your IBMid

ibmcloud login. Use the --sso option to log in using your federated ID.

Set up your org and space

ibmcloud target --cf

To follow this guide, you can use a free cluster. You can also use a paid cluster of type standard on IBM Cloud.

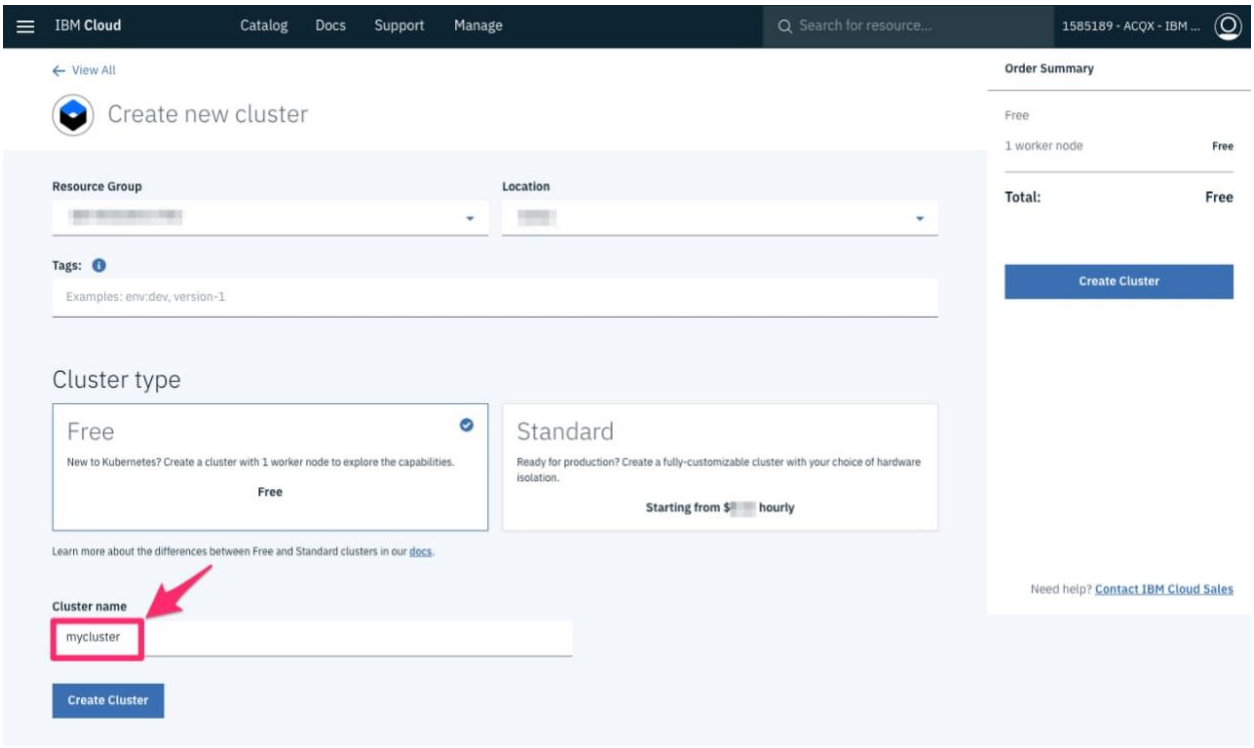
Procedure:

Create a Kubernetes cluster Kubernetes Service delivers powerful tools by combining Docker and Kubernetes technologies, an intuitive user experience, and built-in security and isolation to automate the deployment, operation, scaling, and monitoring of containerized apps in a cluster of computing hosts.

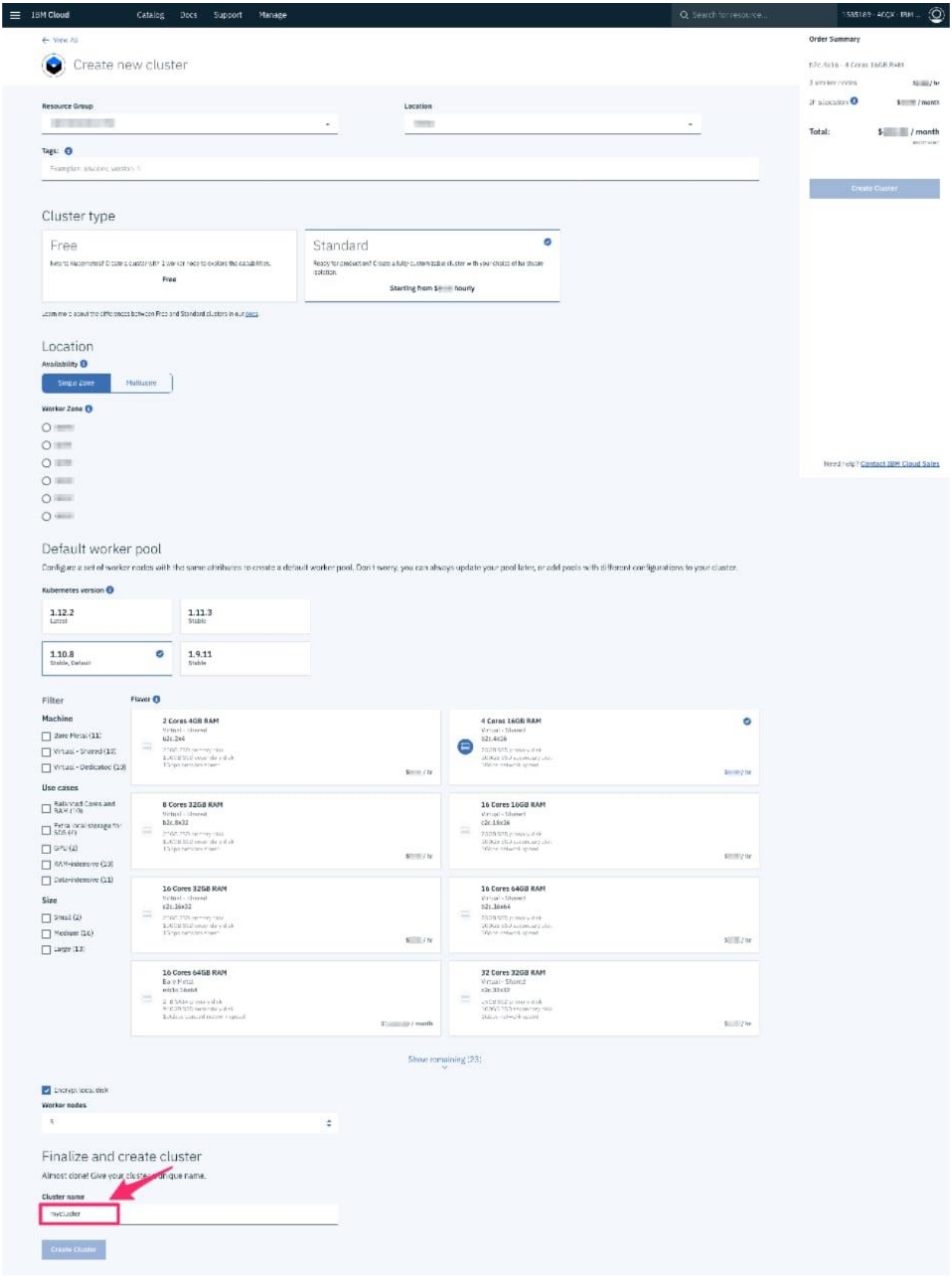
To set up the Kubernetes cluster:

1.Create a Kubernetes cluster from the IBM Cloud catalog).

2.When configuring the new cluster, select the Cluster type and click Create Cluster to provision a Kubernetes cluster. 2.1 In the case of a free cluster you will see something similar to:



2.2 In the case of a paid cluster you will see something similar to:


IBM Cloud
Catalog
Docs
Support
Manage
Search for resource...
Create new cluster
Resource Group
Location
Tags
Cluster type
Free
Standard
Starting from $ hourly
Location
Availability
Single zone
Multizone
Worker Zone
Default worker pool
Configure a set of worker nodes with the same attributes to create a default worker pool. Don't worry, you can always update your pool later, or add pools with different configurations to your cluster.
Kubernetes version
1.12.2
Latest
1.11.3
Stable
1.10.8
Stable, Default
1.9.11
Stable
Filter
Machine
Bare Metal (11)
Virtual - Shared (10)
Virtual - Dedicated (10)
Use cases
GPU (2)
Size
Small (2)
Medium (16)
Large (13)
Flavor
2 Cores 4GB RAM
4 Cores 16GB RAM
8 Cores 32GB RAM
16 Cores 16GB RAM
16 Cores 32GB RAM
16 Cores 64GB RAM
16 Cores 64GB RAM
32 Cores 32GB RAM
Show remaining (23)
Encrypt local disk
Worker nodes
Finalize and create cluster
Almost done! Give your cluster a unique name.
Cluster name
mycluster
Create Cluster
Order Summary
Total:
/ month
Create Cluster
Need help? Contact IBM Cloud Sales

3.Check the status of your Cluster and Worker Nodes and wait for them to be ready.

Or, if you prefer, create the cluster using the IBM Cloud CLI tools)

2.Configure kubectl kubectl is a CLI tool to interact with a Kubernetes cluster. In this occasion, you will use it to point forward to the created Kubernetes cluster.

1.Use ibmcloud login to log in interactively into the IBM Cloud. Provide the organization (org), location and space under which the cluster is created. You can reconfirm the details by running ibmcloud target command.

2When the cluster is ready, retrieve the cluster configuration by using the cluster's name: ibmcloud cs cluster-config <clusterName>

3.Copy and paste the export command to set the KUBECONFIG environment variable as directed. The command should be something similar to:

export KUBECONFIG=/Users/user/.bluemix/plugins/container service/clusters/JupyterHub/kube-config-***-JupyterHub.yml

To verify whether the KUBECONFIG environment variable is set correctly or not, run the following command:

echo $KUBECONFIG

4.Check that the kubectl command is correctly configured
kubectl cluster-info

```
Kubernetes master is running at https://c7.us-south.containers.cloud.ibm.com:24371
Heapster is running at https://c7.us-south.containers.cloud.ibm.com:24371/api/v1/namespaces/kube-system/services/heapster/proxy
KubeDNS is running at https://c7.us-south.containers.cloud.ibm.com:24371/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
kubernetes-dashboard is running at https://c7.us-south.containers.cloud.ibm.com:24371/api/v1/namespaces/kube-system/services/https:kube
rnetes-dashboard:/proxy
```