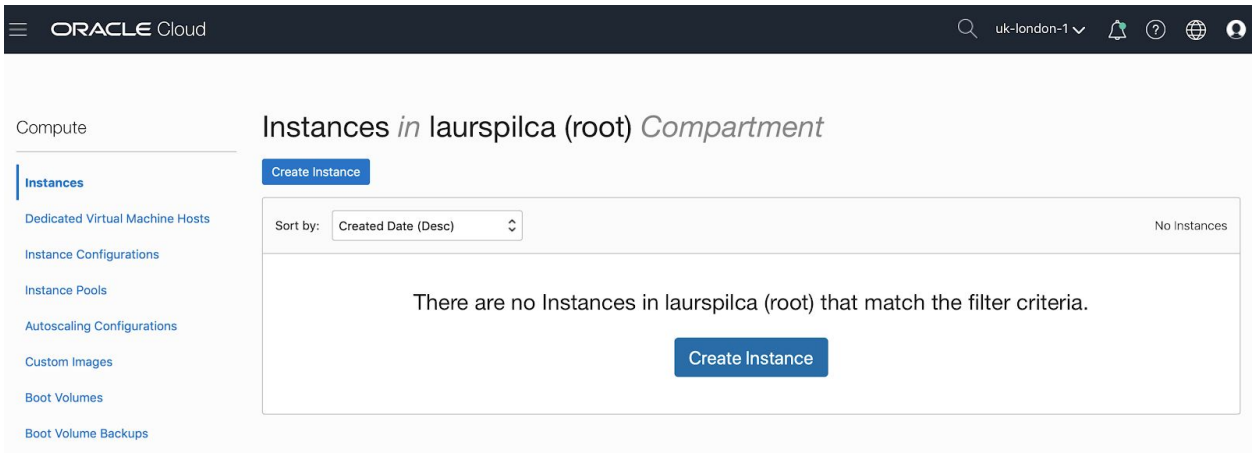
HOL - Architectural Antipatterns when Delivering a Software System with Kubernetes

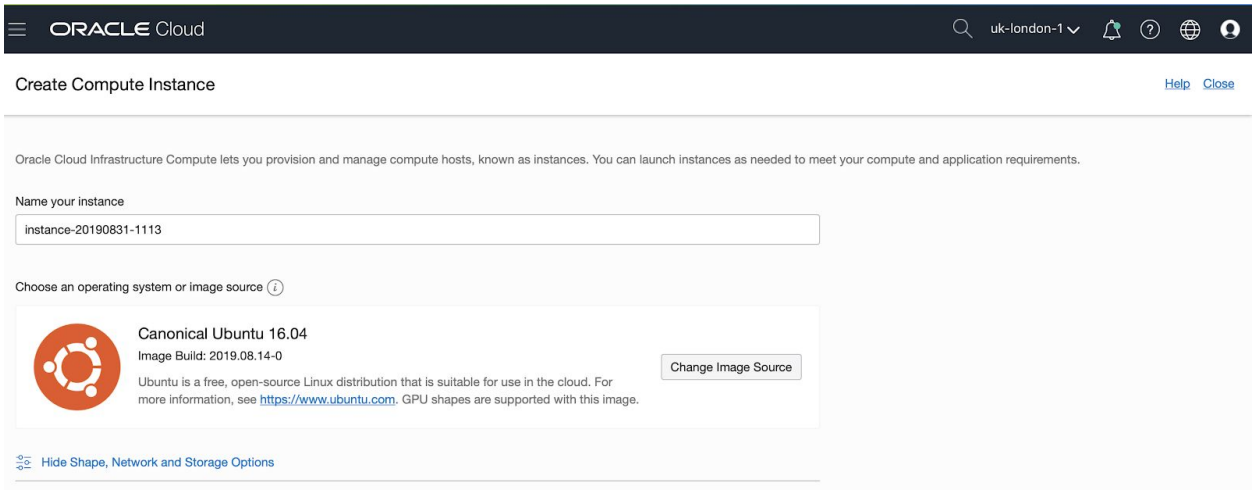
Creating the VM

Our examples will be run in Oracle Cloud infrastructure. Each of us having the same VM setup will allow us to avoid configuration issues and focus on the purpose of the lab.

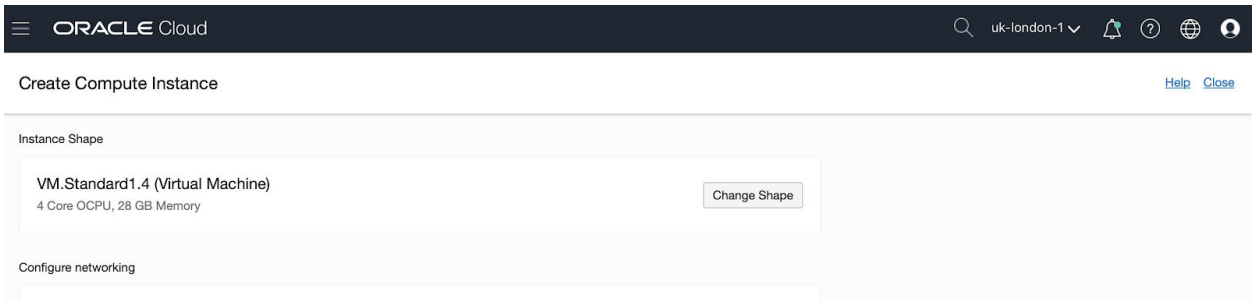
After creating your Oracle Cloud account you can add one new VM from Compute -> Instances section that you see below:



Creating an instance only takes a few seconds. We will use the Ubuntu 16.04 image for our VMs.



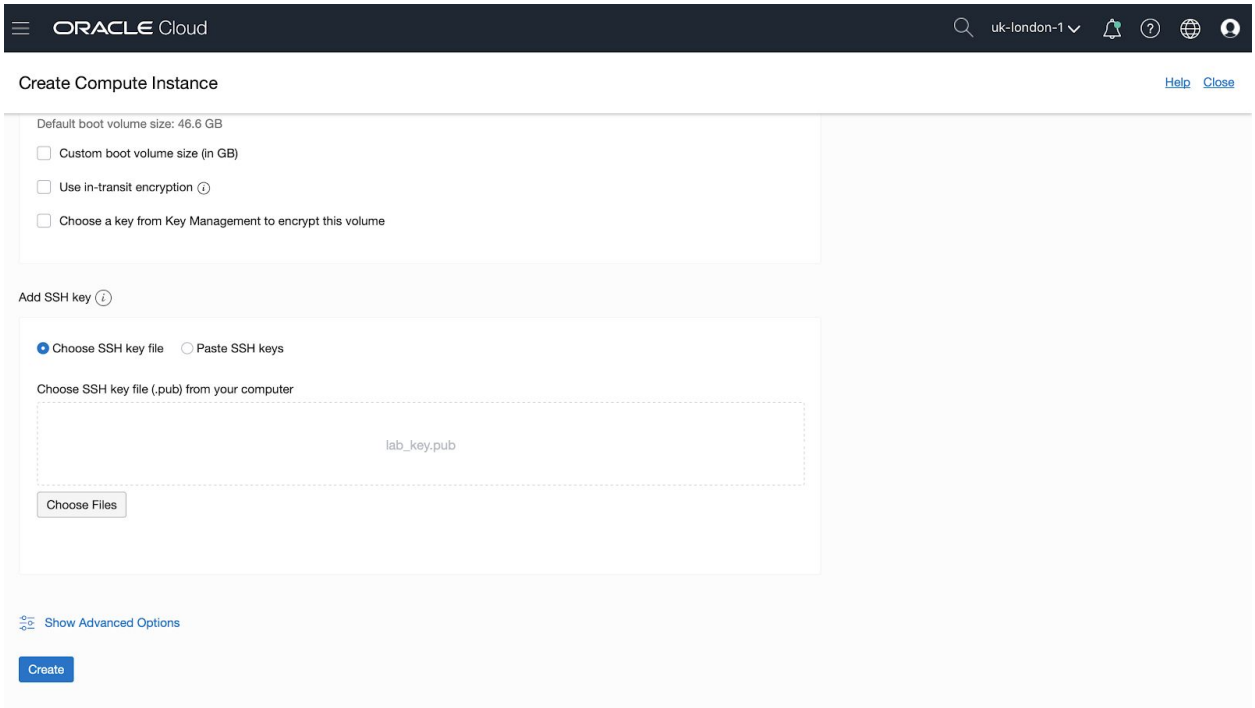
Also, very important, we will change the shape to one that provides us at least 4 cores of CPU.



Further, you will need to create a public - private key pair. If you are using a mac / linux you can do this in the terminal by using the **ssh-keygen -t rsa** command. If you are using a Windows machine, you can as well use the same command in a Git bash console.

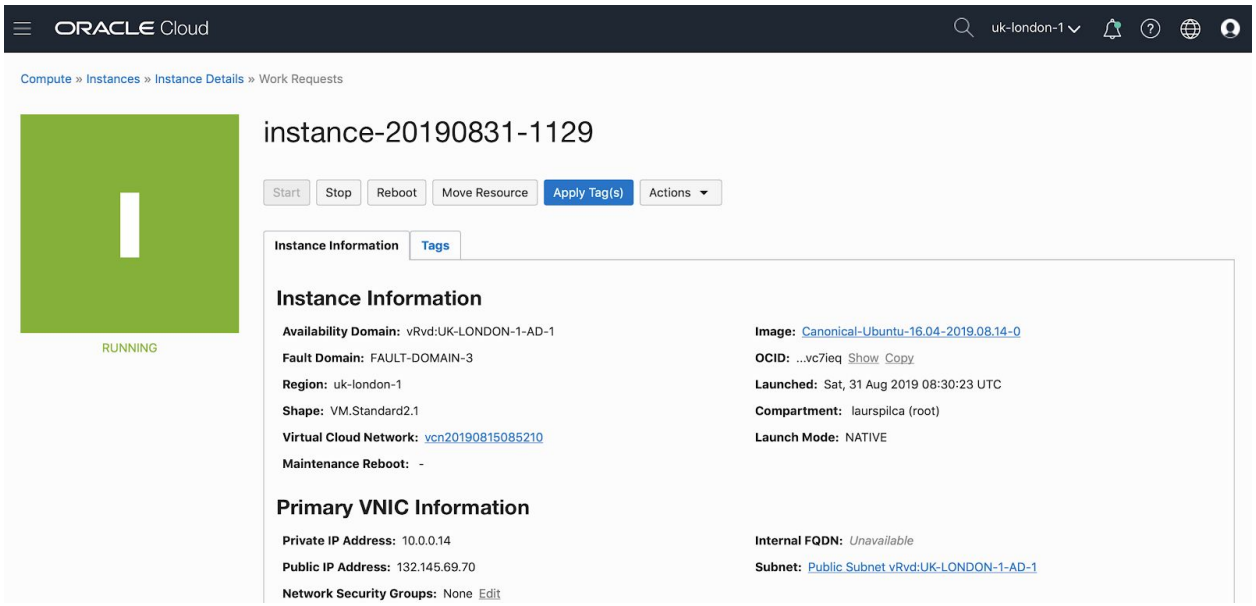
```
192-168-0-100:Documents spilca$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/spilca/.ssh/id_rsa): lab_key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in lab_key.
Your public key has been saved in lab_key.pub.
The key fingerprint is:
SHA256:7jYJunt061dvOzrdQ5/+GalSMG1bUgNwrL/seeqGH7g spilca@192-168-0-100.rdsnet.ro
The key's randomart image is:
+---[RSA 2048]----+
|           .oo.  |
|            .. o |
|            o . .|
|           + + . |
|        S   = +  |
|      o..   o+ ..|
|     o o.o o++o+o|
|    . ..= .oo+O+=|
|    o+ ooo EB@==+|
+----[SHA256]-----+
192-168-0-100:Documents spilca$
```

The resulting public key should be added to the VM before pressing the create button.


ORACLE Cloud
uk-london-1
Create Compute Instance
Help
Close
Default boot volume size: 46.6 GB
Custom boot volume size (in GB)
Use in-transit encryption
Choose a key from Key Management to encrypt this volume
Add SSH key
Choose SSH key file
Paste SSH keys
Choose SSH key file (.pub) from your computer
lab_key.pub
Choose Files
Show Advanced Options
Create

Connecting to the VM

Once your VM instance is running, you will also have a public IP address to connect to it:



Use the public IP address and the private part of the key pair to connect to the machine.

ssh -i lab_key ubuntu@<ip address>

```
192-168-0-100:Documents spilca$ ssh -i lab_key ubuntu@132.145.69.70
The authenticity of host '132.145.69.70 (132.145.69.70)' can't be established.
ECDSA key fingerprint is SHA256:eIhsaglqZlTjJ3OrtqsrNO1J4W1Qoa+ZyLYbOLPNMDQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '132.145.69.70' (ECDSA) to the list of known hosts.
Enter passphrase for key 'lab_key':
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1021-oracle x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.
```

Once you are connected, gain root privileges by using the **sudo -i** command

```
ubuntu@instance-20190831-1129:~$ sudo -i
------------------------------------------------------------------------
WARNING! Your environment specifies an invalid locale.
 The unknown environment variables are:
   LC_CTYPE=UTF-8 LC_ALL=
 This can affect your user experience significantly, including the
 ability to manage packages. You may install the locales by running:

   sudo apt-get install language-pack-UTF-8
     or
   sudo locale-gen UTF-8

To see all available language packs, run:
   apt-cache search "^language-pack-[a-z][a-z]$"
To disable this message for all users, run:
   sudo touch /var/lib/cloud/instance/locale-check.skip
------------------------------------------------------------------------

root@instance-20190831-1129:~#
```

Setting up the Minikube cluster

We start by updating apt-get with the **apt-get update** command as you can see in the image below.

```
root@instance-20190831-1129:~# apt-get update -y
```

Once this is ready, we will install docker using the following command: **apt-get install -y docker.io**

```
root@instance-20190831-1129:~# apt-get install -y docker.io
```

You can check that Docker is installed correctly by running some basic Docker commands.

docker version

```
root@instance-20190831-1129:~# docker version
Client:
 Version:           18.09.7
 API version:       1.39
 Go version:        go1.10.4
 Git commit:        2d0083d
 Built:             Fri Aug 16 14:19:38 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          18.09.7
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.4
  Git commit:       2d0083d
  Built:            Thu Aug 15 15:12:41 2019
  OS/Arch:          linux/amd64
  Experimental:     false
```

docker images

```
root@instance-20190831-1129:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
root@instance-20190831-1129:~#
```

As you can see Docker looks to be correctly installed. For the moment there are no images locally.

Having Docker installed and working, we can go to the next step where we install our orchestration tool. We will install Minikube to simulate a Kubernetes cluster by running the following command in the terminal.

curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

```
root@instance-20190831-1129:~# curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux
-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53.2M  100 53.2M    0     0  24.7M      0  0:00:02  0:00:02 --:--:-- 24.7M
root@instance-20190831-1129:~#
```

We will also need the kubectl client tool to work with the Kubernetes cluster. Install it by running the following command.

curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.15.0/bin/linux/amd64/kubectl && chmod +x ./kubectl && sudo mv ./kubectl /usr/local/bin/kubectl

Pay attention at the version. If you install latest Minikube, as we have done with the previous command, you can face the situation of incompatibility with the current client version.

```
root@instance-20190831-1129:~# curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.15.0/bin/linux/amd
64/kubectl && chmod +x ./kubectl && sudo mv ./kubectl /usr/local/bin/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 40.9M  100 40.9M    0     0  40.3M      0  0:00:01  0:00:01 --:--:-- 40.4M
root@instance-20190831-1129:~#
```

At this moment you can already run Minikube:

minikube start --memory=8192 --cpus=4 --vm-driver=none

```
root@instance-20190831-1129:~# minikube start --memory=8192 --cpus=4 --vm-driver=none
😄  minikube v1.3.1 on Ubuntu 16.04
🤹  Running on localhost (CPUs=2, Memory=15029MB, Disk=46051MB) ...
ℹ️  OS release is Ubuntu 16.04.6 LTS
🐳  Preparing Kubernetes v1.15.2 on Docker 18.09.7 ...
💾  Downloading kubelet v1.15.2
💾  Downloading kubeadm v1.15.2
🚜  Pulling images ...
🚀  Launching Kubernetes ...
```

When the cluster is started, we can test it by running some basic commands:

minikube version

```
root@instance-20190831-1129:~# minikube version
minikube version: v1.3.1
commit: ca60a424ce69a4d79f502650199ca2b52f29e631
root@instance-20190831-1129:~#
```

kubectl version

```
root@instance-20190831-1129:~# kubectl version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0", GitCommit:"e8462b5b5dc2584fdcd18e6bcfe9f1e4d9
70a529", GitTreeState:"clean", BuildDate:"2019-06-19T16:40:16Z", GoVersion:"go1.12.5", Compiler:"gc", Platform:"linux/a
md64"}
Server Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.2", GitCommit:"f6278300bebbb750328ac16ee6dd3aa7d3
549568", GitTreeState:"clean", BuildDate:"2019-08-05T09:15:22Z", GoVersion:"go1.12.5", Compiler:"gc", Platform:"linux/a
md64"}
root@instance-20190831-1129:~#
```

Finally make sure that the pods in the kube system namespace are working by running the following command:

kubectl get pods --all-namespaces

```
root@instance-20190831-1129:~# kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY   STATUS             RESTARTS   AGE
kube-system   coredns-5c98db65d4-fs2fl               0/1     CrashLoopBackOff   4          3m27s
kube-system   coredns-5c98db65d4-vtxzd               0/1     CrashLoopBackOff   5          3m27s
kube-system   etcd-minikube                          1/1     Running            0          2m15s
kube-system   kube-addon-manager-minikube            1/1     Running            0          2m15s
kube-system   kube-apiserver-minikube                1/1     Running            0          2m18s
kube-system   kube-controller-manager-minikube       1/1     Running            0          2m22s
kube-system   kube-proxy-qw6r8                       1/1     Running            0          3m27s
kube-system   kube-scheduler-minikube                1/1     Running            0          2m15s
kube-system   storage-provisioner                    1/1     Running            0          3m24s
root@instance-20190831-1129:~#
```

Occasionally, you may see problems with the CodeDNS service as in the image above. The pods are not starting and remain in a continuous CrashLoopBackOff state. If this happens, run the following commands to clear the local iptables and restart the cluster:

systemctl stop kubelet
systemctl stop docker
iptables --flush
iptables -tnat --flush
systemctl start kubelet
systemctl start docker

Finally, everything should be running. Make sure that the environment is ok before proceeding to the next step.

```
root@instance-20190831-1129:~# kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY   STATUS    RESTARTS   AGE
kube-system   coredns-5c98db65d4-fs2fl               1/1     Running   7          10m
kube-system   coredns-5c98db65d4-vtxzd               1/1     Running   7          10m
kube-system   etcd-minikube                          1/1     Running   1          9m10s
kube-system   kube-addon-manager-minikube            1/1     Running   1          9m10s
kube-system   kube-apiserver-minikube                1/1     Running   1          9m13s
kube-system   kube-controller-manager-minikube       1/1     Running   1          9m17s
kube-system   kube-proxy-qw6r8                       1/1     Running   1          10m
kube-system   kube-scheduler-minikube                1/1     Running   1          9m10s
kube-system   storage-provisioner                    1/1     Running   1          10m
root@instance-20190831-1129:~#
```

Downloading the examples

All the examples are on GitHub in the provided repository.

Start by changing your directory to the one where you would like to store your examples.

Then, use git to checkout the examples repository from GitHub:

```
https://github.com/lspil/codeone2019kubernetes.git
```

```
root@instance-20190831-1129:/var# git clone https://github.com/lspil/codeone2019kubernetes.git
Cloning into 'codeone2019kubernetes'...
remote: Enumerating objects: 165, done.
remote: Counting objects: 100% (165/165), done.
remote: Compressing objects: 100% (70/70), done.
remote: Total 165 (delta 41), reused 165 (delta 41), pack-reused 0
Receiving objects: 100% (165/165), 69.26 KiB | 0 bytes/s, done.
Resolving deltas: 100% (41/41), done.
Checking connectivity... done.
root@instance-20190831-1129:/var#
```

You should now be able to find the examples:

```
root@instance-20190831-1129:/var# git clone https://github.com/lspil/codeone2019kubernetes.git
Cloning into 'codeone2019kubernetes'...
remote: Enumerating objects: 165, done.
remote: Counting objects: 100% (165/165), done.
remote: Compressing objects: 100% (70/70), done.
remote: Total 165 (delta 41), reused 165 (delta 41), pack-reused 0
Receiving objects: 100% (165/165), 69.26 KiB | 0 bytes/s, done.
Resolving deltas: 100% (41/41), done.
Checking connectivity... done.
root@instance-20190831-1129:/var# cd codeone2019kubernetes/
root@instance-20190831-1129:/var/codeone2019kubernetes# ls
saconf2019-e1  saconf2019-e2  saconf2019-e3  saconf2019-e4  saconf2019-e5  saconf2019-e6  saconf2019-e7  saconf2019-e8
root@instance-20190831-1129:/var/codeone2019kubernetes#
```

Examples structure

To make things easy, all the examples have the same structure. They are all Spring Boot applications for which already a Docker image has been created and stored on Docker Hub. So there is no need to create ourselves the images to use them for creating containers. It is not in the scope of the lab to create the Docker images, but if you want to review that part as well you can checkout Appendix 1.

```
root@instance-20190831-1129:/var/codeone2019kubernetes# ls saconf2019-e1
Dockerfile  kube  mvnw  mvnw.cmd  pom.xml  src
root@instance-20190831-1129:/var/codeone2019kubernetes#
```

Beside the source code, each of the examples includes the small Dockerfile that was used to create the docker image. Also, they include a kube folder with the yml files which define the Kubernetes components used in the examples.

```
root@instance-20190831-1129:/var/codeone2019kubernetes# ls saconf2019-e1/kube/
deployment.yml  service.yml
root@instance-20190831-1129:/var/codeone2019kubernetes#
```

To make sure everything works correctly before diving into the first anti-pattern, we can deploy the first example. You can do this by running the following command:

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# kubectl apply -f kube
deployment.apps/hello-deployment created
service/hello-service created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1#
```

After deploying the application, you can check the deployments, pods and services:

kubectl get deployments
kubectl get services
kubectl get pods

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
hello-deployment   1/1     1            1           7m37s
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# kubectl get services
NAME            TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
hello-service   LoadBalancer   10.97.88.88   <pending>     8080:31035/TCP   7m42s
kubernetes      ClusterIP      10.96.0.1     <none>        443/TCP          70m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-deployment-85dc5dd9f6-dfgqk   1/1     Running   0          7m45s
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1#
```

Calling the application is now easy. Using the port from the service that is mapped to the 8080 inside the container, you can call the application endpoint.

curl http://localhost:31305/hello

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# curl http://localhost:31035/hello
Helloroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1#
```

In some of the examples, you will need to check the logs of your pods. This can be done using the kubectl logs command with the pod name as shown below.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-deployment-85dc5dd9f6-dfgqk   1/1     Running   0          11m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# kubectl logs hello-deployment-85dc5dd9f6-dfgqk

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.6.RELEASE)

2019-08-31 10:24:56.042  INFO 1 --- [           main] c.e.s.Saconf2019E1Application            : Starting Saconf2019E1A
pplication v0.0.1-SNAPSHOT on hello-deployment-85dc5dd9f6-dfgqk with PID 1 (/app.jar started by root in /)
2019-08-31 10:24:56.057  INFO 1 --- [           main] c.e.s.Saconf2019E1Application            : No active profile set,
 falling back to default profiles: default
2019-08-31 10:24:56.347  INFO 1 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springf
ramework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@38cccef: startup date [Sat Aug 31
10:24:56 GMT 2019]; root of context hierarchy
```

To delete the application you can run the kubectl delete command and provide the files that describe the components you want to delete. You can even provide the kube folder itself that contains all the components of the example.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# kubectl delete -f kube
deployment.apps "hello-deployment" deleted
service "hello-service" deleted
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1#
```

Example 1 - Statefulness

Statefulness is one of the biggest problems when it comes to horizontal scaling. Statefulness is any kind of data you can change and keep for more than the request scope. It could be a cache, a server side session or a in-memory database.

The container is designed to be disposable. Avoid storing data with it as it is guaranteed to be lost when Kubernetes will take action in recreating your containers if the application gets stuck (see last example). It is also a problem for many other reasons: availability of the data, AB deployments and progressive upgrades.

Deploy the application in folder example 2 by applying the files in the kube folder.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# kubectl apply -f kube
deployment.apps/stateful-app-deployment created
service/stateful-app created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2#
```

Test that the application is running. You should see more than one container. This is actually what happens in mostly all the situations in real-case scenarios. You will have multiple pods deployed in different availability zones.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
stateful-app-deployment-69589499bf-4tc7b   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-96zfx   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-d4g6r   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-drhct   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-dsffw   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-frfzs   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-hrfcw   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-lnn5m   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-mwmgf   1/1     Running   0          7m10s
stateful-app-deployment-69589499bf-whxp6   1/1     Running   0          7m10s
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2#
```

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# kubectl get services
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP      10.96.0.1        <none>        443/TCP          21h
stateful-app   LoadBalancer   10.105.216.184   <pending>     8080:30521/TCP   8m6s
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2#
```

The application exposes two endpoints. One of them is a POST operation changing a value. But the value is stored in-memory. The first pod that gets to be called through the load balancer will obtain the value. When trying to retrieve the value, we will observe that only sometimes the correct value is obtained. This happens only when the GET call is delivered again to the same pod that initially set the value.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# curl -XPOST http://localhost:30521/name/Bill
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# curl http://localhost:30521/hello
Hello, Billroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# curl http://localhost:30521/hello
Hello, nullroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# curl http://localhost:30521/hello
Hello, nullroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# curl http://localhost:30521/hello
Hello, nullroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2#
```

This example is just a dummy implementation to prove statefulness. But similar effects you will see with any application that applies stateful approaches. In a practical scenario it may be more subtle, like a cache causing performance problems.

There is also the approach of “solving” such problems instead of avoiding them. A good and well-known example would be the server side session. Using blackboarding and sticky sessions allows to use stateful approaches in multi-node deployment applications. But is it a good choice?

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2# kubectl delete -f kube
deployment.apps "stateful-app-deployment" deleted
service "stateful-app" deleted
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e2#
```

In the end you can delete the application to allow more resources for the next examples.

Example 2 - Keeping the secrets

It is common to see credentials, private keys and sensitive data in properties files in examples around the web. But that is not the place you should store them in a production environment.

With Kubernetes, use the secrets registry. Never store your sensitive data in code or any file that is eventually subversioned.

A separation approach for storing secrets allows to offer privileges easier as well as securing them better. Moreover, secrets have always been part of the responsibility of the deployment so they should stick where they belong.

Apply the files for example 3 to check how the secrets are used when deploying application with Kubernetes.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3# kubectl apply -f kube/
deployment.apps/secret-app-deployment created
secret/saconf2019-ex3-secret created
service/secret-app-service created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3#
```

The secret value in this case will be taken from the secret.yml file. You can change its value as you wish. The values is Base64 encoded.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3# cat kube/secret.yml
apiVersion: v1
kind: Secret
metadata:
  name: saconf2019-ex3-secret
type: Opaque
data:
  my.secret.name: Sm9obg==root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3#
```

The name that you have setup as a value in the secret file is the one finally consumed by the application. Mind, that the secret is not something you should subversion. In this case I have done this only for demo purpose.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3# kubectl get services
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes          ClusterIP      10.96.0.1       <none>        443/TCP          21h
secret-app-service  LoadBalancer   10.105.41.206   <pending>     8080:32561/TCP   3m31s
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3# curl http://localhost:32561
My secret name is: Johnroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3#
```

In the end you can delete the components deployed to allow more resources for the next examples.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3# kubectl delete -f kube
deployment.apps "secret-app-deployment" deleted
secret "saconf2019-ex3-secret" deleted
service "secret-app-service" deleted
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e3#
```

Example 3 - The undo rollout

Not only statefulness is a problem, but other components that represent dependencies of the application should also be taken into consideration.

One of the great values of using containers based on Docker images is that you can keep the images of the previous versions as much as you need them. The image is a blueprint for the container. So having the images of the previous versions allows you to rebuild containers for them and come back to a previous version fast.

The problem is that sometimes our system is also dependent on other components.

One of the easiest ways to prove this is using a SQL database. Once the structure of the database is changed, sometimes you cannot simply rollback to the previous version.

In example 4 start by deploying a MySQL server.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl apply -f kube/mysql-pv.yml
persistentvolume/mysql-pv-volume created
persistentvolumeclaim/mysql-pv-claim created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl apply -f kube/mysql-deployment.yml
deployment.apps/mysql created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl apply -f kube/mysql-service.yml
service/mysql created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
```

Then run a client to access the server and create a new database.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl run -it --rm --image=mysql:5.6 --restart=
Never mysql-client -- mysql -h mysql -ppassword
If you don't see a command prompt, try pressing enter.

mysql>
mysql>
```

Then you create a database that the application will use.

create database conf;

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl run -it --rm --image=mysql:5.6 --restart=
Never mysql-client -- mysql -h mysql -ppassword
If you don't see a command prompt, try pressing enter.

mysql>
mysql> create database saconf;
Query OK, 1 row affected (0.01 sec)

mysql>
```

The secrets file contains the url and credentials needed to connect to the database. Change the IP address with the correct one from your VM. Do not use localhost as that would refer to the container itself when running the application in a container.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# cat kube/secret.yml
apiVersion: v1
kind: Secret
metadata:
  name: saconf2019-e4-secret
type: Opaque
data:
  spring.datasource.url: amRiYzpteXNxbDovLzE3Mi4zMS40MC4zMzozMDAwMS9zYWNvbmY=
  spring.datasource.username: cm9vdA==
  spring.datasource.password: cGFzc3dvcmQ=root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# echo amRiYzpteXNxbDovLzE3Mi4zMS40MC4zMzozMDAwMS9z
YWNvbmY= | base64 -d
jdbc:mysql://172.31.40.33:30001/saconfroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
```

After encoding with the correct value, use vi to change the file and save it with the new secret.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# echo amRiYzpteXNxbDovLzE3Mi4zMS40MC4zMzozMDAwMS9z
YWNvbmY= | base64 -d
jdbc:mysql://172.31.40.33:30001/saconfroot@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# echo jdbc:mysql://10.0.0.14:30001/saconf | base64
amRiYzpteXNxbDovLzEwLjAuMC4xNDozMDAwMS9zYWNvbmYK
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
```

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl apply -f kube/secret.yml
secret/saconf2019-e4-secret created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl apply -f kube/deployment.yml
deployment.apps/app-with-sql-persistence-deployment created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl apply -f kube/service.yml
service/app-with-sql-persistence-service created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
```

Check that the application runs and then its logs.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl get pods
NAME                                              READY   STATUS    RESTARTS   AGE
app-with-sql-persistence-deployment-78cf96994c-jvvz2   1/1     Running   0          2m40s
mysql-7d7fdd478f-wwkvw                            1/1     Running   1          99m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
```

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl get pods
NAME                                              READY   STATUS    RESTARTS   AGE
app-with-sql-persistence-deployment-78cf96994c-jvvz2   1/1     Running   0          2m40s
mysql-7d7fdd478f-wwkvw                            1/1     Running   1          99m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl logs app-with-sql-persistence-deployment-
78cf96994c-jvvz2
```

In the logs you will see that one migration script was run by the application at deployment. Flyway is used to manage the scripts for the database structure.

```
y table: `saconf`.`flyway_schema_history`
2019-09-01 08:43:32.603  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Current version of sch
ema `saconf`: << Empty Schema >>
2019-09-01 08:43:32.605  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Migrating schema `saco
nf` to version 1 - CREATE USERS TABLE
2019-09-01 08:43:32.642  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Successfully applied 1
 migration to schema `saconf` (execution time 00:00.105s)
2019-09-01 08:43:32.833  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing
PersistenceUnitInfo [
        name: default
        ...]
2019-09-01 08:43:32.948  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate C
ore {5.3.7.Final}
```

We will now upgrade the application to the second version. To do that, in the deployment.yml file, change the version tag to v2.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-with-sql-persistence-deployment
  labels:
    app: app-with-sql-persistence-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app-with-sql-persistence-deployment
  template:
    metadata:
      labels:
        app: app-with-sql-persistence-deployment
    spec:
      containers:
      - name: app-with-sql-persistence-deployment
        image: laurentiuspilca/saconf2019-e4:v2
        env:
          - name: spring.datasource.url
            valueFrom:
              secretKeyRef:
                name: saconf2019-e4-secret
                key: spring.datasource.url
          - name: spring.datasource.username
            valueFrom:
              secretKeyRef:
                name: saconf2019-e4-secret
                key: spring.datasource.username
          - name: spring.datasource.password
            valueFrom:
              secretKeyRef:
                name: saconf2019-e4-secret
                key: spring.datasource.password
        ports:
-- INSERT --                                                                                    19,48         Top
```

Apply again the deployment configuration.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# vi kube/deployment.yml
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl apply -f kube/deployment.yml
deployment.apps/app-with-sql-persistence-deployment configured
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
```

In the logs you will see that one more script was applied and the version of the schema is now version 2.

```
/10.0.0.14:30001/saconf (MySQL 5.6)
2019-09-01 08:55:21.901  INFO 1 --- [           main] o.f.core.internal.command.DbValidate     : Successfully validated
 2 migrations (execution time 00:00.063s)
2019-09-01 08:55:21.915  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Current version of sch
ema `saconf`: 1
2019-09-01 08:55:21.919  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Migrating schema `saco
nf` to version 2 - ADD PASSWORD COLUMN
2019-09-01 08:55:21.978  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Successfully applied 1
 migration to schema `saconf` (execution time 00:00.067s)
2019-09-01 08:55:22.184  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing
PersistenceUnitInfo [
        name: default
        ...]
2019-09-01 08:55:22.315  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate C
ore {5.3.7.Final}
```

We'll assume that for some reason you want to rollback to the previous version of your application. Typically, this is done easily by only running one command with kubectl. So this is exactly what we will try to do.

kubectl rollout undo deployment app-with-sql-persistence-deployment

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl get deployments
NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
app-with-sql-persistence-deployment   1/1     1            1           23m
mysql                                1/1     1            1           119m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4# kubectl rollout undo deployment app-with-sql-pers
istence-deployment
deployment.extensions/app-with-sql-persistence-deployment rolled back
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e4#
```

Everything looks alright but when you check the logs you will see that actually there is a problem:

```
 2 migrations (execution time 00:00.096s)
2019-09-01 09:07:15.754  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Current version of sch
ema `saconf`: 2
2019-09-01 09:07:15.755  WARN 1 --- [           main] o.f.core.internal.command.DbMigrate      : Schema `saconf` has a
version (2) that is newer than the latest available migration (1) !
2019-09-01 09:07:15.765  INFO 1 --- [           main] o.f.core.internal.command.DbMigrate      : Schema `saconf` is up
to date. No migration necessary.
2019-09-01 09:07:16.028  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing
PersistenceUnitInfo [
        name: default
        ...]
```

Can you say why does this warning appear?

Example 4 - Readiness and Healthiness

Projects 5 and 6 are both related to this example.
Project 5 shows us how to configure readiness and healthiness and allows us to see the difference in deployment compared to the previous examples.

You can deploy project 5 directly by running the **kubectl apply** command with the files in the kube folder.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e5# kubectl apply -f kube
deployment.apps/health-check-app-deployment created
service/health-check-app-service created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e5# kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
health-check-app-deployment-56f55f5df-xs8h8   0/1     Running   0          7s
mysql-7d7fdd478f-wwkvw                      1/1     Running   1          131m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e5#
```

You will first notice that the pod is not alive instantaneously as it happened for the previous examples.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e5# kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
health-check-app-deployment-56f55f5df-xs8h8   1/1     Running   0          74s
mysql-7d7fdd478f-wwkvw                      1/1     Running   1          132m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e5#
```

After a few seconds the app will be up and running. But there are two problems that are usually related to these probes:

1. There is a subtle difference between what healthiness means and what readiness means in regards to an application. It's very improbable that the same logic will apply to both, so calling the same actuator endpoint for both probes could be the sign of a wrong configuration.
2. Consider dependencies when designing the health probe.

With project 6 you can observe one of the consequences of not taking into account dependencies when designing the health endpoint.

Start by updating the secret.yml file with the correct URL:

```
apiVersion: v1
kind: Secret
metadata:
  name: saconf2019-e6-secret
type: Opaque
data:
  spring.datasource.url: mamRiYzpteXNxbDovLzEwLjAuMC4xNDozMDAwMS9zYWNvbmYK
  spring.datasource.username: cm9vdA==
  spring.datasource.password: cGFzc3dvcmQ=
~
```

Then apply the secret, the deployment and the service of the project and wait for the pod to be started.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6# kubectl apply -f kube/secret.yml
secret/saconf2019-e6-secret created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6# kubectl apply -f kube/deployment.yml
deployment.apps/health-check-app-db-deployment created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6# kubectl apply -f kube/service.yml
service/health-check-app-db-service created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6# kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
health-check-app-db-deployment-8494dd789d-rkczs   0/1     Running   0          16s
health-check-app-deployment-56f55f5df-xs8h8       1/1     Running   0          14m
mysql-7d7fdd478f-wwkvw                        1/1     Running   1          145m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6#
```

After a few seconds, when the pod is up, delete the deployment for the mysql database. This simulates what happens if the connection between the application server and the database server is down.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6# kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
health-check-app-db-deployment-8494dd789d-rkczs   1/1     Running   0          88s
health-check-app-deployment-56f55f5df-xs8h8       1/1     Running   0          15m
mysql-7d7fdd478f-wwkvw                        1/1     Running   1          146m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6#
```

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6# kubectl delete deployment mysql
deployment.extensions "mysql" deleted
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6#
```

In just a few seconds the pod will begin to be continuously restarted.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6# kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
health-check-app-db-deployment-8494dd789d-rkczs   0/1     Running   0          3m34s
health-check-app-deployment-56f55f5df-xs8h8       1/1     Running   0          17m
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e6#
```

Example 5 - Having multiple containers for one pod

The basic unit of replication for Kubernetes is the pod. A pod can have multiple containers. It is not necessary a bad idea to have more than one container inside one pod (as proven for example by Istio), but when the containers are divisions of the business logic of the same application, this can lead to a modular monolith.

Project 7 shows how a pod could be configured for having multiple containers. You can play by scaling up and down the pod, but you cannot individually do this for each of the containers.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e7# kubectl apply -f kube
deployment.apps/multiple-containers-deployment created
service/multiple-containers-app created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e7#
```

If you get the pods, you will see that there are 2 containers ready.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e7# kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
multiple-containers-deployment-5779ddb9b9-wwg79   2/2     Running   0          30s
```

And you can use the **kubectl describe** command to see the container definition in the pod.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e7# kubectl describe pod multiple-containers-deployme
nt-5779ddb9b9-wwg79
Name:           multiple-containers-deployment-5779ddb9b9-wwg79
Namespace:      default
Priority:       0
Node:           minikube/10.0.0.14
Start Time:     Sun, 01 Sep 2019 09:43:33 +0000
Labels:         app=multiple-containers-app
                pod-template-hash=5779ddb9b9
Annotations:    <none>
Status:         Running
IP:             172.17.0.4
Controlled By:  ReplicaSet/multiple-containers-deployment-5779ddb9b9
Containers:
  app1:
    Container ID:   docker://b2d9c18719586d591cb2e9095b2502bded450c375d3f84ff942c09cfc15de849
    Image:          laurentiuspilca/saconf2019-e7-1:v1
    Image ID:       docker-pullable://laurentiuspilca/saconf2019-e7-1@sha256:3b8cbca5047ba66be70fe6a16e5e9014cf66fface2
2b1be579bf21108776fef8
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 01 Sep 2019 09:43:38 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-b2vg7 (ro)
  app2:
    Container ID:   docker://89777d5a78653c83442fb75ebe76a7263f8bd23bf23828dc8bfa2c19db9aa8d4
    Image:          laurentiuspilca/saconf2019-e7-2:v1
    Image ID:       docker-pullable://laurentiuspilca/saconf2019-e7-2@sha256:8276b372e91bbec5fe8afe3a2efc60baa7ae27e6a3
447bfa3949a3de6e5c11e5
    Port:           9090/TCP
```

Use the **kubectl scale** command to scale up or down the deployments.

kubectl scale --replicas=3 deployment/<name>

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e7# kubectl scale --replicas=3 deployment/multiple-co
ntainers-deployment
deployment.extensions/multiple-containers-deployment scaled
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e7# kubectl get pods
NAME                                           READY   STATUS    RESTARTS   AGE
multiple-containers-deployment-5779ddb9b9-2s6hr   2/2     Running   0          5s
multiple-containers-deployment-5779ddb9b9-qv8zh   2/2     Running   0          5s
multiple-containers-deployment-5779ddb9b9-wwg79   2/2     Running   0          5m12s
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e7#
```

Example 6 - Autoscaling

Autoscaling is important for your production applications. It helps sparing resources and act to the changes in the number of requests to have a maximum availability.

To allow autoscaling, the metrics server has to be enabled.

To do this run the following steps.

minikube addons enable metrics-server

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# minikube addons enable metrics-server
✅  metrics-server was successfully enabled
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8#
```

git clone https://github.com/kubernetes-incubator/metrics-server.git

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# git clone https://github.com/kubernetes-incubator
/metrics-server.git
Cloning into 'metrics-server'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 11219 (delta 0), reused 0 (delta 0), pack-reused 11216
Receiving objects: 100% (11219/11219), 12.12 MiB | 11.46 MiB/s, done.
Resolving deltas: 100% (5851/5851), done.
Checking connectivity... done.
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8#
```

kubectl apply -f metrics-server/deploy/1.8+

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# kubectl apply -f metrics-server/deploy/1.8+
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io configured
serviceaccount/metrics-server created
deployment.extensions/metrics-server configured
service/metrics-server configured
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8#
```

In a few seconds, the **kubectl top node** command will show you the metrics. You have to wait until the metrics are collected. Until this happens you will see the following error:

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# kubectl top node
error: metrics not available yet
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8#
```

Then the metrics will be displayed like this:

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# kubectl top node
NAME       CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
minikube   155m         7%     1524Mi          10%
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8#
```

In the autoscaler definition you set the threshold for cpu that is used to compute the number of replicas needed. The autoscaler will then take care to have the needed number of pods running according the the usage.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# cat kube/hpa.yml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: ex8-pod-autoscaler
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: autoscaling-app-deployment
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 70
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8#
```

You can apply the configuration files. First you will only see one active pod.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# kubectl apply -f kube/
deployment.apps/autoscaling-app-deployment created
horizontalpodautoscaler.autoscaling/ex8-pod-autoscaler created
service/autoscaling-app created
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
autoscaling-app-deployment-787c468867-dwdr6   1/1     Running   0          6s
```

After a few seconds, the autoscaler will create another pod because the minimum set in the configuration is 2.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8# kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
autoscaling-app-deployment-787c468867-crl7k   1/1     Running   0          51s
autoscaling-app-deployment-787c468867-dwdr6   1/1     Running   0          67s
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e8#
```

If you try to stress the application using a tool like Apache Benchmark you can even see the pods scaling up and down according to the cpu usage.

Appendix 1 - Creating the Docker images

For all the examples provided in this lab the Docker image was already created and stored on a Docker Hub account. To spare time and to avoid repetitions, we did not include the process of creating the Docker image in each example.

But, you can create your own Docker images starting from the source code. This way you can even create your own examples or use various other frameworks and technologies for the development of the applications.

Let's take the first project. Here, and in all the others a Docker file is already present. Its content is very simple, starting from the openjdk:8-jdk-alpine image, using a parameter for the JAR file of the application and then starting the application in the container.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# ls
Dockerfile  kube  mvnw  mvnw.cmd  pom.xml  src
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# cat Dockerfile
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG JAR_FILE
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]root@instance-20190831-1129:/var/codeone
2019kubernetes/saconf2019-e1#
```

So before being able to create the image using the Docker file we first need to build the application. We can do this by using Maven. Also, you need to have a JDK installed.

To install Maven first run the command:

apt-get install maven

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# apt-get install maven
```

To install the JDK run the command:

apt-get install default-jdk

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# apt-get install default-jdk
```

To build the application using Maven run:

mvn clean install

After running the command, the target folder appears. Here you should find the JAR file.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# ls
Dockerfile  kube  mvnw  mvnw.cmd  pom.xml  src  target
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# ls target/
classes                 maven-archiver                   saconf2019-e1-0.0.1-SNAPSHOT.jar.original
generated-sources       maven-status                     surefire-reports
generated-test-sources  saconf2019-e1-0.0.1-SNAPSHOT.jar  test-classes
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1#
```

You can now build the Docker image. Let's first check there is no Docker image yet created.

Run **docker images** to check the current images.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
mysql                                     5.6                 732765f8c7d2        2 weeks ago         257MB
k8s.gcr.io/kube-scheduler                 v1.15.2             88fa9cb27bd2        3 weeks ago         81.1MB
k8s.gcr.io/kube-apiserver                 v1.15.2             34a53be6c9a7        3 weeks ago         207MB
k8s.gcr.io/kube-controller-manager        v1.15.2             9f5df470155d        3 weeks ago         159MB
k8s.gcr.io/kube-proxy                     v1.15.2             167bbf6c9338        3 weeks ago         82.4MB
k8s.gcr.io/metrics-server-amd64           v0.3.3              c6b5d3e48b43        4 months ago        39.9MB
laurentiuspilca/saconf2019-e8             v1                  481d2700373d        7 months ago        119MB
```

Build a new image using the provided Docker file and the JAR you have just built:

docker build . --build-arg=JAR_FILE=target/saconf2019-e1-0.0.1-SNAPSHOT.jar

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# docker build . --build-arg=JAR_FILE=target/saconf
2019-e1-0.0.1-SNAPSHOT.jar
Sending build context to Docker daemon  16.36MB
Step 1/5 : FROM openjdk:8-jdk-alpine
8-jdk-alpine: Pulling from library/openjdk
e7c96db7181b: Pull complete
f910a506b6cb: Pull complete
c2274a1a0e27: Pull complete
Digest: sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3
Status: Downloaded newer image for openjdk:8-jdk-alpine
 ---> a3562aa0b991
Step 2/5 : VOLUME /tmp
 ---> Running in aae00ed25edf
Removing intermediate container aae00ed25edf
 ---> 64898e295452
Step 3/5 : ARG JAR_FILE
 ---> Running in 10aa4e6d71ae
Removing intermediate container 10aa4e6d71ae
 ---> e071feff37f0
Step 4/5 : COPY ${JAR_FILE} app.jar
 ---> 82b5787b20a2
Step 5/5 : ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
 ---> Running in 034779cf5f22
Removing intermediate container 034779cf5f22
 ---> d187a62d7fa3
Successfully built d187a62d7fa3
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1#
```

You can check again the docker images to find the new one created.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
<none>                                    <none>              d187a62d7fa3        4 minutes ago       121MB
mysql                                     5.6                 732765f8c7d2        2 weeks ago         257MB
k8s.gcr.io/kube-scheduler                 v1.15.2             88fa9cb27bd2        3 weeks ago         81.1MB
k8s.gcr.io/kube-apiserver                 v1.15.2             34a53be6c9a7        3 weeks ago         207MB
k8s.gcr.io/kube-controller-manager        v1.15.2             9f5df470155d        3 weeks ago         159MB
```

You can now use **docker tag** to tag your image so that it can be referred from the deployment.yml file.

```
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# docker tag d187a62d7fa3 myapp:v1
root@instance-20190831-1129:/var/codeone2019kubernetes/saconf2019-e1# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
myapp                                     v1                  d187a62d7fa3        6 minutes ago       121MB
mysql                                     5.6                 732765f8c7d2        2 weeks ago         257MB
k8s.gcr.io/kube-proxy                     v1.15.2             167bbf6c9338        3 weeks ago         82.4MB
```

Of course if your intention is to upload it to a Docker repository you will also need to prefix with the repository name.

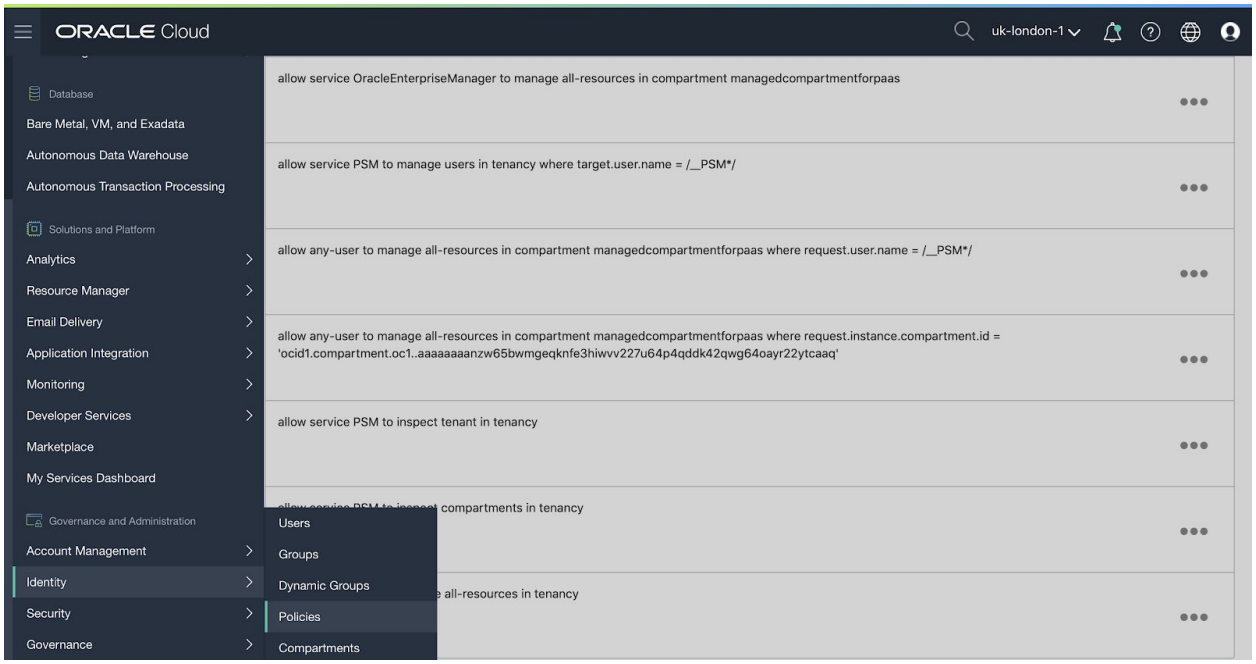
That’s it. Your image can now be referred from deployment.yml file and deployed with Kubernetes.

Appendix 2 - Using OKE for deployment

In a production ready environment you will use a real Kubernetes cluster. With today's cloud services you have plenty of options to do this. For example you could choose to use OKE which is an as-a-service solution to deploy your Kubernetes cluster.

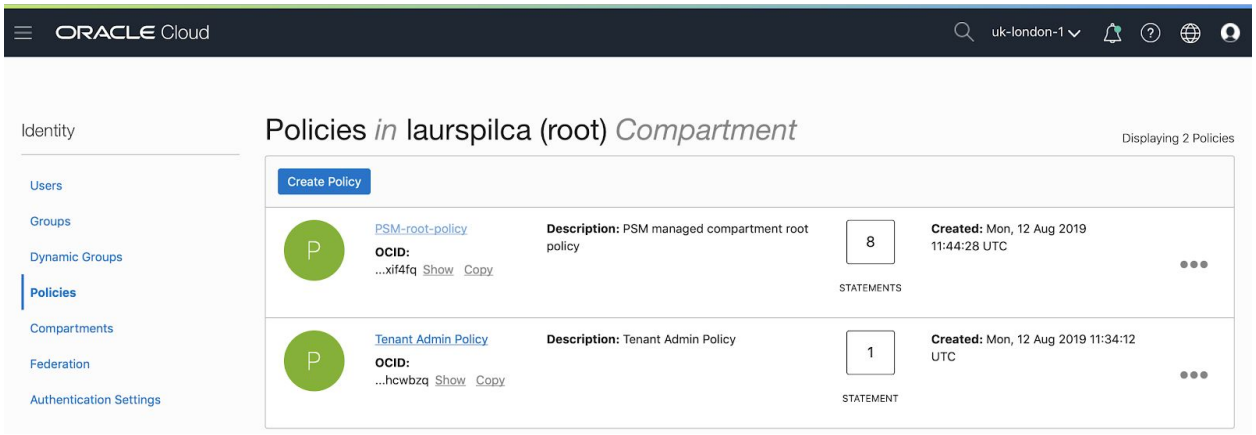
Below you have the steps you need to follow to create your own cluster and deploy your applications.

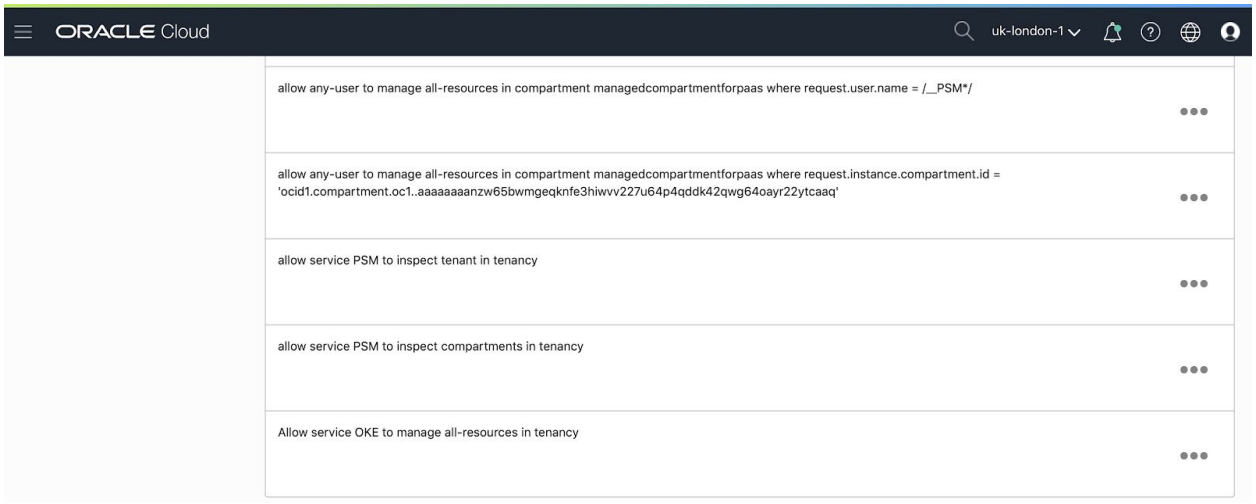
From the left menu, choose Identity -> Policies



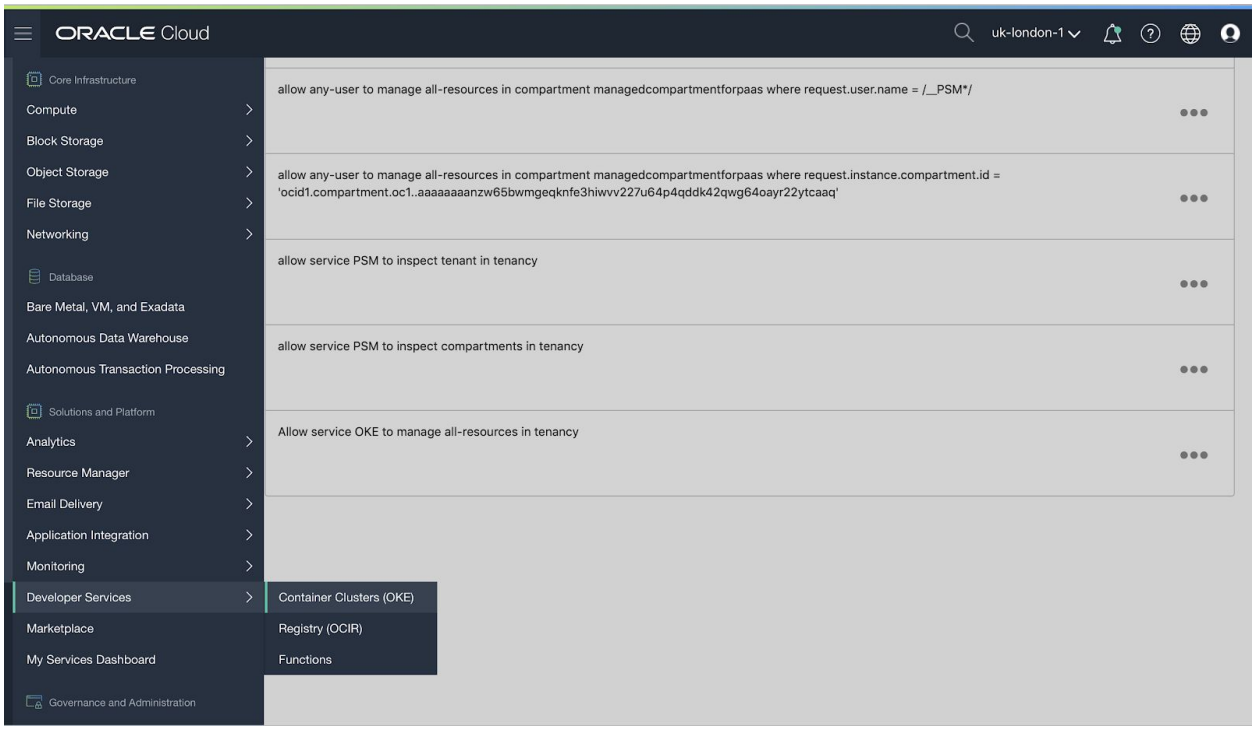
Choose the root policy and make sure you have the following policy statement:

Allow service OKE to manage all-resources in tenancy

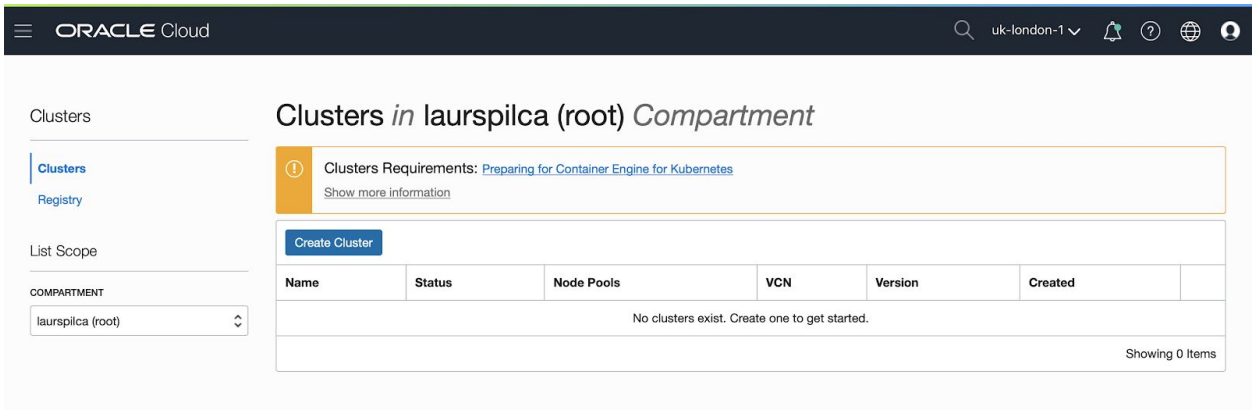

ORACLE Cloud
uk-london-1
Identity
Users
Groups
Dynamic Groups
Policies
Compartments
Federation
Authentication Settings
Policies in laurspilca (root) Compartment
Displaying 2 Policies
Create Policy
P
PSM-root-policy
OCID:
...xif4fq Show Copy
Description: PSM managed compartment root policy
8
STATEMENTS
Created: Mon, 12 Aug 2019 11:44:28 UTC
P
Tenant Admin Policy
OCID:
...hcwbzq Show Copy
Description: Tenant Admin Policy
1
STATEMENT
Created: Mon, 12 Aug 2019 11:34:12 UTC


ORACLE Cloud
uk-london-1
allow any-user to manage all-resources in compartment managedcompartmentforpaas where request.user.name = /__PSM*/
allow any-user to manage all-resources in compartment managedcompartmentforpaas where request.instance.compartment.id = 'ocid1.compartment.oc1..aaaaaaaanzw65bwmgeqknfe3hiwvv227u64p4qddk42qwg64oayr22ytcaaq'
allow service PSM to inspect tenant in tenancy
allow service PSM to inspect compartments in tenancy
Allow service OKE to manage all-resources in tenancy

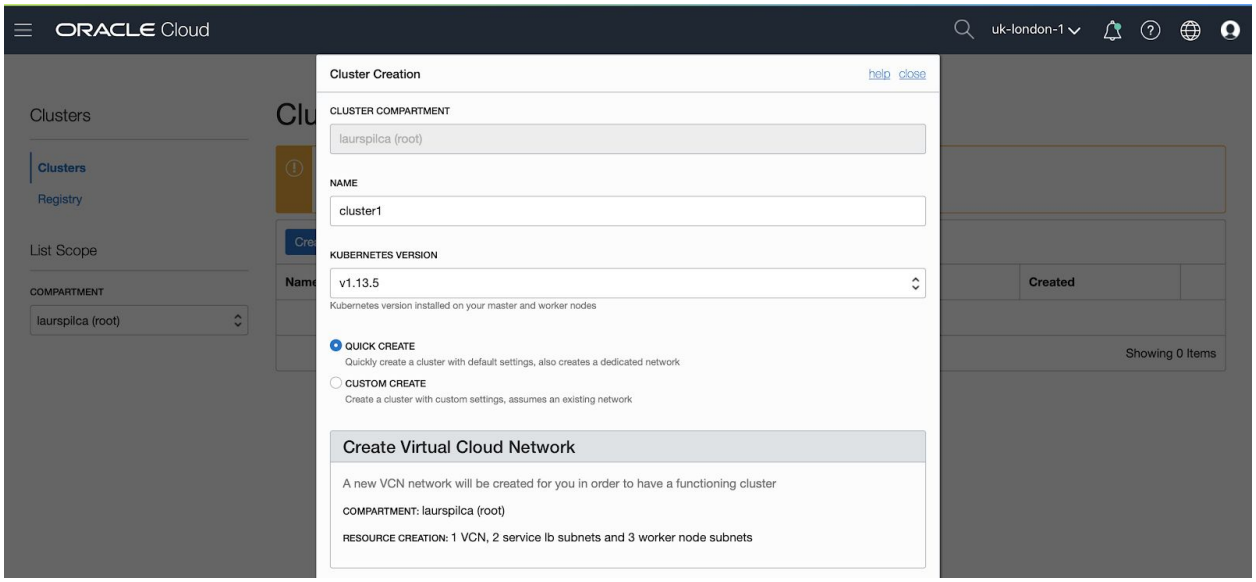
Then, from the left menu you can choose Developer Services -> Container Clusters (OKE)



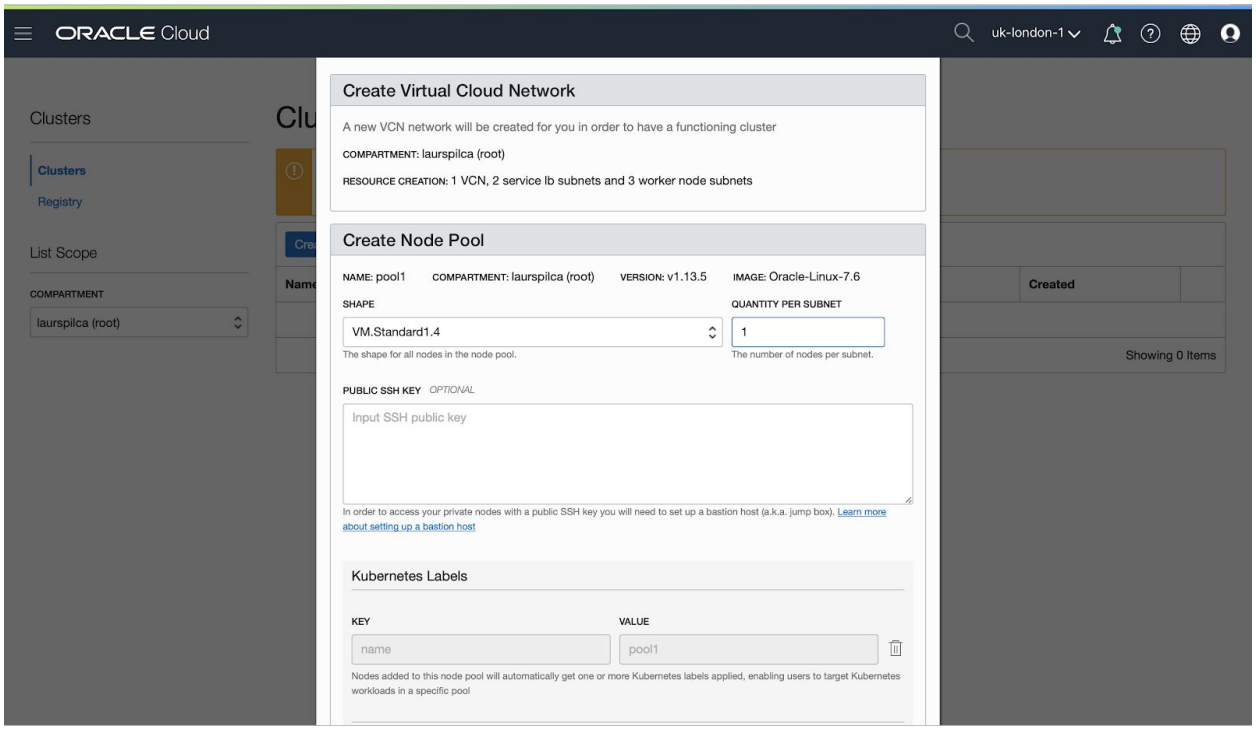
Press the Create Cluster button



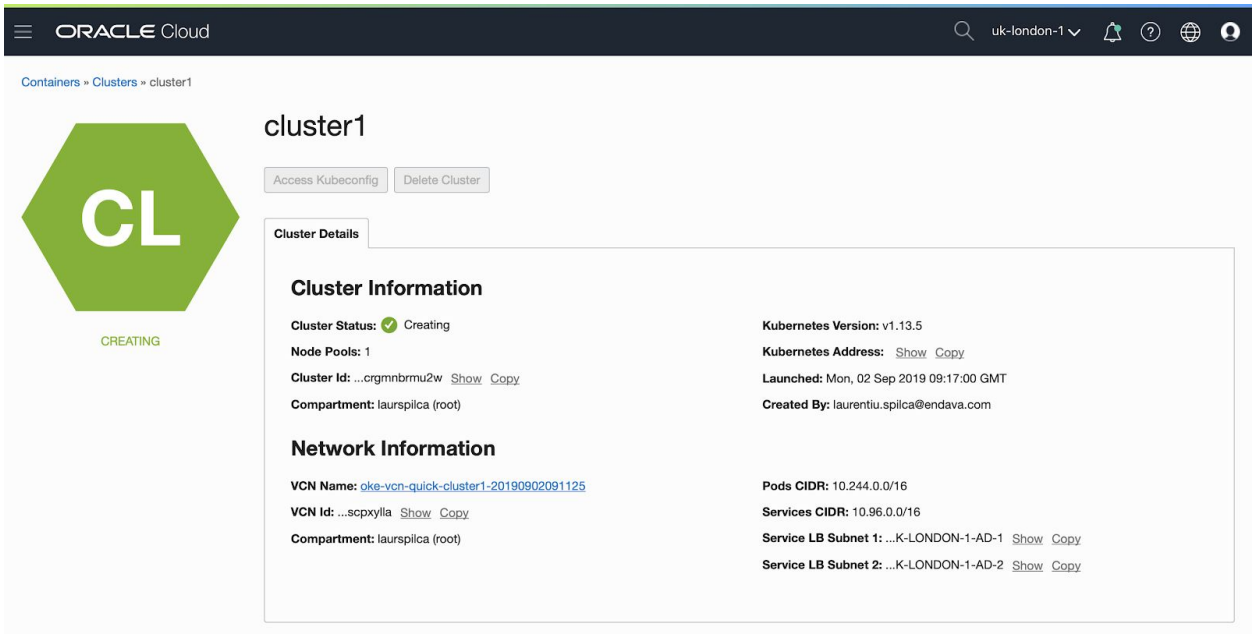
Name your cluster and select the Kubernetes version you wish to deploy. You can choose quick setup to avoid setting up the VCN.



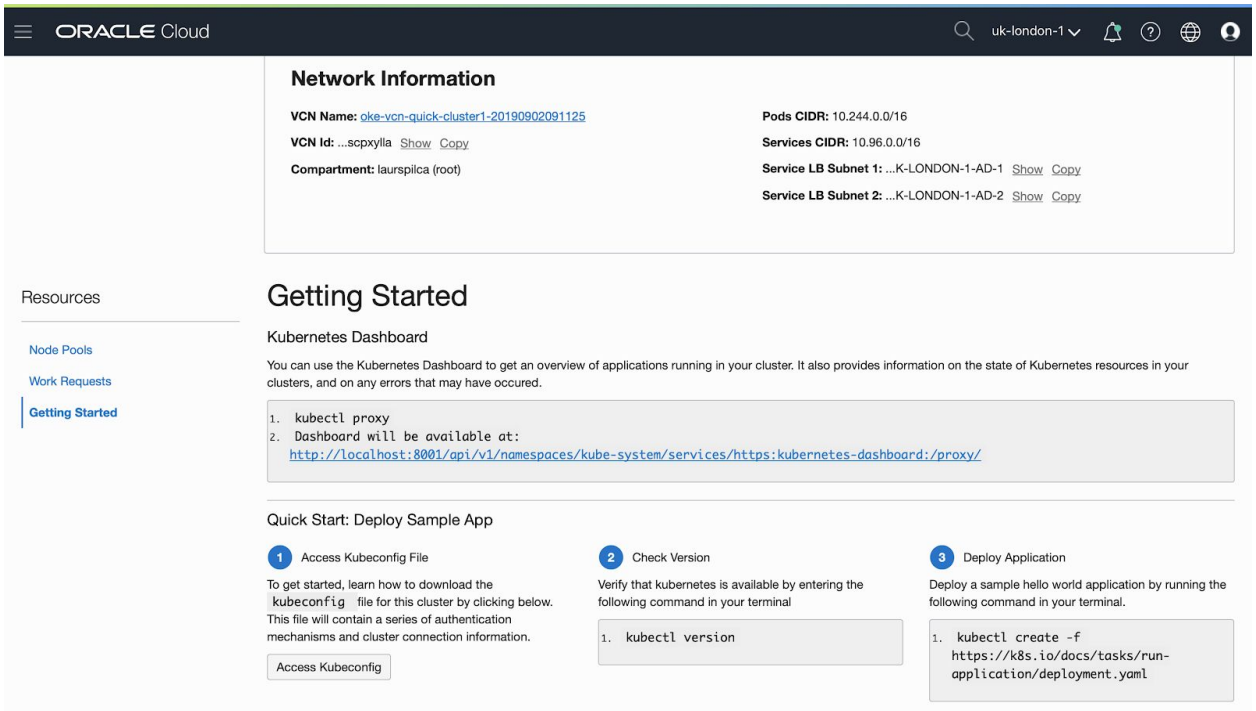
Select the VM shape as well as the number of VMs per each availability zone.



Then create your cluster and wait for the instances to be up.



Once the cluster is up and running, from the left side of the page choose Getting Started and set up the Kubeconfig for your local client.



You Kubernetes cluster is now set up. You can also access it from your local client to deploy the application.

```
EN614620:~ spilca$ oci ce cluster create-kubeconfig --cluster-id ocid1.cluster.oc1.uk-london-1.aaaaaaaaae2dazbqmfrwen3f
hayteytdgvrtkolfmvqwkmtcmcrgmnbrmu2w --file $HOME/.kube/config --region uk-london-1
Existing Kubeconfig file found at /Users/spilca/.kube/config and new config merged into it
EN614620:~ spilca$ kubectl version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.2", GitCommit:"f6278300bebbb750328ac16ee6dd3aa7d3
549568", GitTreeState:"clean", BuildDate:"2019-08-05T09:23:26Z", GoVersion:"go1.12.5", Compiler:"gc", Platform:"darwin/
amd64"}
Server Version: version.Info{Major:"1", Minor:"13+", GitVersion:"v1.13.5-5+270f968ee96a91", GitCommit:"270f968ee96a9166
c3dee050b3f45d213e49a1d5", GitTreeState:"clean", BuildDate:"2019-07-25T04:08:14Z", GoVersion:"go1.11.5", Compiler:"gc",
 Platform:"linux/amd64"}
EN614620:~ spilca$
```