Deploying a Cloud Native App Using Docker and Kubernetes v3

LTRPRG-1200

Speaker: Hector Morales

Learning Objectives

Upon completion of this lab, you will be able to:

- Learn how to deploy a Cloud-Native App using Kubernetes
- Learn the basic Kubernetes components: cluster, deployment, services, pods, labels and scaling.
- Learn how to install a single node Kubernetes cluster and use it for your development on the go

Table of Contents

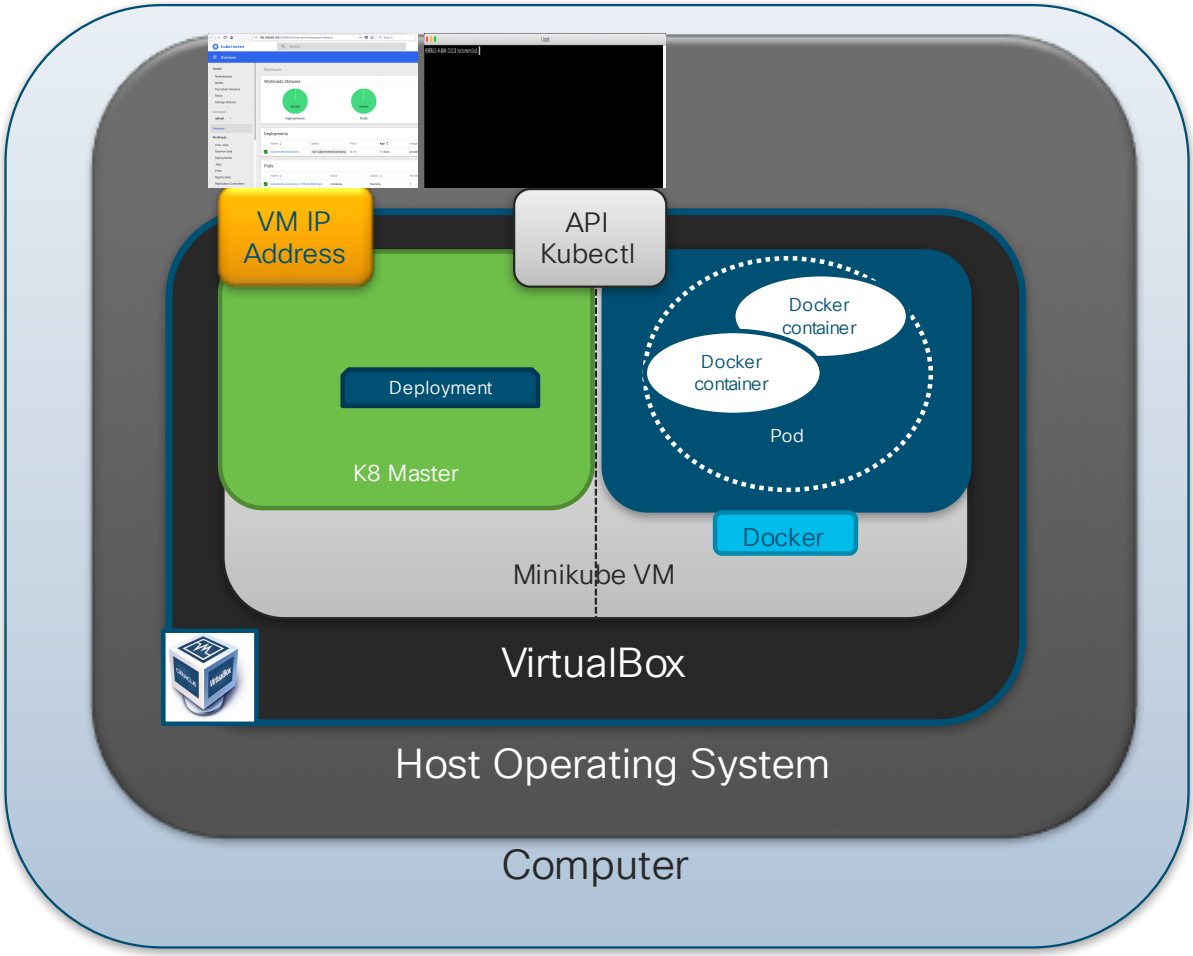
Scenario

In this lab activity, you will learn how to install Minikube, a single-node Kubernetes cluster that you can use to deploy your containerized applications on the move and get all benefits of Kubernetes and Containers.

We will use Docker as the container engine, given that is the most common and it comes natively in Minikube. Using the built-in daemon will help a lot in your development of containerized applications. However, you can use Docker natively.

At the end of the lab you will be able to understand the main Kubernetes concept and you will have deployed your first cloud-native containerized App using Kubernetes.

Here is the scenario that we will use in this lab:



Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop or computer. This gives a great advantage for users looking to try out Kubernetes or develop with it day-to-day. Minikube offers the same baseline code that any other Kubernetes solution.

Requirements:

- A hypervisor –
  - the default hypervisor for Minikube is Virtual Box
  - minikube supports KVM, Hyper-V and Hyperkit
  - minikube can run with no hypervisor as well. There are some caveats.
- Kubectl – is the Kubernetes API server
- Minikube – single-node Kubernetes cluster
- Optionally, Docker installed locally, when Minikube is running with no VM driver (however, Minikube comes with its own Docker engine so no need to install)

You can replicate this lab in your own computer. There are specific instructions for Mac OSX, Windows and Linux, but basically the requirements are exactly the same, so you will get exactly the same functionality and you can replicate this lab at home.

General instructions and notation:

You will have to use the following directory during the lab, when using the Ciscolive laptops:

```
c:\CLUS19\LTRPRG-1200
```

The following conventions are used during the document:

- `c:\CLUS19\LTRPRG-1200\dir  #indicates a directory or command`
- Most of the commands shown in the presentation (PPT) are shown using MAC OSX shell but commands and structure are the same. I have not seen any difference between the Mac OSX version of Kubernetes I used to build the lab, the Windows version that we will use in the lab and Ubuntu version running on dCloud.
- **NOTES are highlighted in bold. Please pay attention and read carefully.**
- **[OPTIONAL]** is used for those instructions during the lab that does not affect the progress and lab goal.
- **TIPS/Recommendations/alternatives** are noted in green bold

IMPORTANT – BEFORE START. Please git clone the following directory locally. You will have the latest files that you will use during the lab

```
cd c:\CLUS19\LTRPRG-1200
git clone https://github.com/hemorale/CLUS19-LTRPRG-1200/
```

Please use the latest version of this guide as there could be some important changes. Also, the latest version of the presentation in PDF will be available.

Lab 1: Installing a Kubernetes Cluster (30 minutes)

Create a Minikube Cluster on Windows – Prerequisites

- Before your start (always good to know!):

```
systeminfo      #This command allows to know if the system is able to support
                virtualization
```

```
Hyper-V Requirements:     VM Monitor Mode Extensions: Yes
                          Virtualization Enabled In Firmware: Yes
                          Second Level Address Translation: Yes
                          Data Execution Prevention Available: Yes
```

If you see the following output, your system already has a Hypervisor installed and you can skip the next step.

```
Hyper-V Requirements:     A hypervisor has been detected. Features required for Hyper-V will not be displayed.
```

- Install hypervisor
  - Virtual Box – **recommended. We will use this one for this lab**
  - Hyper-V – only if you have Windows 10 and you know what you are doing
- Install kubectl

    - Use the same version of your server. Using a newer version may cause errors
- Install minikube (Kubernetes single node installation)
- **[OPTIONAL]** – Install Docker for Windows 10 ONLY. No need to install Docker. We will use Docker inside Minikube.
    - For more information and installation documentation https://store.docker.com/editions/community/docker-ce-desktop-windows
    - Presentation shows instructions how to install it.

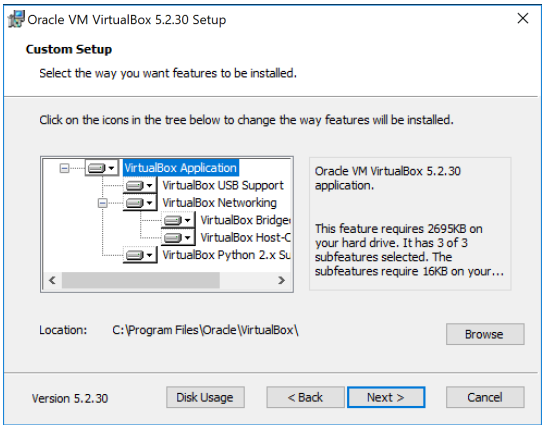
1. Open a Command Line window. **Recommendation: Use Windows Powershell**
2. Change to your working directory, all software images you need will be there and you will work in this directory for the rest of the lab

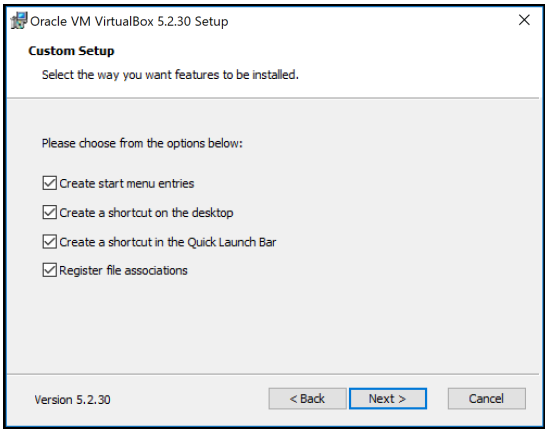
   ```
   cd c:\CLUS19\LTRPRG-1200\
   ```

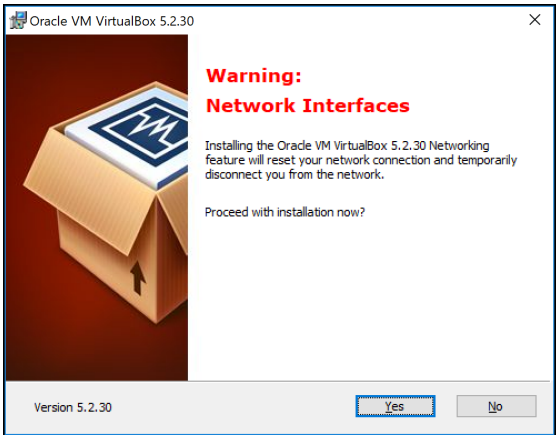
3. Install Virtual Box. Virtual Box is the default hypervisor for Minikube.
4. A Virtual Box executable is already downloaded in your machine. Please run the installer. Alternatively download from here

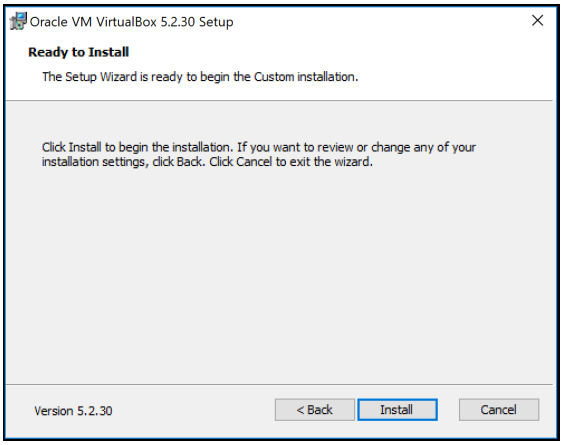
   https://download.virtualbox.org/virtualbox/5.2.30/VirtualBox-5.2.30-130521-Win.exe

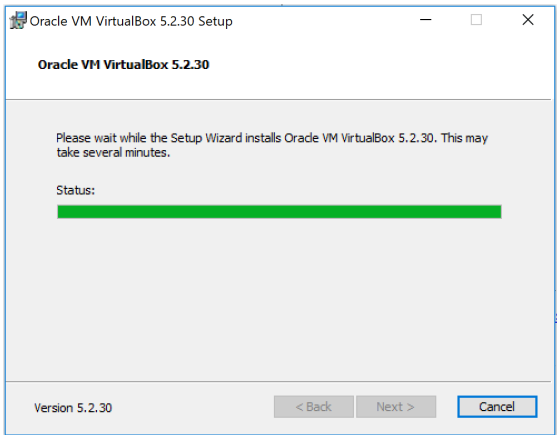
   Click on the link, download the installer and follow the instructions. Similar screens are shown below:


Oracle VM VirtualBox 5.2.30 Setup
Custom Setup
Select the way you want features to be installed.
Click on the icons in the tree below to change the way features will be installed.
VirtualBox Application
VirtualBox USB Support
VirtualBox Networking
VirtualBox Bridge
VirtualBox Host-C
VirtualBox Python 2.x Su
Oracle VM VirtualBox 5.2.30 application.
This feature requires 2695KB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 16KB on your...
Location:
C:\Program Files\Oracle\VirtualBox\
Browse
Version 5.2.30
Disk Usage
< Back
Next >
Cancel


Oracle VM VirtualBox 5.2.30 Setup
Custom Setup
Select the way you want features to be installed.
Please choose from the options below:
Create start menu entries
Create a shortcut on the desktop
Create a shortcut in the Quick Launch Bar
Register file associations
Version 5.2.30
< Back
Next >
Cancel


Oracle VM VirtualBox 5.2.30
Warning:
Network Interfaces
Installing the Oracle VM VirtualBox 5.2.30 Networking feature will reset your network connection and temporarily disconnect you from the network.
Proceed with installation now?
Version 5.2.30
Yes
No



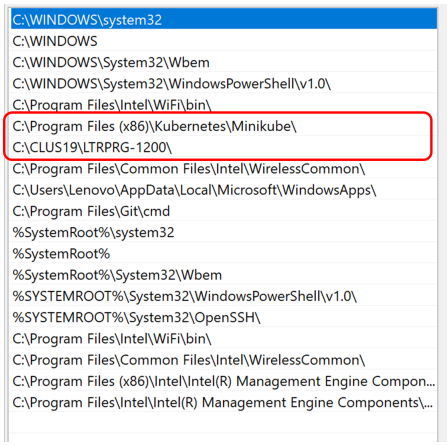


Probably you will get a couple of Windows messages warning of network interfaces and software installation. Click on continue and install

You will get a window like this after installation is complete. Click on finish.

You will get a similar screen when VirtualBox opens after the installation. (Bear with me! Ignore the Spanish version ☺ that was my Windows machine where I tested with this lab.)

5. Download kubectl. There is an executable version of kubectl in your machine. Alternatively download from here: https://storage.googleapis.com/kubernetes-release/release/v1.14.0/bin/windows/amd64/kubectl.exe
6. Kubectl does not need an installer. Just make sure YOUR WORKING DIRECTORY it is in your PATH.



You can use Windows Powershell as well to double check your PATH environment:

```
$env:PATH
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\Lenovo> $env:PATH
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\WiFi\bin\;C:\Pr
ogram Files (x86)\Kubernetes\Minikube\;C:\CLUS19\LTRPRG-1200\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Users\Lenovo\AppData\
Local\Microsoft\WindowsApps\;C:\Program Files\Git\cmd;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsP
owerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Progr
am Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\WINDOWS
\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\WiFi\bin\\;C:\CLUS18\LTRD
EV-1200\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Users\Lenovo\AppData\Local\Microsoft\WindowsApps" /m;C:\Users\Lenovo\AppDa
ta\Local\Microsoft\WindowsApps;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\
PS C:\Users\Lenovo>
```

```
C:\CLUS19\LTRPRG-1200\kubectl.exe
```

NOTE: When you deploy Kubernetes in production, please make sure to use the same version of your server in your local minikube install. Using a newer version may cause errors

NOTE: just run ONE of the following methods to install minikube!

7. Install Minikube for Windows. There are two ways to install Minikube in your Windows machine:
   - **METHOD I –** Using Chocolatey from https://chocolatey.org/ and run:

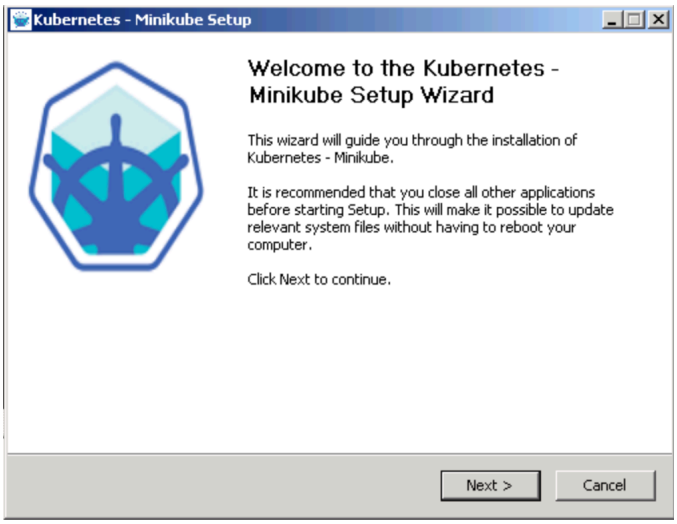
     ```
     C:\choco install minikube
     ```

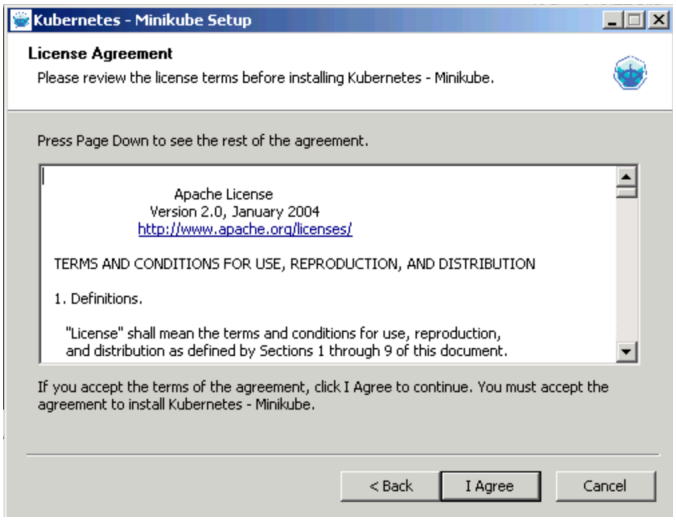
   - **METHOD II –** Installing manually. Download and run the installer https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe

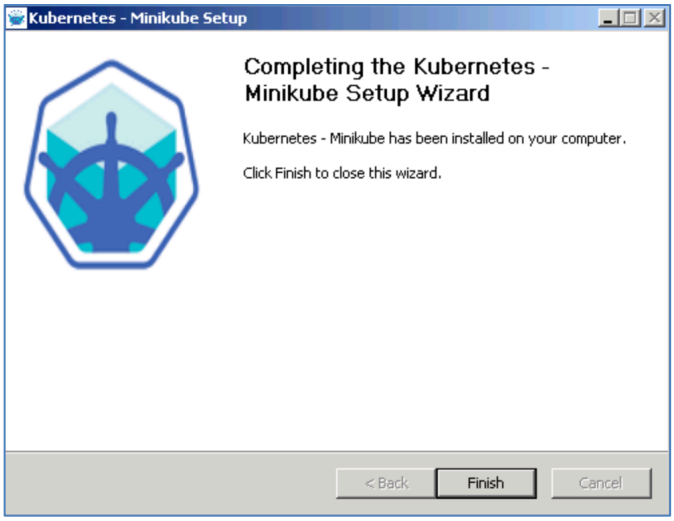
   - **METHOD III –** Installing minikube locally. Download minikube:

     https://github.com/kubernetes/minikube/releases/download/v1.1.1/minikube-windows-amd64.exe
     Rename as minikube.exe and copy to the following directory. If the Minikube directory does not exist, create it:
     `C:\Program Files (x86)\Kubernetes\Minikube`
     Finally add the directory above to the PATH.

8. If you prefer to run the installer, just follow the wizard. It will add the proper name and PATH. You will see a series of similar windows:







7. Make sure your minikube is in your PATH, (Normally, Windows or MAC OSX updates the PATH):

   ```
   C:\Program Files (x86)\Kubernetes\Minikube\
   ```

   NOTE: If you have a previous version of minikube, minikube installer might fail, so you will need to install manually as described above.

Verify that you have installed successfully your Minikube cluster

1. Once you have installed Virtual Box (Windows installation), kubectl and Minikube, you are ready to start your cluster:

   ```
   minikube start   #When VirtualBox is installed locally
   ```

[NOTE] Please reach out your instructor if you have an issue when you are starting minikube
[NOTE] First time you run minikube, it downloads the VM so it will take a bit and you will see a screen like this:

```
PS C:\Users\Lenovo> minikube version
minikube version: v1.1.0
PS C:\Users\Lenovo> minikube start
* minikube v1.1.0 on windows (amd64)
* minikube will upgrade the local cluster from Kubernetes 1.10.0 to 1.14.2
* Downloading Minikube ISO ...
 21.58 MB / 131.28 MB [======>----------------------------------]  16.44% 3m30s
```

2. When using a Minikube, it will install the ISO for the VM, so it will download it. It will take a few minutes. Depending on your computer features and your internet speed it might take between 15 to 25 minutes. Once downloaded, if you see a set of messages like this, you are good to go:

```
PS C:\Users\Lenovo> minikube start
* minikube v1.1.0 on windows (amd64)
* minikube will upgrade the local cluster from Kubernetes 1.10.0 to 1.14.2
* Downloading Minikube ISO ...
 131.28 MB / 131.28 MB [============================================] 100.00% 0s
* Tip: Use 'minikube start -p <name>' to create a new cluster, or 'minikube delete' to delete this one.
* Re-using the currently running virtualbox VM for "minikube" ...
* Waiting for SSH access ...
* Configuring environment for Kubernetes v1.14.2 on Docker 17.12.1-ce
* Downloading kubeadm v1.14.2
* Downloading kubelet v1.14.2
* Pulling images ...
* Relaunching Kubernetes v1.14.2 using kubeadm ...
* Verifying: apiserver proxy etcd scheduler controller dns
* Done! kubectl is now configured to use "minikube"
PS C:\Users\Lenovo>
PS C:\Users\Lenovo>
PS C:\Users\Lenovo>
```

Minikube installed locally on Windows

NOTE: If you are experiencing problems in your local Windows machine after installation, you can use our dCloud Minikube lab. Please go to Annex 2 to get to dCloud. Once you have logged in type the following command:

```
minikube start --vm-driver=none  #when in dCloud
```

```
root@ubuntu:~# minikube start --vm-driver=none
Starting local Kubernetes v1.10.0 cluster...
Starting VM...
Getting VM IP address...
Moving files into cluster...
Downloading kubeadm v1.10.0
Downloading kubelet v1.10.0
Finished Downloading kubelet v1.10.0
Finished Downloading kubeadm v1.10.0
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
===================
WARNING: IT IS RECOMMENDED NOT TO RUN THE NONE DRIVER ON PERSONAL WORKSTATIONS
        The 'none' driver will run an insecure kubernetes apiserver as root that may leave the host vulnerable
to CSRF attacks

When using the none driver, the kubectl config and credentials generated will be root owned and will appear in
the root home directory.
You will need to move the files to the appropriate location and then set the correct permissions.  An example o
f this is below:

        sudo mv /root/.kube $HOME/.kube # this will write over any previous configuration
        sudo chown -R $USER $HOME/.kube
        sudo chgrp -R $USER $HOME/.kube

        sudo mv /root/.minikube $HOME/.minikube # this will write over any previous configuration
        sudo chown -R $USER $HOME/.minikube
        sudo chgrp -R $USER $HOME/.minikube

This can also be done automatically by setting the env var CHANGE_MINIKUBE_NONE_USER=true
Loading cached images from config file.
```

Minikube installed on dCloud

3. In older versions on minikube, after installing it, if you run a minikube status you probably will get an error like the screen shot below:

```
HEMORALE-M-80A4:~ hectormorales$ minikube status
E0425 18:49:39.361798   44634 status.go:85] Error cluster status: Error: Unrecognized output from ClusterStatus:
E0425 18:49:39.362876   44634 util.go:151] Error formatting error message: Error msg with no stack trace cannot be reported
```

4. **NOTE: Run this command ONLY if you get a similar error. Otherwise go to step 5.**

   If you get this error, you need to indicate what cluster you want to use

   ```
   kubectl config use-context minikube
   ```

5. Run the minikube status and check your cluster is running fine.

   ```
   minikube status
   ```

```
PS C:\Users\Lenovo> minikube status
There is a newer version of minikube available (v1.1.1).  Download it here:
https://github.com/kubernetes/minikube/releases/tag/v1.1.1

To disable this notification, run the following:
minikube config set WantUpdateNotification false
host: Running
kubelet: Running
apiserver: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.100
```

6. Now check the status of the Kubernetes Cluster

```
kubectl cluster-info
```

```
PS C:\Users\Lenovo> kubectl cluster-info
Kubernetes master is running at https://192.168.99.100:8443
KubeDNS is running at https://192.168.99.100:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

7. Check your Master and Node versions:

```
minikube version
kubectl version
```

```
PS C:\Users\Lenovo> minikube version
minikube version: v1.1.0
PS C:\Users\Lenovo> kubectl version
Client Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.0", GitCommit:"641856db18352033a0d96dbc99153fa3b27298e5", GitTreeSta
te:"clean", BuildDate:"2019-03-25T15:53:57Z", GoVersion:"go1.12.1", Compiler:"gc", Platform:"windows/amd64"}
Server Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.2", GitCommit:"66049e3b21efe110454d67df4fa62b08ea79a19b", GitTreeSta
te:"clean", BuildDate:"2019-05-16T16:14:56Z", GoVersion:"go1.12.5", Compiler:"gc", Platform:"linux/amd64"}
```

8. View nodes on your Cluster:

```
kubectl get nodes
```

```
PS C:\Users\Lenovo> kubectl get nodes
NAME       STATUS   ROLES    AGE     VERSION
minikube   Ready    <none>   4d16h   v1.14.2
```

Lab 2: Create a Deployment

What is a Deployment?

Kubernetes uses two types of workloads to control your apps. These are Pods and Controllers

- A Controller is a logical function which drives your app on a specific way using a specific set of instructions
- A Deployment is a declarative Kubernetes controller, where you described a desired state of a specific Pod image with specific needs

A deployment controller is the recommended form to run a cloud-native stateless application as it has the following features:

- Declare a desired state of your Pods, including the number of replicas of your App
- Rollback to an earlier version of the Pods
- Scale up the deployment
- Pause the deployment in case your App needs a fix
- Know the status of your deployment and make proper actions/fixes

- Clean up older replicas

Creating a Deployment

NOTE: if you are using dCloud, please change to the working directory and git clone the files

In this lab, we will learn how to create a Deployment file:

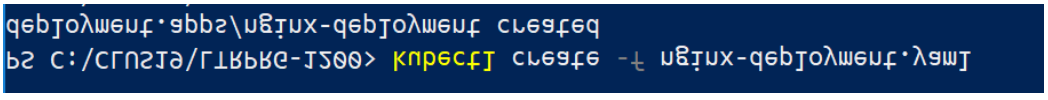
1. Use the deployment file nginx-deployment.yaml or create the file as shown below.

```
touch nginx-deployment.yaml   #or edit in Windows notepad
vi nginx-deployment.yaml      #or edit in Windows notepad
# I recommend use Atom. You can download from https://atom.io

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
   selector:
      matchLabels:
         app: nginx
   replicas: 3
   template:
      metadata:
         labels:
           app: nginx
      spec:
         containers:
          - name: nginx
            image: nginx:1.7.9
            ports:
            - containerPort: 80
```

2. Please check the file is exactly the same as shown in previous step (check spaces at the beginning of each line, otherwise you will get errors)
3. Create the deployment using the following command:

```
kubectl create -f nginx-deployment.yaml
```



4. Verify your deployment status

```
kubectl get deploy      #deployments
kubectl get rs          #replicasets
kubectl rollout status deployment/nginx-deployment
kubectl get pods
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl get deploy
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           3m22s
PS C:\CLUS19\LTRPRG-1200> kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-6dd86d77d    3         3         3       3m36s
PS C:\CLUS19\LTRPRG-1200> kubectl rollout status deployment/nginx-deployment
deployment "nginx-deployment" successfully rolled out
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6dd86d77d-bwgtn    1/1     Running   0          4m31s
nginx-deployment-6dd86d77d-kd9g5    1/1     Running   0          4m31s
nginx-deployment-6dd86d77d-t866r    1/1     Running   0          4m31s
```

5. This is the basics for creating a Deployment. You can create any deployment using this syntax and your own containers. Try this at home.
6. Finally, delete your deployment

```
kubectl delete deployment/nginx-deployment
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl delete deployment/nginx-deployment
deployment.extensions "nginx-deployment" deleted
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
NAME                                READY   STATUS        RESTARTS   AGE
nginx-deployment-6dd86d77d-bwgtn    0/1     Terminating   0          6m20s
nginx-deployment-6dd86d77d-c7m5n    0/1     Terminating   0          4s
nginx-deployment-6dd86d77d-kd9g5    0/1     Terminating   0          6m20s
nginx-deployment-6dd86d77d-t866r    0/1     Terminating   0          6m20s
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
No resources found.
```

Creating a Deployment using the run command

Alternatively, Kubernetes can create a Deployment using the run command (similar to Docker run) from an existing image. We will use this image for the rest of the lab.

1. Run the following command.

```
kubectl run kubernetes-bootcamp --image=gcr.io/google-
samples/kubernetes-bootcamp:v1 --port=8080
```

NOTE: This command will be deprecated in future releases. If you need to know about the command update visit
https://kubernetes.io/docs/reference/kubectl/conventions/

2. The previous command creates a deployment called “kubernetes-bootcamp” from image repository. For the purposes of this lab, we won’t discuss how to create an image but use existing ones for learning purposes. Now, let’s see our newly created deployment and their associated Pods.

```
kubectl get deployments
kubectl get pod
```

TIP – if you get the status *"ImagePullBackOff"* when getting your Pods, most likely, you have a firewall enabled. Change it to allow Private connections.

[dCloud] you can try this command to download the image locally and the run again the kubectl run command:

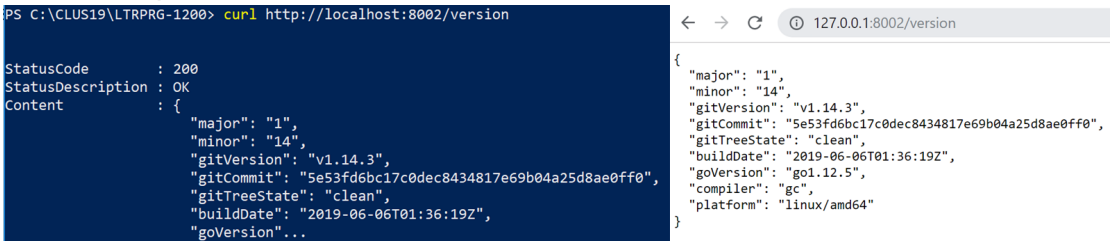
```
docker pull gcr.io/google-samples/kubernetes-bootcamp:v1
```

3. Open another command window and run the following command:

```
kubectl proxy --port 8002          #Default port is 8001
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl proxy --port 8002
Starting to serve on 127.0.0.1:8002
```

4. The previous command allows us to interact directly with the Kubernetes API Server. As we recall, all commands are done via the API. Let's play a bit with it:
   ```
   curl http://localhost:8002/version
   ```
5. In older Windows versions, where there is no curl installed, run directly on the browser. Output is the same!



6. Let's play more. Do the following commands:
   ```
   export POD_NAME=$(kubectl get pod -o
   jsonpath="{.items[0].metadata.name}")
   echo Name of the Pod: $POD_NAME #get the Pod name to query using API
   Server
   curl http://localhost:8002/api/v1/namespaces/default/pods/$POD_NAME/
   ```

   Windows Powershell command:
   ```
   $POD_NAME = (kubectl get pod -o jsonpath="{.items[0].metadata.name}")
   curl http://localhost:8002/api/v1/namespaces/default/pods/$POD_NAME/
   ```

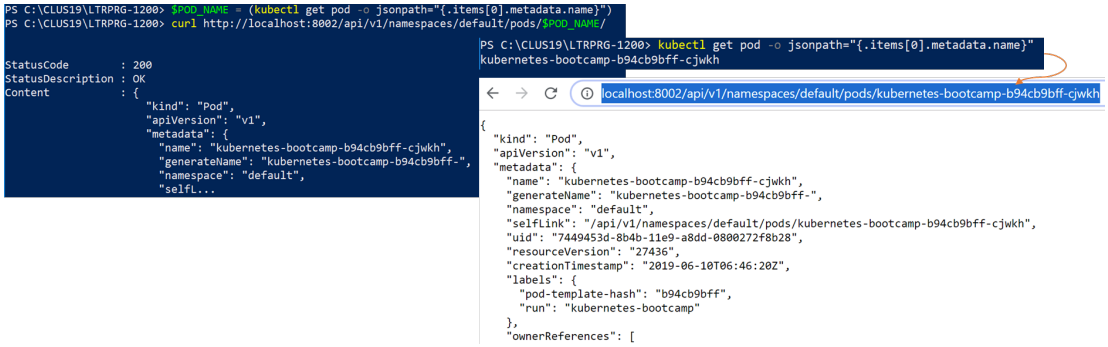
```
PS C:\CLUS19\LTRPRG-1200> $POD_NAME = (kubectl get pod -o jsonpath="{.items[0].metadata.name}")
PS C:\CLUS19\LTRPRG-1200> curl http://localhost:8002/api/v1/namespaces/default/pods/$POD_NAME/


StatusCode        : 200
StatusDescription : OK
Content           : {
                      "kind": "Pod",
                      "apiVersion": "v1",
                      "metadata": {
                        "name": "kubernetes-bootcamp-b94cb9bff-cjwkh",
                        "generateName": "kubernetes-bootcamp-b94cb9bff-",
                        "namespace": "default",
                        "selfL...
```

7. Alternatively, copy the output of the command:

   ```
   kubectl get pod -o jsonpath="{.items[0].metadata.name}"
   ```

8. And use it for the curl command or in your browser (instead of the $POD_NAME). You will see a similar output:



Lab 3: Viewing Pods and Nodes

Pod is the smallest and simplest Kubernetes unit, which runs on the Kubernetes nodes. A Pod represents a running process:

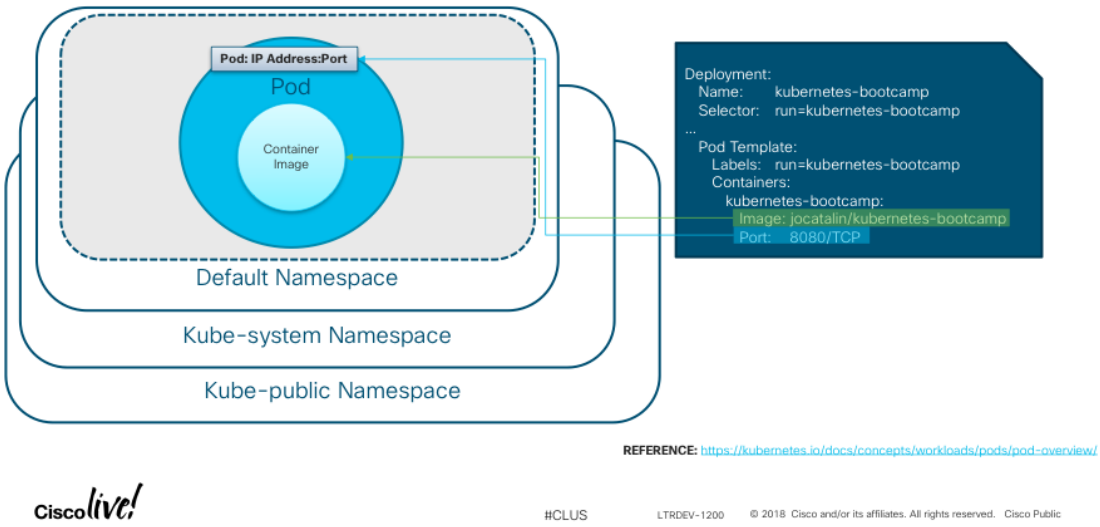
- Container or containers
- Storage resources
- An IP Address to access these resources
- A mechanism that indicates how your container should run

In a few words, the Pod is where you place your app, manage app resources and the unit you manage. We can see the Pod in Kubernetes is the equivalent to a Container in Docker.

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called **namespaces**.

Nodes are the hosts where Namespaces allocates Pods. Namespaces are Virtual Hosts that can be used for multiple Pod operations (we won't discuss Namespaces in this lab).

NOTE: For more information on namespaces, you can visit
https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/



The previous image shows the relationship between the Pod and the Deployment we recently created. The Deployment is like the recipe and the Pod is the actual cake ☺

Also, we can see how the namespaces play in Kubernetes. Namespaces can be seen as Virtual Hosts where Pods are running. A user can create a specific namespace for their own containers, as Namespaces share the same IP addressing. In this context, all Pods within the same namespace are visible via IP from other Pods in the same namespace.

Let's run a few commands to understand better how a Pod works:

1. Let's look for existing Pod with an extended view:
   ```
   kubectl get pods
   kubectl get pods -o wide
   ```

```
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-b94cb9bff-cjwkh   1/1     Running   0          11h
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-b94cb9bff-cjwkh   1/1     Running   0          11h   172.17.0.5   minikube   <none>           <none>
```

2. The difference between the commands is that the second provides an extended output, in this case, we are showing the INTERNAL IP address of the Pod within the namespace. If we would have other Pods, we will see the IP address within the same address space (wait for that lab!). Let's take a deeper look:
   ```
   kubectl get pods --namespace=default
   kubectl get pods --namespace=kube-system
   kubectl describe pods
   ```

```
PS C:\Users\Lenovo> kubectl get pods --namespace=default
NAME                                READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running   0          15m
PS C:\Users\Lenovo> kubectl get pods --namespace=kube-system
NAME                                 READY   STATUS    RESTARTS   AGE
coredns-fb8b8dccf-8ktlg              1/1     Running   1          36m
coredns-fb8b8dccf-hwfwm              1/1     Running   1          36m
etcd-minikube                        1/1     Running   0          35m
kube-addon-manager-minikube          1/1     Running   0          35m
kube-apiserver-minikube              1/1     Running   0          35m
kube-controller-manager-minikube     1/1     Running   0          35m
kube-proxy-5qwvj                     1/1     Running   0          36m
kube-scheduler-minikube              1/1     Running   0          35m
kubernetes-dashboard-d7c9687c7-jqvgr   1/1     Running   2          35m
storage-provisioner                  1/1     Running   0          35m
PS C:\Users\Lenovo> kubectl describe pods
Name:               kubernetes-bootcamp-b94cb9bff-2lm54
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               minikube/10.0.2.15
Start Time:         Mon, 10 Jun 2019 19:23:53 -0500
Labels:             pod-template-hash=b94cb9bff
                    run=kubernetes-bootcamp
Annotations:        <none>
Status:             Running
IP:                 172.17.0.5
Controlled By:      ReplicaSet/kubernetes-bootcamp-b94cb9bff
```

As we could see, we can inspect existing Pods in different namespaces. In this case, the default and kube-system. And also, we can inspect in detail what the Pod contains. Things like node, node IP, pod IP, container image and container port, can be seen.

Let's go a bit deeper into the Pod. As you could notice, our Pod is running a Docker Container, so it is possible to actually access to it. Let's play:

3. Run a curl command or type the following in your browser:
   curl
   http://localhost:8002/api/v1/namespaces/default/pods/$POD_NAME/proxy/
   See the output:

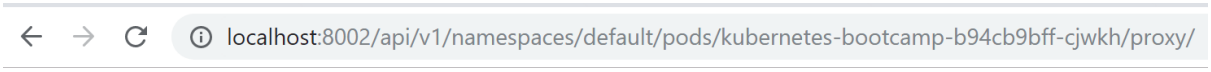
```
PS C:\CLUS19\LTRPRG-1200> curl http://localhost:8002/api/v1/namespaces/default/pods/$POD_NAME/proxy/


StatusCode        : 200
StatusDescription : OK
Content           : Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-b94cb9bff-cjwkh | v=1

RawContent        : HTTP/1.1 200 OK
                    Content-Length: 83
                    Content-Type: text/plain
                    Date: Mon, 10 Jun 2019 18:27:23 GMT

                    Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-b94cb9bff-cjwkh | v=1

Forms             : {}
Headers           : {[Content-Length, 83], [Content-Type, text/plain], [Date, Mon, 10 Jun 2019 18:27:23 GMT]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 83
```



NOTE: In order to get the response in your browser, remember you need your actual Pod Name (not the variable!)

4. Something is producing that output. Where is it? Let's play more:
   `kubectl logs $POD_NAME`

```
PS C:\CLUS19\LTRPRG-1200> kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2019-06-10T06:46:22.067Z | Running On:  kubernetes-bootcamp-b94cb9bff-cjwkh

Running On: kubernetes-bootcamp-b94cb9bff-cjwkh | Total Requests: 1 | App Uptime: 42061.463 seconds | Log Time: 2019-06-10T18:27:23.531Z
Running On: kubernetes-bootcamp-b94cb9bff-cjwkh | Total Requests: 2 | App Uptime: 42230.029 seconds | Log Time: 2019-06-10T18:30:12.096Z
```

   Anything that the application would normally send to STDOUT becomes logs for the container within the Pod. We can retrieve these logs using the kubectl logs command.

5. Let's login into the container:
   ```
   kubectl exec $POD_NAME env
   kubectl exec -ti $POD_NAME bash
   cat server.js
   curl localhost:8080
   exit
   ```

   As we could see, we were able to execute a few commands into the actual Docker container. The first one, returned the environment variables. The second one, logged us into the container and we could see the actual web server code and run locally.

```
PS C:\CLUS19\LTRPRG-1200> kubectl exec $POD_NAME env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=kubernetes-bootcamp-b94cb9bff-cjwkh
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
NPM_CONFIG_LOGLEVEL=info
NODE_VERSION=6.3.1
HOME=/root
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl exec -ti $POD_NAME bash
root@kubernetes-bootcamp-b94cb9bff-cjwkh:/# cat server.js
var http = require('http');
var requests=0;
var podname= process.env.HOSTNAME;
var startTime;
var host;
var handleRequest = function(request, response) {
  response.setHeader('Content-Type', 'text/plain');
  response.writeHead(200);
  response.write("Hello Kubernetes bootcamp! | Running on: ");
  response.write(host);
  response.end(" | v=1\n");
  console.log("Running On:" ,host, "| Total Requests:", ++requests,"| App Uptime:", (new Date(
:",new Date());
}
var www = http.createServer(handleRequest);
www.listen(8080,function () {
    startTime = new Date();;
    host = process.env.HOSTNAME;
    console.log ("Kubernetes Bootcamp App Started At:",startTime, "| Running On: " ,host, "\n"
});
root@kubernetes-bootcamp-b94cb9bff-cjwkh:/# curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-b94cb9bff-cjwkh | v=1
root@kubernetes-bootcamp-b94cb9bff-cjwkh:/# exit
exit
```

We are inside the container!!

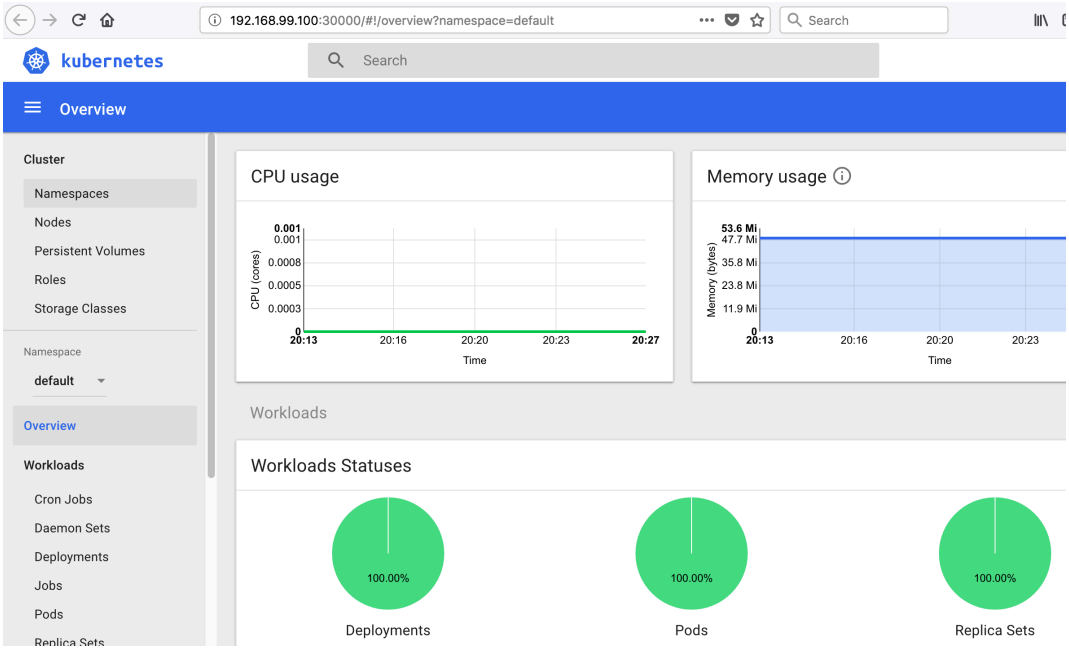
6. Now that we have a better understanding of Pods and Nodes, let's take a look to our actual cluster. Let's run a few commands:

   `minikube service list`

```
PS C:\CLUS19\LTRPRG-1200> minikube service list
|-------------|----------------------|--------------|
|  NAMESPACE  |         NAME         |     URL      |
|-------------|----------------------|--------------|
| default     | kubernetes           | No node port |
| kube-system | kube-dns             | No node port |
| kube-system | kubernetes-dashboard | No node port |
|-------------|----------------------|--------------|
```

You will see a similar output. As can be seen, there are namespaces, the “virtual machines” where Minikube services are running, and an associated URL. Let’s review in the next lab what this URL means. Meanwhile try this command and play a little bit with the interface:

```
minikube dashboard
```

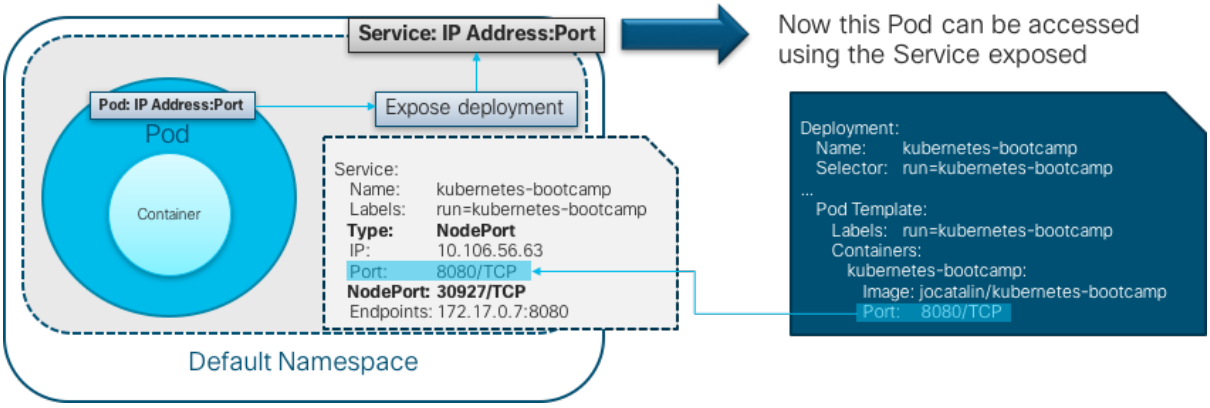


Lab 4: Using Services, Labels and Locators

We learned Pods that are running inside Kubernetes are running on a private, isolated network, i.e. within the namespace.
When we use kubectl, we're interacting through an API endpoint to communicate with our application. Kubectl is the API agent that minikube uses to communicate the server with the node (remember the Kubernetes basics).

We also learned that the API server will automatically create an endpoint for each pod, based on the pod name, that is also accessible through the Kubectl proxy.

How Pods, for our case, our App, is able to communicate with the external world? We will learn this during this lab.



Kubernetes uses the Services abstraction to expose the Pod to the external world. There are some Service Types that can be used:

- Cluster IP
- Node Port
- Load Balancer

Minikube supports these services. So, let's understand better.

1. We know we have one pod running and its IP Address. Let's find out how many services we have:

```
kubectl get pods -o wide
kubectl get services
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -o wide
NAME                                   READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-b94cb9bff-cjwkh   1/1     Running   0          12h   172.17.0.5   minikube   <none>           <none>
PS C:\CLUS19\LTRPRG-1200> kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   6d17h
```

As we can see, we have only one Service, which was created by Minikube to create the Cluster. But how can we expose our Pod via a Service. Let's continue

2. Let's create a new Service using our current Deployment. This service will expose our Pod to the external world using port 8080

```
kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port
8080
kubectl get services
```

What we did is to expose our Pod via NodePort (Port NAT) from an internal 8080 port in our Pod to an external port chosen by the cluster. You will see something similar:

```
PS C:\CLUS19\LTRPRG-1200> kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080
service/kubernetes-bootcamp exposed
PS C:\CLUS19\LTRPRG-1200> kubectl get services
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes            ClusterIP   10.96.0.1       <none>        443/TCP          6d17h
kubernetes-bootcamp   NodePort    10.103.200.60   <none>        8080:32499/TCP   10s
```

3. Let's take a look to the new created services

```
kubectl describe services/kubernetes-bootcamp

Windows Powershell commands:
$NODE_PORT = (kubectl get services/kubernetes-bootcamp -o
jsonpath="{.spec.ports[0].nodePort}")
$NODE_PORT

LINUX commands:
export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o
jsonpath="{.spec.ports[0].nodePort}")

echo NODE_PORT=$NODE_PORT
```

4. Let's use this Port Number and run the following command:

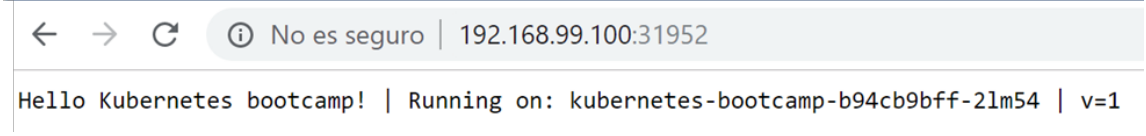
```
Windows PowerShell commands:
minikube ip       #Get Minikube IP
$NODE_PORT        #Get Minikube IP

curl [minikube_ip]:[$NODE_PORT]     #Type in ip_address and port
```

Alternatively, you can run the following command and use it in your browser. Please include the Port number you just get from alternative command in step 3

```
PS C:\Users\Lenovo> curl 192.168.99.100:31952


StatusCode        : 200
StatusDescription : OK
Content           : Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-b94cb9bff-2lm54 | v=1
```



5. Let's understand what a Deployment means. If we type

```
kubectl describe deployments
```

We see a bunch of details regarding our Pod. So, a deployment is the actual Pod description and the way it has to work within our cluster. See the label in the Pod

```
Labels:  run=kubernetes-bootcamp
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl describe deployments
Name:                   kubernetes-bootcamp
Namespace:              default
CreationTimestamp:      Mon, 10 Jun 2019 19:23:53 -0500
Labels:                 run=kubernetes-bootcamp
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               run=kubernetes-bootcamp
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=kubernetes-bootcamp
  Containers:
   kubernetes-bootcamp:
    Image:        gcr.io/google-samples/kubernetes-bootcamp:v1
    Port:         8080/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   kubernetes-bootcamp-b94cb9bff (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  39m   deployment-controller  Scaled up replica set kubernetes-bootcamp-b94cb9bff to 1
```

6. Run the following commands:

```
kubectl get pods -l run=kubernetes-bootcamp
kubectl get services -l run=kubernetes-bootcamp
```

You will see a similar output like the one below. What we are doing is using the label called run=kubernetes-bootcamp to invoke a pod and an associated service. Imagine a cluster with hundreds of Pods. Using Labels help us to create a good way to identify workloads like environment=production or tier=front end.

```
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -l run=kubernetes-bootcamp
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running   0          44m
PS C:\CLUS19\LTRPRG-1200> kubectl get services -l run=kubernetes-bootcamp
NAME                  TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes-bootcamp   NodePort   10.98.92.163   <none>        8080:31952/TCP   14m
```

7. Let's apply a new label to our Pod. Run the following commands:
   `kubectl label pod $POD_NAME app=v1`

   You will see a similar output:

```
PS C:\CLUS19\LTRPRG-1200> kubectl label pod $POD_NAME app=v1
pod/kubernetes-bootcamp-b94cb9bff-2lm54 labeled
```

   Let's explore again our Pod:
   `kubectl describe pods $POD_NAME     #Or copy your Pod Name`

   We can now list our Pod using the label we added:
   `kubectl get pods -l app=v1`

   You will see a similar output:

```
PS C:\CLUS19\LTRPRG-1200> kubectl describe pods $POD_NAME
Name:               kubernetes-bootcamp-b94cb9bff-2lm54
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               minikube/10.0.2.15
Start Time:         Mon, 10 Jun 2019 19:23:53 -0500
Labels:             app=v1
                    pod-template-hash=b94cb9bff
                    run=kubernetes-bootcamp
Annotations:        <none>
Status:             Running
IP:                 172.17.0.5
Controlled By:      ReplicaSet/kubernetes-bootcamp-b94cb9bff
Containers:
  kubernetes-bootcamp:
    Container ID:   docker://ddb7eebdf694c1ee3ff8042ed40ec37f
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -l app=v1
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running   0          49m
```

8. We can do several operations using labels as explained in 6. Let's play a little more. Let's delete the service we created and run a few commands:

```
kubectl delete service -l run=kubernetes-bootcamp
kubectl get services
minikube ip       #Get Minikube IP
$NODE_PORT        #Get Minikube IP

curl [minikube_ip]:[$NODE_PORT]     #Type in ip_address and port
```

Let's run this one, what happens?:

```
kubectl exec -ti $POD_NAME curl localhost:8080
```

You will get similar outputs:

```
PS C:\CLUS19\LTRPRG-1200> curl 192.168.99.100:31952
curl :  Can't connect to remote server

+ curl 192.168.99.100:31952
+ ~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl exec -ti $POD_NAME curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-b94cb9bff-2lm54 | v=1
```

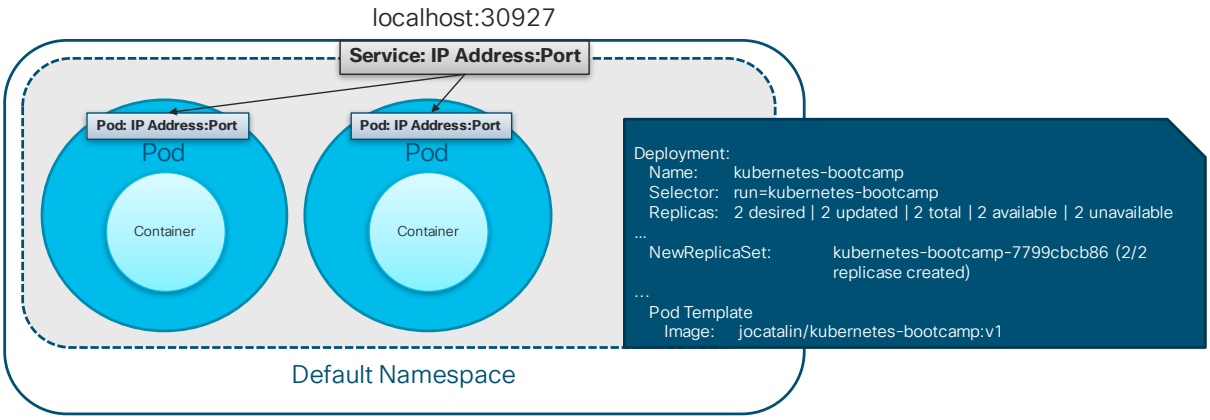
Lab 5: Scaling a Deployment

So far, we have learned about:

- Deployments
- Pods and Nodes
- Services and Labels

In our current scenario, we have only one Pod instance, but as we know, this isn't the right scenario for cloud-native or web-scale type of applications. We typically need more than one instance per each of our containers and we need to have a mechanism that controls scale.

We learned that Deployments is the recipe where we indicate how many REPLICAS we need. So, we'll learn in this lab, how to do it.



1. Let's take a look a little deeper to our current deployment

   ```
   kubectl get deployments
   ```

   You should have something like this output:

   ```
   PS C:\CLUS19\LTRPRG-1200> kubectl get deployments
   NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
   kubernetes-bootcamp   1/1     1            1           66m
   ```

   We can see that our deployment has 1 Pod and the command provides this information:

   - DESIRED = number of desired replicas
   - CURRENT = how many replicas are running
   - UP-TO-DATE = number of replicas updated to get to the desired state
   - AVAILABLE = how many replicas are available to be used

   You may check your pod as well and you will see it is in RUNNING State

   ```
   kubectl get pods
   kubectl get pods -o wide
   ```

   ```
   PS C:\CLUS19\LTRPRG-1200> kubectl get pods
   NAME                                  READY   STATUS    RESTARTS   AGE
   kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running   0          68m
   PS C:\CLUS19\LTRPRG-1200> kubectl get pods -o wide
   NAME                                  READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
   kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running   0          68m   172.17.0.5   minikube   <none>           <none>
   ```

2. In order to increase the number of running instances of your app, we need to increase the number of Replicas in your Deployment. Let's increase the number of replicas to 4 in the Deployment. We will use the kubectl scale command:

   ```
   kubectl scale deployments/kubernetes-bootcamp --replicas=4
   ```

3. Run also again these commands for a few times to see the changes on the deployment and pods

   ```
   kubectl get deployments
   kubectl get pods
   kubectl get pods -o wide
   ```

   You will see a similar output:

```
PS C:\CLUS19\LTRPRG-1200> kubectl scale deployments/kubernetes-bootcamp --replicas=4
deployment.extensions/kubernetes-bootcamp scaled
PS C:\CLUS19\LTRPRG-1200> kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   4/4     4            4           70m
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running   0          70m
kubernetes-bootcamp-b94cb9bff-7dhnt   1/1     Running   0          14s
kubernetes-bootcamp-b94cb9bff-p7sfc   1/1     Running   0          14s
kubernetes-bootcamp-b94cb9bff-pkwxl   1/1     Running   0          13s
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running   0          70m   172.17.0.5   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-7dhnt   1/1     Running   0          20s   172.17.0.6   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-p7sfc   1/1     Running   0          20s   172.17.0.7   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-pkwxl   1/1     Running   0          19s   172.17.0.8   minikube   <none>           <none>
```

   You can check the detail of the deployment, using the kubectl describe deployment command

   ```
   kubectl describe deployments/kubernetes-bootcamp
   ```

   See the Replicas line and check it is updated to 4 replicas

```
PS C:\CLUS19\LTRPRG-1200> kubectl describe deployments/kubernetes-bootcamp
Name:                   kubernetes-bootcamp
Namespace:              default
CreationTimestamp:      Mon, 10 Jun 2019 19:23:53 -0500
Labels:                 run=kubernetes-bootcamp
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               run=kubernetes-bootcamp
Replicas:               4 desired | 4 updated | 4 total | 4 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=kubernetes-bootcamp
  Containers:
   kubernetes-bootcamp:
    Image:        gcr.io/google-samples/kubernetes-bootcamp:v1
    Port:         8080/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Progressing    True    NewReplicaSetAvailable
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   kubernetes-bootcamp-b94cb9bff (4/4 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  2m9s  deployment-controller  Scaled up replica set kubernetes-bootcamp-b94cb9bff to 4
```


4. Let's recreate the service we deleted in the last step of the previous lab

```
kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port
8080
kubectl get services
```

```
HEMORALE-M-80A4:~ hectormorales$ kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080
service "kubernetes-bootcamp" exposed
HEMORALE-M-80A4:~ hectormorales$ kubectl get services
NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes            ClusterIP   10.96.0.1      <none>        443/TCP          7d
kubernetes-bootcamp   NodePort    10.106.65.63   <none>        8080:30927/TCP   3s
```

5. Now, let's update the NODE_PORT variable with the new port associated to the service just created

Windows Powershell commands:

```
$NODE_PORT = (kubectl get services/kubernetes-bootcamp -o
jsonpath="{.spec.ports[0].nodePort}")
minikube ip
$NODE_PORT
curl [minikup_ipaddr]:[port]
```

Linux commands:

```
export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o
jsonpath="{.spec.ports[0].nodePort}")
$echo NODE_PORT=$NODE_PORT
```

6. Now, let's decrease the number of replicas to 2 in the Deployment:

```
kubectl scale deployments/kubernetes-bootcamp --replicas=2
kubectl get pods
kubectl get pods -o wide
```

You will get a similar output:

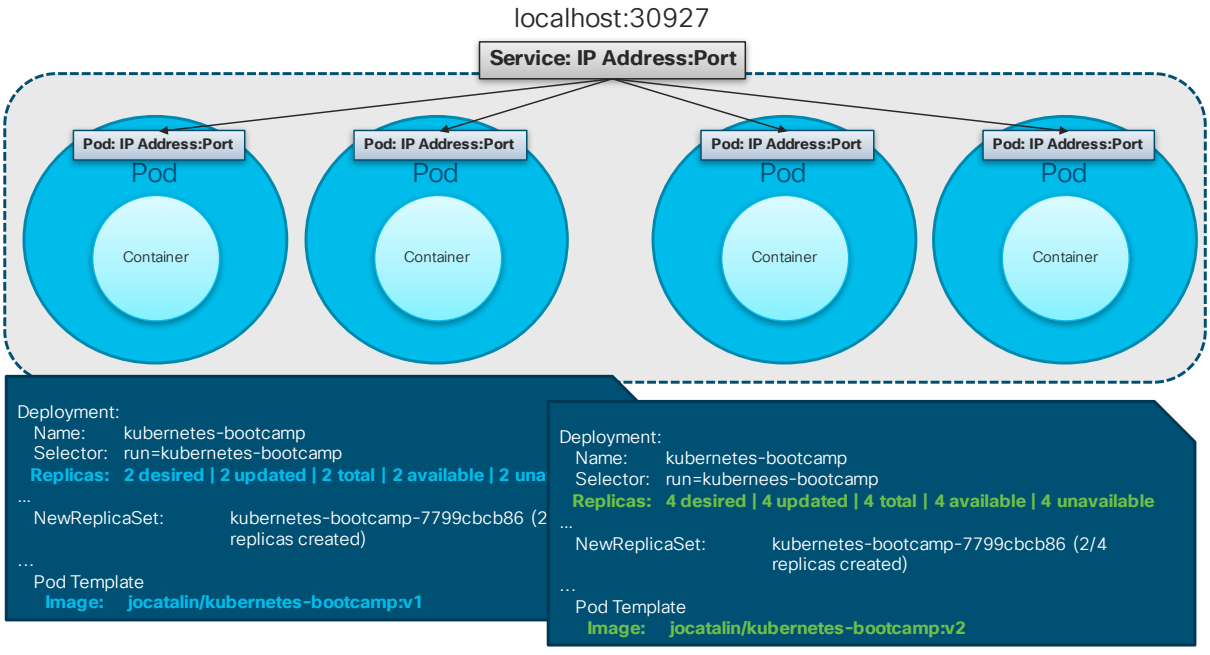
```
PS C:\CLUS19\LTRPRG-1200> kubectl scale deployments/kubernetes-bootcamp --replicas=2
deployment.extensions/kubernetes-bootcamp scaled
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
NAME                                  READY   STATUS        RESTARTS   AGE
kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running       0          88m
kubernetes-bootcamp-b94cb9bff-7dhnt   1/1     Running       0          18m
kubernetes-bootcamp-b94cb9bff-p7sfc   1/1     Terminating   0          18m
kubernetes-bootcamp-b94cb9bff-pkwxl   1/1     Terminating   0          18m
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -o wide
NAME                                  READY   STATUS        RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-b94cb9bff-2lm54   1/1     Running       0          88m   172.17.0.5   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-7dhnt   1/1     Running       0          18m   172.17.0.6   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-p7sfc   1/1     Terminating   0          18m   172.17.0.7   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-pkwxl   1/1     Terminating   0          18m   172.17.0.8   minikube   <none>           <none>
```

You can run again the minikube_IP:node_port commands and type in your browser to see how the Pod id's have changed again (in fact, the 2 POD's running are part of the previous 4 replicas we created)

Lab 6: Updating your App

In the previous section, we learned how to scale in and scale out your application. We increased and decrease the number of replicas for our App. However, during the lifecycle of the App, we know that there will be versions of the same App. There could be updates or major releases.

Kubernetes provides a way to update our App. We will learn how to do it in this lab.



In this lab, we will create another version of our app, that potentially can change the way its deployed.

1. Let's find out what is our current App version.

   ```
   kubectl describe pods
   ```

   What is the app version? (HINT: we added a LABEL in Lab 4, so look for the labels)

   ```
   kubectl describe pods -l app=v1
   ```

2. As can be seen, we are using version 1. Let's create another version:

   ```
   kubectl set image deployments/kubernetes-bootcamp kubernetes-
   bootcamp=jocatalin/kubernetes-bootcamp:v2
   kubectl get pods -o wide
   kubectl get deployment -o wide
   ```

   [dCloud] if you get the status erro *"ImagePullBackOff"* you can try this command to download the image locally and the run again the kubectl run command:
   `docker pull gcr.io/google-samples/kubernetes-bootcamp:v1`

   You will see a similar output:

```
PS C:\CLUS19\LTRPRG-1200> kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=jocatalin/kubernetes-bootcamp:v2
deployment.extensions/kubernetes-bootcamp image updated
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -o wide
NAME                                   READY   STATUS        RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-64bfc9b489-dzvxw   1/1     Running       0          9s    172.17.0.8   minikube   <none>           <none>
kubernetes-bootcamp-64bfc9b489-j55sl   1/1     Running       0          13s   172.17.0.7   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-2lm54    1/1     Terminating   0          92m   172.17.0.5   minikube   <none>           <none>
kubernetes-bootcamp-b94cb9bff-7dhnt    1/1     Terminating   0          22m   172.17.0.6   minikube   <none>           <none>
PS C:\CLUS19\LTRPRG-1200> kubectl get deployment -o wide
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS            IMAGES                                SELECTOR
kubernetes-bootcamp   2/2     2            2           93m   kubernetes-bootcamp   jocatalin/kubernetes-bootcamp:v2   run=kubernetes-bootcamp
```

3. Let's check the current service detail:

   ```
   kubectl describe services/kubernetes-bootcamp
   kubectl get deployment -o wide
   ```

```
PS C:\CLUS19\LTRPRG-1200> kubectl describe services/kubernetes-bootcamp
Name:                     kubernetes-bootcamp
Namespace:                default
Labels:                   run=kubernetes-bootcamp
Annotations:              <none>
Selector:                 run=kubernetes-bootcamp
Type:                     NodePort
IP:                       10.96.104.179
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  32210/TCP
Endpoints:                172.17.0.7:8080,172.17.0.8:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
PS C:\CLUS19\LTRPRG-1200> kubectl get deployment -o wide
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS            IMAGES                                SELECTOR
kubernetes-bootcamp   2/2     2            2           96m   kubernetes-bootcamp   jocatalin/kubernetes-bootcamp:v2   run=kubernetes-bootcamp
```

4. It's running the 2 replicas. We expect it is running version 2 of the deployment. Before proceed, please check that your $NODE_PORT variable is updated. Otherwise please run the following commands:

```
Windows Powershell command:
$NODE_PORT = (kubectl get services/kubernetes-bootcamp -o
jsonpath="{.spec.ports[0].nodePort}")
$NODE_PORT

Linux command:
export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o
jsonpath="{.spec.ports[0].nodePort}")
echo NODE_PORT=$NODE_PORT
```

Let's do a curl to the specified IP:Port. Alternatively, you can put the URL on your browser (HINT: do a "minikube IP" if you don't remember the cluster IP). Do it several times to verify that version 2 is actually running

```
Windows Powershell command:
minikube ip
$NODE_PORT
curl [minikup_ipaddr]:[port]

Linux command:
curl $(minikube ip):$NODE_PORT
```

```
PS C:\CLUS19\LTRPRG-1200> $NODE_PORT
32210
PS C:\CLUS19\LTRPRG-1200> minikube ip
192.168.99.100
PS C:\CLUS19\LTRPRG-1200> curl 192.168.99.100:32210


StatusCode        : 200
StatusDescription : OK
Content           : Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-64bfc9b489-j55sl | v=2
```

When we rollout a new version, we need to have a way to control in case of a failure and rollback to the previous stable version. Kubernetes also provide that mechanism. Let's play a bit:

5. We can confirm the rollout of the new version using:

```
kubectl rollout status deployment/kubernetes-bootcamp
```

```
HEMORALE-M-80A4:~ hectormorales$ kubectl rollout status deployments/kubernetes-bootcamp
deployment "kubernetes-bootcamp" successfully rolled out
```

And now let's check the image id on the pods:

```
kubectl get pods -o wide
```

```
kubectl get deployment –o wide
```

```
PS C:\CLUS19\LTRPRG-1200> kubectl rollout status deployment/kubernetes-bootcamp
deployment "kubernetes-bootcamp" successfully rolled out
PS C:\CLUS19\LTRPRG-1200> kubectl get pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-64bfc9b489-dzvxw  1/1     Running   0          10m   172.17.0.8   minikube   <none>           <none>
kubernetes-bootcamp-64bfc9b489-j55sl  1/1     Running   0          10m   172.17.0.7   minikube   <none>           <none>


PS C:\CLUS19\LTRPRG-1200> kubectl get deployment -o wide
NAME                READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS            IMAGES                              SELECTOR
kubernetes-bootcamp 2/2     2            2           103m  kubernetes-bootcamp   jocatalin/kubernetes-bootcamp:v2    run=kubernetes-bootcamp
```

6. Now, let's learn how Kubernetes deals with rollbacks. Let's do another update and deploy another image, say v10. Run these commands:

```
kubectl set image deployments/kubernetes-bootcamp kubernetes-
bootcamp=gcr.io/google-samples/kubernetes-bootcamp:v10
kubectl get deployments
kubectl get pods
```

You will see a similar output:

```
PS C:\CLUS19\LTRPRG-1200> kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=gcr.io/google-samples/kubernetes-bootcamp:v10
deployment.extensions/kubernetes-bootcamp image updated
PS C:\CLUS19\LTRPRG-1200> kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp 2/2     1            2           106m
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
NAME                                  READY   STATUS             RESTARTS   AGE
kubernetes-bootcamp-64bfc9b489-dzvxw  1/1     Running            0          13m
kubernetes-bootcamp-64bfc9b489-j55sl  1/1     Running            0          13m
kubernetes-bootcamp-66ffd96587-dwf4s  0/1     ImagePullBackOff   0          13s
```

What happened? Let's get more detail. Run:

```
kubectl describe pods
```

Take a good look at the Events section in every container. We observe that the v10 image simply does not exist, then Kubernetes doesn't load it and is backing this off.

```
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  87s                default-scheduler  Successfully assigned default/kubernetes-bootcamp-66ffd96587-dwf4s to minikube
  Normal   Pulling    41s (x3 over 86s)  kubelet, minikube  Pulling image "gcr.io/google-samples/kubernetes-bootcamp:v10"
  Warning  Failed     40s (x3 over 85s)  kubelet, minikube  Failed to pull image "gcr.io/google-samples/kubernetes-bootcamp:v10": rpc error: code = Unknown
 desc = Error response from daemon: manifest for gcr.io/google-samples/kubernetes-bootcamp:v10 not found
  Warning  Failed     40s (x3 over 85s)  kubelet, minikube  Error: ErrImagePull
  Normal   BackOff    13s (x5 over 85s)  kubelet, minikube  Back-off pulling image "gcr.io/google-samples/kubernetes-bootcamp:v10"
  Warning  Failed     13s (x5 over 85s)  kubelet, minikube  Error: ImagePullBackOff
```

7. Let's rollback to the previous version:

```
kubectl rollout undo deployments/kubernetes-bootcamp
kubectl get pods
```

You can see the events again if you run the kubectl describe pods command.

```
PS C:\CLUS19\LTRPRG-1200> kubectl rollout undo deployments/kubernetes-bootcamp
deployment.extensions/kubernetes-bootcamp rolled back
PS C:\CLUS19\LTRPRG-1200> kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-64bfc9b489-dzvxw   1/1     Running   0          17m
kubernetes-bootcamp-64bfc9b489-j55sl   1/1     Running   0          17m
```

Lab 7: Do it on your own. Deploy a Web Server app in Kubernetes

Now it's time for you to show what you learned. The lab will consist on creating a small *Hello World* web server running in node.js in a container managed from Kubernetes. You can replicate this lab at home either on MacOSX or Linux and you can try with other Docker containers you have created. You can do this in Windows 10 as well with a few caveats. Ask your instructor about them.

You will be challenged to create this server using what you learned in the lab.

1. Please logon into your dCloud Pod
2. Make sure your minikube is running. Otherwise, start minikube in no vm-driver mode

   ```
   root@ubuntu:~# minikube start --vm-driver=none
   ```

3. In your working directory, create a new directory called **hellonode** and change to that directory. Make sure you are in the **hellonode** directory before proceed.
4. **Create a node.js File.** Create a file named server.js with the following content (a copy of this file has already cloned. Just move it to your new directory):

   ```
   var http = require('http');

   var handleRequest = function(request, response) {
     console.log('Received request for URL: ' + request.url);
     response.writeHead(200);
     response.end('Hello World!');
   };
   var www = http.createServer(handleRequest);
   www.listen(8080);
   ```

5. **[OPTIONAL step]** – you can test your server locally but you will need to download node.js. In dCloud we don't have node.js

   Open another command window and type node server.js
   Open your browser and type localhost:8080

6. **Create a Docker File**. Create the following file in your current directory (a copy of this file has already cloned. Just move it to your new directory):

```
FROM node:6.9.2
EXPOSE 8080
COPY server.js .
CMD node server.js
```

7. **[Optional step] – running Docker from Minikube**

   [dCLoud] given we are running Docker natively, you don't need this step. GO TO STEP 8

   [Minikube locally] As we recall, Minikube brings with a built-in Docker engine, which makes very easy to use it to build your own images. Let's build our container image using this feature:

   ```
   eval $(minikube docker-env)
   ```

8. **Create a Docker Image.** Create a docker image using the command syntax as follows. **Don't forget the dot at the end of the command!!**:

   ```
   docker build -t [Image_Name:version] .
   ```

   Image name = hello-node
   Version = v1

9. Create an image using the command kubectl run

   Name = hello-node
   Image Name = [Docker image created in step 5]
   Port = Please take a look to the Port number you need to use (hint: see the port number in the Dockerfile)

   ```
   kubectl run [Name] --image=[Image Name] --port=[Port]
   ```

10. Expose the deployment

    Name = hello-node
    Type = LoadBalancer

    ```
    kubectl expose deployment [Name] --type=[Type]
    ```

11. Run the service using the following command:

    ```
    minikube service hello-node
    ```

    [dCloud] as you can't run a browser, please do the following command:
    `curl $(minikube ip):<Port>`
    You can check the port using `kubectl get services`

12. Check the pod and the logs from that hello-node pod
13. Change the content to your server. Edit the `response.end` line as follows

```
response.end('Hello World! I am Running Docker & Kubernetes');
```

14. Change the content to your server. Edit the `response.end` line as follows
15. Build docker image version 2 using the command syntax and tag as follows:

```
docker build -t [Image_Name:version] .
Image name = hello-node
Version = v2
```

16. Set a new image to your existing deployment using the command:

```
kubectl set image
```

Make sure you tag your new image a v2

17. Run again the command:

```
minikube service hello-node
```

[dCloud] as you can't run a browser, please do the following command:
`curl $(minikube ip):<Port>`
You can check the port using `kubectl get services`

In order to see the changes.

18. **CONGRATULATIONS! YOU HAVE FINISHED!**

Extra Points – Lab 8

Enabling addons on Minikube

1. Let's see additional services on Minikube.

```
minikube addons list
```

2. Let's enable heapster

```
minikube addons enable heapster
```

3. Let's look for the pods that were created

```
kubectl get po,svc -n kube-system
```

4. Open heapster

```
minikube addons open heapster
```

Cleaning up

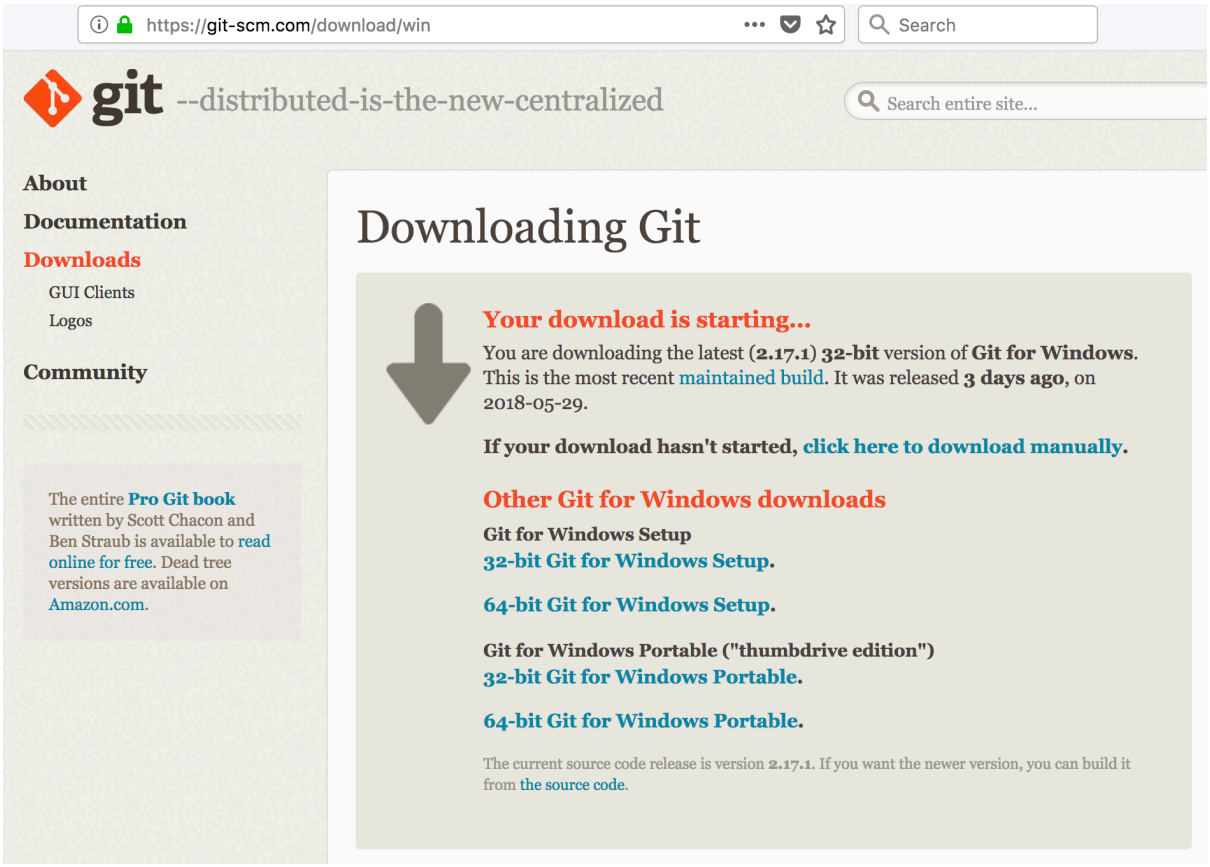
1. Enter these commands to clean up your lab

```
kubectl delete service hello-node
kubectl delete deployment hello-node
docker rmi hello-node:v1 hello-node:v2 -f
minikube stop
eval $(minikube docker-env -u)
```

Annex 1 – Installing Git for Windows

If you want to install Git for Windows, go this URL:
https://git-scm.com/download/win

It will automatically start the downloading and you will see an screen like this one:

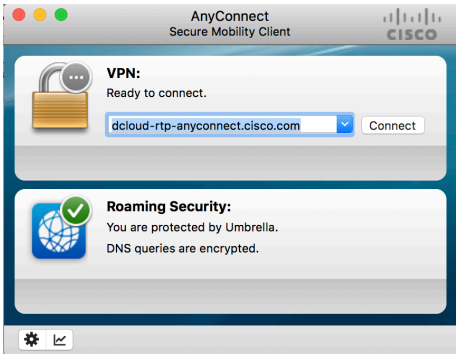


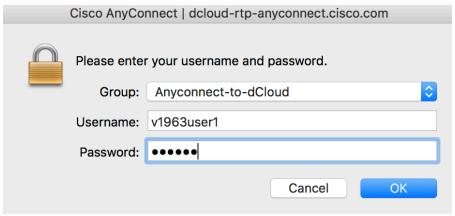
It is highly recommended to install using the suggested defaults as it modifies certain registries.

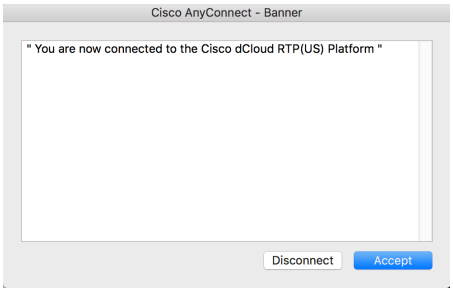
Annex 2 – Accessing into dCloud

1. Login into dcloud session using anyconnect from the Windows laptop. Use the assigned username/password combination.

   dcloud-rtp-anyconnect.cisco.com

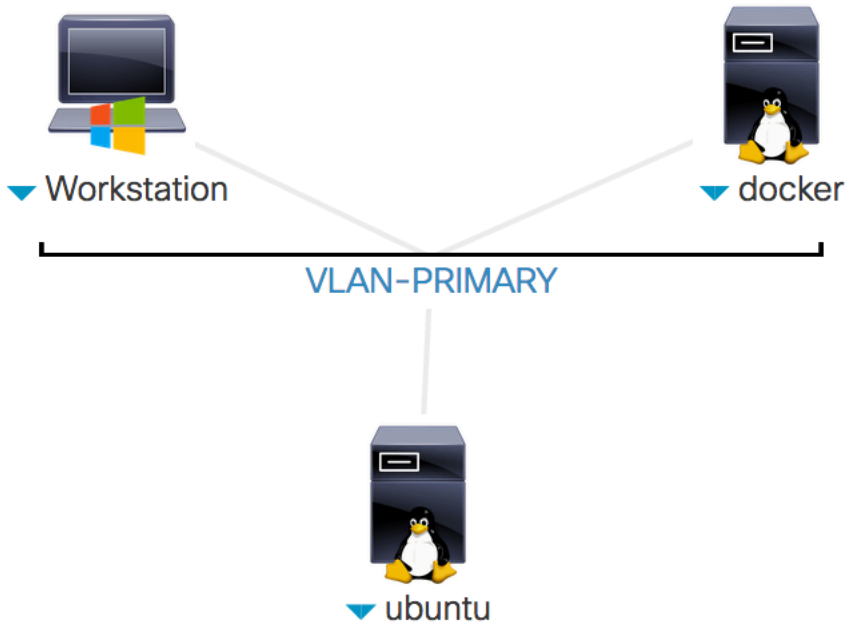






2. You have three servers in this lab:

- Workstation, your jump server. You are logging here.
- Ubuntu, your actual lab server where minikube, kubectl and docker are installed.
- Docker, a DNS server (we won't use this one for the purposes of the lab)



3. Open a Powershell window and ssh to the Ubuntu machine:

```
ssh root@198.18.134.28
password: C1sco12345
```

```
HEMORALE-M-892Q:~ hectormorales$ ssh root@198.18.134.28
root@198.18.134.28's password:
```

```
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-128-generic x86_64)

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

188 packages can be updated.
105 updates are security updates.

New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Mon Jun 10 23:53:14 2019 from 10.16.122.161
root@ubuntu:~#
```

4. In the Ubuntu machine you have:
    - Minikube
    - Kubectl
    - Docker

```
root@ubuntu:~# minikube version
minikube version: v0.27.0
root@ubuntu:~# kubectl version
Client Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.2", GitCommit:"66049e3b21efe110454d67df4fa62b08ea79a19b", GitTreeState:"clean",
BuildDate:"2019-05-16T16:23:09Z", GoVersion:"go1.12.5", Compiler:"gc", Platform:"linux/amd64"}
The connection to the server 198.18.134.28:8443 was refused - did you specify the right host or port?
root@ubuntu:~# docker version
Client:
 Version:      18.03.1-ce
 API version:  1.37
 Go version:   go1.9.5
 Git commit:   9ee9f40
 Built:        Thu Apr 26 07:17:20 2018
 OS/Arch:      linux/amd64
 Experimental: false
 Orchestrator: swarm
```

5. Given that this is a virtualized environment, we will use Minikube in no vm-driver mode, i.e. docker engine is installed on the machine.

```
minikube start --vm-driver=none  #when in dCloud
```

NOTE: If you have errors on minikube like certificates not valid or minikube does no start, you will need to reinstall minikube and kubectl:

UNINSTALL minikube and kubectl

```
sudo minikube stop; sudo minikube delete &&
sudo rm -rf ~/.kube ~/.minikube &&
sudo rm -rf /usr/local/bin/localkube /usr/local/bin/minikube &&
sudo systemctl stop '*kubelet*.mount' &&
sudo systemctl stop localkube.service &&
sudo systemctl disable localkube.service &&
sudo rm -rf /etc/kubernetes/
sudo rm /usr/local/bin/kubectl
kubeadm reset
sudo apt-get purge kubeadm kubectl kubelet kubernetes-cni kube*
sudo apt-get autoremove
sudo rm -rf ~/.kube
reboot   #CRITICAL TO REBOOT!!
```

INSTALL minikube and kubectl

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/v1.1.1/minikube-linux-
amd64 && chmod +x minikube && sudo cp minikube /usr/local/bin/ && rm
minikube

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl
-s https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

minikube start --vm-driver=none  #when in dCloud
```

You should see a similar screen:

```
root@ubuntu:~# minikube start --vm-driver=none
😄  minikube v1.1.1 on linux (amd64)
🤹  Creating none VM (CPUs=2, Memory=2048MB, Disk=20000MB) ...
🤹  Configuring environment for Kubernetes v1.14.3 on Docker 18.03.1-ce
⚠  Unable to load cached images: loading cached images: loading image /root/.minikube/cache/images/gcr.io/k8s-minikube/storage-provisioner_v1.8.1: stat /ro
ot/.minikube/cache/images/gcr.io/k8s-minikube/storage-provisioner_v1.8.1: no such file or directory
💾  Downloading kubeadm v1.14.3
💾  Downloading kubelet v1.14.3
🚜  Pulling images ...
🚀  Launching Kubernetes ...
🤹  Configuring local host environment ...

⚠  The 'none' driver provides limited isolation and may reduce system security and reliability.
⚠  For more information, see:
⚠  https://github.com/kubernetes/minikube/blob/master/docs/vmdriver-none.md

⚠  kubectl and minikube configuration will be stored in /root
⚠  To use kubectl or minikube commands as your own user, you may
⚠  need to relocate them. For example, to overwrite your own settings:

    ▪ sudo mv /root/.kube /root/.minikube $HOME
    ▪ sudo chown -R $USER $HOME/.kube $HOME/.minikube

⚠  This can also be done automatically by setting the env var CHANGE_MINIKUBE_NONE_USER=true
🤹  Verifying: apiserver proxy etcd scheduler controller dns
🏄  Done! kubectl is now configured to use "minikube"
```

6. For LAB 7 you will need to use dCloud!

You can go to lab 2 now