**Solutions**

All the following work has been done on a MacBook Pro.

### **Task 0: Install a ubuntu 16.04 server 64-bit**

Installation of ubuntu 16.04 server 64-bit on virtualBox

Step 1: Choose OS x hosts and download the virtualbox for Mac (<https://www.virtualbox.org/wiki/Downloads>).

A screenshot of a cell phone

Description automatically generated

Step 2: download ubuntu 16.04 server 64-bit install image (<https://releases.ubuntu.com/16.04.7/?_ga=2.177823845.1147325132.1598571763-486386747.1598297836>)

A screenshot of a cell phone

Description automatically generated

Step 3: install VirtualBox on host machine.

Step 4: install the guest machine on the VirtualBox.

* Open the installed VirtualBox from host machine, choose machine->new…
* Fill the name (ubuntu16.04), choose type and version(Ubuntu 64-bit)
* Choose memory size (2GB) and file size (10GB). Make sure VDI is chosen for the hard disk file type.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

* Then choose the default settings in the following steps

A screenshot of a social media post

Description automatically generated

* Right click the “ubuntu16.04” and choose ->start->normal start, choose the downloaded .iso file and click ‘start’. Then a window will be popup and now you can install the Ubuntu system in the VirtualBox.

A screenshot of a cell phone

Description automatically generated

* Then click the green “start” button, a new window will be popup and the new Ubuntu system will be bootup. Type the username and password will deliver you to the Ubuntu.

A screenshot of a cell phone

Description automatically generated

for VM, use NAT network and forward required ports to host machine

* 22->2222 for ssh
* 80->8080 for gitlab
* 8081/8082->8081/8082 for go app
* 31080/31081->31080/31081 for go app in k8s

The port forwarding rule is displayed below.

A screenshot of a cell phone

Description automatically generated

The first time when I run the following command, it fails:

ssh luobozhi@127.0.0.1 -p 2222

ssh\_exchange\_identification: Connection closed by remote host

This is because the SSH service is not installed on the guest machine. Execute the following command can start the SSH service of the guest machine:

Sudo apt-get install openssh-server

Then start the service:

/etc/init.d/ssh start

After that, you can check if you can login the guest machine from host machine through SSH. Open a new terminal in the host machine and type the following command:

|  |
| --- |
| (base) lobos-mbp:~ zhiboluo$ ssh luobozhi@localhost -p 2222  The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.  ECDSA key fingerprint is SHA256:ye4I/ENIopDrGgc4gr+2CCOS/NdoRWU0/Rknvpd8PGQ.  Are you sure you want to continue connecting (yes/no)? |

Type “yes” and try the above command again will get the following:

|  |
| --- |
| (base) lobos-mbp:~ zhiboluo$ ssh luobozhi@localhost -p 2222  luobozhi@localhost's password:  Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-187-generic x86\_64)  \* Documentation: https://help.ubuntu.com  \* Management: https://landscape.canonical.com  \* Support: https://ubuntu.com/advantage  \* Are you ready for Kubernetes 1.19? It's nearly here! Try RC3 with  sudo snap install microk8s --channel=1.19/candidate --classic  https://microk8s.io/ has docs and details.  0 packages can be updated.  0 updates are security updates.  New release '18.04.5 LTS' available.  Run 'do-release-upgrade' to upgrade to it.  Last login: Thu Aug 27 16:28:31 2020 |

At this moment, we can access to the guest machine from host machine through ssh.

### **Task 1: Update system**

ssh to guest machine from host machine ($ ssh user@localhost -p 2222) and update the system to the latest

<https://help.ubuntu.com/16.04/serverguide/apt.html>

upgrade the kernel to the 16.04 latest

Step 1: First setup the ssh connection between the host machine and guest machine. Then open a terminal in the host machine and type $ ssh luobozhi@localhost -p 2222 to login the guest machine.

A screenshot of a social media post

Description automatically generated

Step 2: check the installed kernel version

|  |
| --- |
| **luobozhi@luobo**:**~**$ uname -sr  Linux 4.4.0-187-generic |

Step 3: upgrade to the newest kernel.

In order to upgrade to the newest kernel version, open <http://kernel.ubuntu.com/~kernel-ppa/mainline/> and choose the newest kernel version (at this moment, v5.8.5 is the latest version). For 64-bit system, download the following .deb files through wget.

|  |
| --- |
| **luobozhi@luobo**:**~**$ wget <https://kernel.ubuntu.com/~kernel-ppa/mainline/v5.8.5/amd64/linux-headers-5.8.5-050805_5.8.5-050805.202008270831_all.deb>  **luobozhi@luobo**:**~**$ wget <https://kernel.ubuntu.com/~kernel-ppa/mainline/v5.8.5/amd64/linux-headers-5.8.5-050805-generic_5.8.5-050805.202008270831_amd64.deb>  **luobozhi@luobo**:**~**$ wget <https://kernel.ubuntu.com/~kernel-ppa/mainline/v5.8.5/amd64/linux-image-unsigned-5.8.5-050805-generic_5.8.5-050805.202008270831_amd64.deb>  **luobozhi@luobo**:**~**$ wget <https://kernel.ubuntu.com/~kernel-ppa/mainline/v5.8.5/amd64/linux-modules-5.8.5-050805-generic_5.8.5-050805.202008270831_amd64.deb> |

And the following .deb files are downloaded

|  |
| --- |
| **luobozhi@luobo**:**~**$ ls  **linux-headers-5.8.5-050805\_5.8.5-050805.202008270831\_all.deb** **linux-image-unsigned-5.8.5-050805-generic\_5.8.5-050805.202008270831\_amd64.deb**  **linux-headers-5.8.5-050805-generic\_5.8.5-050805.202008270831\_amd64.deb**  **linux-modules-5.8.5-050805-generic\_5.8.5-050805.202008270831\_amd64.deb** |

Then, install through the following command.

$ sudo dpkg -i \*.deb

But found the following error:

|  |
| --- |
| The following packages have unmet dependencies:  linux-headers-4.18.12-041812-generic : Depends: libssl1.1 (>= 1.1.0) but it is not installable ….  Package libssl1.1 is not installed. |

Then manually download a libssl1.1.deb file and install libssl1.1

|  |
| --- |
| wget http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl1.1\_1.1.0g-2ubuntu4\_amd64.deb  sudo dpkg -i libssl1.1\_1.1.0g-2ubuntu4\_amd64.deb |

Then run this command again: $ sudo dpkg -i \*.deb

Finally, reboot the guest machine and check the kernel version. And it can be seen the kernel version is update to v5.8.5.

|  |
| --- |
| **luobozhi@luobo**:**~**$ uname -sr  Linux 5.8.5-050805-generic |

### **Task 2:** **Install** **gitlab-ce version in the host**

<https://about.gitlab.com/install/#ubuntu?version=ce>

Expect output: Gitlab is up and running at [http://127.0.0.1](http://127.0.0.1/) (no tls or FQDN required)

Access it from host machine [http://127.0.0.1:8080](http://127.0.0.1:8080/)

Step 1: install and configure the necessary dependencies

|  |
| --- |
| sudo apt-get install -y curl openssh-server ca-certificates tzdata |

Step 2: add the Gitlab package repository and install the package

|  |
| --- |
| **luobozhi@luobo**:**~**$ curl -sS https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash  **luobozhi@luobo**:**~**$ sudo apt-get install gitlab-ce  **luobozhi@luobo**:**~**$ sudo gitlab-ctl reconfigure  **luobozhi@luobo**:**~**$ sudo gitlab-ctl status // check gitlab status  run: alertmanager: (pid 11523) 64s; run: log: (pid 11205) 159s  run: gitaly: (pid 11485) 67s; run: log: (pid 10724) 284s  run: gitlab-exporter: (pid 11495) 66s; run: log: (pid 11148) 177s  run: gitlab-workhorse: (pid 11463) 67s; run: log: (pid 11053) 201s  run: grafana: (pid 11536) 63s; run: log: (pid 11426) 85s  run: logrotate: (pid 11083) 190s; run: log: (pid 11095) 187s  run: nginx: (pid 11067) 196s; run: log: (pid 11074) 195s  run: node-exporter: (pid 11470) 67s; run: log: (pid 11126) 183s  run: postgres-exporter: (pid 11530) 63s; run: log: (pid 11226) 153s  run: postgresql: (pid 10849) 272s; run: log: (pid 10861) 269s  run: prometheus: (pid 11502) 66s; run: log: (pid 11186) 165s  run: puma: (pid 10998) 215s; run: log: (pid 11009) 212s  run: redis: (pid 10677) 294s; run: log: (pid 10685) 291s  run: redis-exporter: (pid 11497) 66s; run: log: (pid 11165) 171s  run: sidekiq: (pid 11017) 208s; run: log: (pid 11027) 207s |

Step 3: browse to the hostname and login

Step 4: Type [http://127.0.0.1:8080](http://127.0.0.1:8080/) in the browser will redirect to the Gitlab page, setup the new password and finally we can enter the following welcome page.

A screenshot of a cell phone

Description automatically generated

### **Task 3: Create a demo group/project in gitlab**

named demo/go-web-hello-world (demo is group name, go-web-hello-world is project name).

Use golang to build a hello world web app (listen to 8081 port) and check-in the code to mainline.

<https://golang.org/>

<https://gowebexamples.com/hello-world/>

Expect source code at <http://127.0.0.1:8080/demo/go-web-hello-world>

Step 1: create a new group named with “demo” in the above GitLab page.

Step 2: in the newly created group, create a new project named with “go-web-hello-world”. The project can be accessed through <http://127.0.0.1:8080/demo/go-web-hello-world>. As shown below. Now the project is empty as displayed below. Next step is to use Golang in the host machine (my MBP) and build a helloweb.go app (listen to 8081).

A screenshot of a cell phone

Description automatically generated

Step 3: Download and install Golang (<https://golang.org/dl/>) in the host machine. Open a terminal and type “go” to validate the correctness of the installation.

|  |
| --- |
| zhiboluo$ go  Go is a tool for managing Go source code.  Usage:  go <command> [arguments]  The commands are:  bug start a bug report  …… |

Step 4: Create a new folder and create a .go file in that folder.

|  |
| --- |
| zhiboluo$ cd Documents/  Documents zhiboluo$ mkdir GOProjects  Documents zhiboluo$ cd GOProjects  GOProjects zhiboluo$ vim helloweb.go |

Step 5: type the following code in the helloweb.go

|  |
| --- |
| package main  import (  "fmt"  "net/http"  "strings"  "log"  )  func sayhelloName(w http.ResponseWriter, r \*http.Request) {  r.ParseForm()  fmt.Println(r.Form)  fmt.Println("path", r.URL.Path)  fmt.Println("scheme", r.URL.Scheme)  fmt.Println(r.Form["url\_long"])  for k, v := range r.Form {  fmt.Println("key:", k)  fmt.Println("val:", strings.Join(v, ""))  }  fmt.Fprintf(w, "Go Web Hello World!\n") // message output to brower  }  func main() {  http.HandleFunc("/", sayhelloName)  err := http.ListenAndServe(":8081", nil) // listen port  if err != nil {  log.Fatal("ListenAndServe: ", err)  }  } |

Step 6: build the go file and start the exe file.

|  |
| --- |
| GOProjects zhiboluo$ go build helloweb.go  GOProjects zhiboluo$ ./helloweb |

Step 7: open the browser and go to <http://127.0.0.1:8081/>, appears the following message.

A screenshot of a social media post

Description automatically generated

Step 8: At this moment, we have built a hello world web app (listen to 8081 port) using Golang. Next step is to push the code to the newly created project in the GitLab.

Step 9: cd to the folder where we create the go project. Type the following commands to create a local git repository.

|  |
| --- |
| GOProjects zhiboluo$ git init  GOProjects zhiboluo$ git remote add origin [http://127.0.0.1:8080/demo/go-web-hello-world.git](http://127.0.0.1:8080/demo/go-web-hello-world.git%20//)  //followed by typing username(root) and password  GOProjects zhiboluo$ git add .  GOProjects zhiboluo$ git commit -m "Initial commit"  GOProjects zhiboluo$ git push -u origin master  Username for 'http://127.0.0.1:8080': root  Password for 'http://root@127.0.0.1:8080':  Counting objects: 5, done.  Delta compression using up to 4 threads.  Compressing objects: 100% (5/5), done.  Writing objects: 100% (5/5), 3.20 MiB | 5.64 MiB/s, done.  Total 5 (delta 0), reused 0 (delta 0)  To http://127.0.0.1:8080/demo/go-web-hello-world.git  \* [new branch] master -> master  Branch 'master' set up to track remote branch 'master' from 'origin'. |

Step 10: refresh the GitLab page and notice the initial commit has been pushed to this project.

A screenshot of a social media post

Description automatically generated

### **Task 4: Build the app and expose ($ go run) the service to 8081 port**

Expect output:

curl http://127.0.0.1:8081

Go Web Hello World!

Step 1: open a new terminal in the host machine and start the helloweb app by typing “./helloweb”. Then type “curl http://127.0.0.1:8081” will output the following message.

|  |
| --- |
| zhiboluo$ curl http://127.0.0.1:8081  Go Web Hello World! |

### **Task 5: Install docker**

According to the installation instructions (<https://docs.docker.com/install/linux/docker-ce/ubuntu/>), there are 3 different ways to install docker in Ubuntu. Here I choose the first method which will install docker using the repository.

Step 1: Before install the Docker engine, it is suggested to set up the repository. Update the apt package index and install packages to allow apt to use a repository over HTTPS.

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo apt-get update  **luobozhi@luobo**:**~**$ sudo apt-get install \  > apt-transport-https \  > ca-certificates \  > curl \  > gnupg-agent \  > software-properties-common |

Step 2: Add Docker’s official GPG key, and Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

|  |
| --- |
| **luobozhi@luobo**:**~**$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  OK  **luobozhi@luobo**:**~**$ sudo apt-key fingerprint 0EBFCD88  pub 4096R/0EBFCD88 2017-02-22  Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88  uid Docker Release (CE deb) <docker@docker.com>  sub 4096R/F273FCD8 2017-02-22 |

Step 3: Use the following command to set up the stable repository.

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo add-apt-repository \  > "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  > $(lsb\_release -cs) \  > stable" |

Step 4: This step is going to install Docker Engine. First update the apt package index, and install the latest version of Docker Engine and containerd, or go to the next step to install a specific version:

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo apt-get update  **luobozhi@luobo**:**~**$ sudo apt-get install docker-ce docker-ce-cli containerd.io |

Step 5: finally, verify that Docker Engine is installed correctly by running the hello-world image. This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo docker run hello-world |

Docker Engine is installed and running. The docker group is created but no users are added to it. You need to use sudo to run Docker commands.

### **Task 6: Run the app in container**

build a docker image ($ docker build) for the web app and run that in a container ($ docker run), expose the service to 8082 (-p)

<https://docs.docker.com/engine/reference/commandline/build/>

Check in the Dockerfile into gitlab

Expect output:

curl http://127.0.0.1:8082

Go Web Hello World!

In Task 3 and 4, the hello world program was built and run on the host machine (I made a misunderstanding of this Task). Here I will build the go web app again and expose the service to 8081 port in the guest machine.

Step 1: create a folder for the Go web project.

|  |
| --- |
| **luobozhi@luobo**:**~**$ mkdir web-app  **luobozhi@luobo**:**~**$ cd web-app/ |

Step 2: install the Golang in the guest machine.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ sudo apt install golang-go |

Step 3: create a file named with “helloweb\_2.go” and type the following code.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ vim helloweb\_2.go  **luobozhi@luobo**:**~/web-app**$ cat helloweb\_2.go  package main  import (  "fmt"  "net/http"  "log"  )  func main() {  http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {  fmt.Fprintf(w, "GO WEB, Hello World!\n")  })  err := http.ListenAndServe(":8081", nil)  if err != nil {  log.Fatal("ListenAndServe: ", err)  }  } |

Step 4: Build the .go file and run it in the background.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ go build helloweb\_2.go  **luobozhi@luobo**:**~/web-app**$ ./helloweb\_2 &  [1] 9694 |

Step 5: check the output.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ curl http://127.0.0.1:8081  GO WEB, Hello World! |

Next, I will build a docker image ($ docker build) for the web app and run that in a container ($ docker run), expose the service to 8082 (-p)

Step 6: create a Dockerfile to tell the Docker how to containerize this web app.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ vim Dockerfile  **luobozhi@luobo**:**~/web-app**$ cat Dockerfile  FROM golang: latest  RUN mkdir -p /go/src/web-app  WORKDIR /go/src/web-app  COPY . /go/src/web-app  RUN go-wrapper download  RUN go-wrapper install  ENV PORT 8082  EXPOSE 8082  CMD ["go-wrapper", "run"] |

Step 7: Build the Go web app in the Docker and get some errors,

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ sudo docker build --rm -t web-app . |

The following error was found.

|  |
| --- |
| ….  ---> 517a74d637a8  Step 4/9 : COPY . /go/src/web-app  ---> fbac6d82795b  Step 5/9 : RUN go-wrapper download  ---> Running in b9369dccddf0  /bin/sh: 1: go-wrapper: not found  … |

Step 8: After google search, change the image of golang from “latest” to “1.9.6-alpine3.7” and build it again.

Step 9: now we can docker run the web app. Execute the following command and found the port 8082 has been used.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ sudo docker run -p 8082:8082 --name="test" -d web-app  bfd5aa086a7d81012a55005191e8f1f4c082037e9744d5261d4f2c4ab2c81e5c  docker: Error response from daemon: driver failed programming external connectivity on endpoint test0 (a88c038c87d0c8712dffd795fffd30ddeaa4899c16af2ada25a7ce49fd83b7b9): Error starting userland proxy: listen tcp 0.0.0.0:8082: bind: address already in use. |

Step 10: Check which process using the 8082 port.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ sudo lsof -i:8082  COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME  prometheu 1268 gitlab-prometheus 30u IPv4 26615 0t0 TCP localhost:46832->localhost:8082 (ESTABLISHED)  bundle 1344 git 25u IPv4 26493 0t0 TCP localhost:8082 (LISTEN)  bundle 1344 git 78u IPv4 26616 0t0 TCP localhost:8082->localhost:46832 (ESTABLISHED) |

Step 11: Here I’m not sure if the bundle can be killed, so I check another port 8084 and found no one use it.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ sudo docker run -p 8084:8082 --name="test01" -d web-app  962e37ef6190dce5fbd44082b5302a2808cf841f36428492ffdccc85bfe7d6c9 |

Step 12: check the output.

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ curl <http://127.0.0.1:8084>  GO WEB, Hello World! |

Step 13: finally, using this command “git push origin master” to push the “Dockerfile” to GitLab.

|  |
| --- |
| git add Dockerfile  git commit -m "add Dockerfile"  git push origin master |

### **Task 7: Push image to dockerhub**

tag the docker image using your\_dockerhub\_id/go-web-hello-world:v0.1 and push it to docker hub (<https://hub.docker.com/>)

Expect output: <https://hub.docker.com/repository/docker/your_dockerhub_id/go-web-hello-world>

Step 1: create a Docker account to share the web-app image. And create a new repository named with “go-web-hello-world”

A screenshot of a cell phone

Description automatically generated

Step 2: login to the local machine

|  |
| --- |
| **luobozhi@luobo**:**/**$ sudo docker login  Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.  Username: luobozhi  Password:  WARNING! Your password will be stored unencrypted in /home/luobozhi/.docker/config.json.  Configure a credential helper to remove this warning. See  https://docs.docker.com/engine/reference/commandline/login/#credentials-store  Login Succeeded |

Step 3: tag the image using this command “docker tag image username/repository:tag”.

|  |
| --- |
| **luobozhi@luobo**:**/**$ sudo docker images  REPOSITORY TAG IMAGE ID CREATED SIZE  web-app latest f439a5478344 3 hours ago 283MB  **luobozhi@luobo**:**/**$ sudo docker tag f439a5478344 luobozhi/go-web-hello-world:v0.1  **luobozhi@luobo**:**/**$ sudo docker images  REPOSITORY TAG IMAGE ID CREATED SIZE  luobozhi/go-web-hello-world v0.1 f439a5478344 3 hours ago 283MB  web-app latest f439a5478344 3 hours ago 283MB |

Step 4: push the tagged image to Docker hub.

|  |
| --- |
| **luobozhi@luobo**:**/**$ sudo docker push luobozhi/go-web-hello-world:v0.1  The push refers to repository [docker.io/luobozhi/go-web-hello-world]  5c3d13c97904: Pushed  12c3b4d59a45: Pushed  40f2c7848dea: Pushed  1560f6488009: Pushed  dd84dbaa91f8: Mounted from library/golang  40451ef226a1: Mounted from library/golang  dc7275ed3768: Mounted from library/golang  6eeebfabfc25: Mounted from library/golang  79d579ee9ade: Mounted from library/golang  747a1ba085b0: Mounted from library/golang  cd7100a72410: Mounted from library/golang  v0.1: digest: sha256:08dd8cda2c5d1017b605db2fcfd63495428dbd187fddf026743fb917364f2eb3 size: 2614 |

Step 5: Finally, login to my Docker account and check the repository (<https://hub.docker.com/r/luobozhi/go-web-hello-world/>)

A screenshot of a social media post

Description automatically generated

### **Task 8: Document the procedure in a MarkDown file**

create a README.md file in the gitlab repo and add the technical procedure above (0-7) in this file

Step 1: login to the GitLab and directly add a README.md file. Finally commit the changes.

A screenshot of a social media post

Description automatically generated

### **Task 9: Install a single node Kubernetes cluster using kubeadm**

<https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/>

Check in the admin.conf file into the gitlab repo

Next steps will configure a single node Kubernetes cluster on the Ubuntu Server

Step 1: First use the following commands to install Kubernetes:

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo apt-get update && sudo apt-get install -y apt-transport-https && curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add –  **luobozhi@luobo**:**~**$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list && sudo apt-get update  **luobozhi@luobo**:**~**$ sudo apt install -y kubeadm kubelet kubernetes-cni |

Step 2: Next, turn off the swap on Ubuntu Server because it is required in the changelog:

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo swapoff -a  **luobozhi@luobo**:**~**$ sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab |

Step 3: Creating Single Node Kubernetes Cluster with Kubeadm. Here I will use kubeadm which helps bootstrap Kubernetes cluster. At the beginning, install kubectl using the following command:

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo apt-get update && sudo apt-get install -y apt-transport-https && curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add –  **luobozhi@luobo**:**~**$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list && sudo apt-get update  **luobozhi@luobo**:**~**$ sudo apt install -y kubeadm kubelet kubernetes-cni |

Step 4: Now create a single node cluster. Remember to change –apiserver-advertise-address=10.0.2.15 with your VM inet address.

|  |
| --- |
| **luobozhi@luobo**:**~**$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=10.0.2.15 --ignore-preflight-errors=NumCPU  … Your Kubernetes control-plane has initialized successfully! … |

Step 5: To start using your cluster, you need to run the following as a regular user:

|  |
| --- |
| **luobozhi@luobo**:**~**$ mkdir -p $HOME/.kube  **luobozhi@luobo**:**~**$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  **luobozhi@luobo**:**~**$ sudo chown $(id -u):$(id -g) $HOME/.kube/config |

Step 6: Next, we will copy the admin.conf file (under folder: /etc/kubernetes/admin.conf) change its permission (chmod 755 admin.conf) and push it to the GitLab.

A screenshot of a cell phone

Description automatically generated

In order for Kubernetes to work for a non-root user, you must run the following commands:

|  |
| --- |
| mkdir $HOME/.k8s  sudo cp /etc/kubernetes/admin.conf $HOME/.k8s/  sudo chown **$(**id -u**)**:**$(**id -g**)** $HOME/.k8s/admin.conf  export KUBECONFIG**=**$HOME/.k8s/admin.conf  echo "export KUBECONFIG=$HOME/.k8s/admin.conf" | tee -a ~/.bashrc |

Configuring Kubernetes Cluster Networking. After kubernetes installation, we need to configure networking. Execute two commands to set the networking:

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml  **luobozhi@luobo**:**~/web-app**$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifests/kube-flannel-rbac.yml |

Finally, we have to use the kubectl taint command, so that our master node can create pods and be used as a worker node:

|  |
| --- |
| **luobozhi@luobo**:**~/web-app**$ kubectl taint nodes --all node-role.kubernetes.io/master- |

### **Task 10: Deploy the hello world container**

in the kubernetes above and expose the service to nodePort 31080

Expect output:

curl http://127.0.0.1:31080

Go Web Hello World!

Check in the deployment yaml file or the command line into the gitlab repo

I was failed to deploy the hello world container in the Kubernetes. I attached the steps I followed from the reference.

Reference: <https://kubernetes.io/docs/tasks/access-application-cluster/service-access-application-cluster/>

Step 1: create a file named “hello-application.yaml”

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: hello-world  spec:  selector:  matchLabels:  run: load-balancer-example  replicas: 2  template:  metadata:  labels:  run: load-balancer-example  spec:  containers:  - name: hello-world  image: gcr.io/google-samples/node-hello:1.0 //here should change to your own images that are generated through Docker  ports:  - containerPort: 8080  protocol: TCP |

Step 2: Run a Hello World application in your cluster: Create the application Deployment using the file above:

|  |
| --- |
| kubectl apply -f hello-application.yaml |

Step 3: Display information about the Deployment:

|  |
| --- |
| kubectl get deployments hello-world  kubectl describe deployments hello-world |

Step 4: Display information about your ReplicaSet objects:

|  |
| --- |
| kubectl get replicasets  kubectl describe replicasets |

Step 5: Create a Service object that exposes the deployment:

|  |
| --- |
| kubectl expose deployment hello-world --type=NodePort --name=example-service |

Step 6: Display information about the Service:

|  |
| --- |
| kubectl describe services example-service |

The output is similar to this:

A picture containing bird

Description automatically generated

Make a note of the NodePort value for the service. For example, in the preceding output, the NodePort value is 31496.

Step 7: List the pods that are running the Hello World application:

|  |
| --- |
| kubectl get pods --selector="run=load-balancer-example" --output=wide |

The output is similar to this:

A screenshot of a cell phone

Description automatically generated

Step 8: Get the public IP address of one of your nodes that is running a Hello World pod. How you get this address depends on how you set up your cluster. For example, if you are using Minikube, you can see the node address by running kubectl cluster-info. If you are using Google Compute Engine instances, you can use the gcloud compute instances list command to see the public addresses of your nodes.

Step 9: On your chosen node, create a firewall rule that allows TCP traffic on your node port. For example, if your Service has a NodePort value of 31568, create a firewall rule that allows TCP traffic on port 31568. Different cloud providers offer different ways of configuring firewall rules.

Step 10: Use the node address and node port to access the Hello World application:

|  |
| --- |
| curl http://<public-node-ip>:<node-port> |

where <public-node-ip> is the public IP address of your node, and <node-port> is the NodePort value for your service. The response to a successful request is a hello message:

|  |
| --- |
| Hello Kubernetes! |

Step 11: Cleaning up. To delete the Service, enter this command:

|  |
| --- |
| kubectl delete services example-service |

To delete the Deployment, the ReplicaSet, and the Pods that are running the Hello World application, enter this command:

|  |
| --- |
| kubectl delete deployment hello-world |

### **Task 11: Install kubernetes** **dashboard**

and expose the service to nodeport 31081

<https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>

Expect output: [https://127.0.0.1:31081](https://127.0.0.1:31081/) (asking for token)

Step 1: Kubernetes Dashboard is web-based UI for Kbuernetes clusters. It allows to manage applications deployed in the cluster. Use the following command to deploy the dashboard on the cluster:

|  |
| --- |
| **luobozhi@luobo**:**~**$ kubectl apply -f <https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml> |

Step 2: To access Dashboard from your Ubuntu you must create a secure channel to your Kubernetes cluster. Run the following command:

|  |
| --- |
| **luobozhi@luobo**:**~**$ kubectl proxy  Starting to serve on 127.0.0.1:8001 |

Step 3: Now check in the new Putty window if dashboard is properly exposed: You should see 200 OK Http response:

|  |
| --- |
| **luobozhi@luobo**:**~**$ Starting to serve on 127.0.0.1:8001  curl -I 127.0.0.1:8001  HTTP/1.1 200 OK  Cache-Control: no-cache, private  Content-Type: application/json  Date: Sat, 29 Aug 2020 14:25:53 GMT |

Step 4: Proxy to host Windows

We will use the Windows browser to manage the cluster. To enable this run the following command in Ubuntu terminal (remember to change the inet IP addrress 10.0.2.15 with your VM’s inet IP address.):

|  |
| --- |
| **luobozhi@luobo**:**~**$ kubectl proxy --address 10.0.2.15 --accept-hosts '^.\*$' --port=8001  Starting to serve on 10.0.2.15:8001 |

Step 5: Go to http:// 10.0.2.15:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login in your browser on windows. You should have seen the dashboard login screen. But unfortunately, I failed to connect to the dashboard in my computer. I will check the installation again and figure out how to access Kubernetes dashboard through the host browser.

A screenshot of a cell phone

Description automatically generated

Typing the following command, it shows the dashboard is in the pending status.

|  |
| --- |
| **luobozhi@luobo**:**~**$ kubectl get pods --all-namespaces  NAMESPACE NAME READY STATUS RESTARTS AGE  kube-system coredns-f9fd979d6-249q8 0/1 Pending 0 57m  kube-system coredns-f9fd979d6-vg257 0/1 Pending 0 57m  kube-system etcd-luobo 1/1 Running 0 57m  kube-system kube-apiserver-luobo 1/1 Running 0 57m  kube-system kube-controller-manager-luobo 1/1 Running 0 57m  kube-system kube-flannel-ds-amd64-xgx2z 1/1 Running 0 54m  kube-system kube-proxy-mlmxh 1/1 Running 0 57m  kube-system kube-scheduler-luobo 1/1 Running 0 57m  kube-system kubernetes-dashboard-6ff6454fdc-s2gh7 0/1 Pending 0 52m  kubernetes-dashboard dashboard-metrics-scraper-7b9b99d599-2tk9n 0/1 Pending 0 7m59s  kubernetes-dashboard kubernetes-dashboard-6d4799d74-8h9qm 0/1 Pending 0 7m59s |

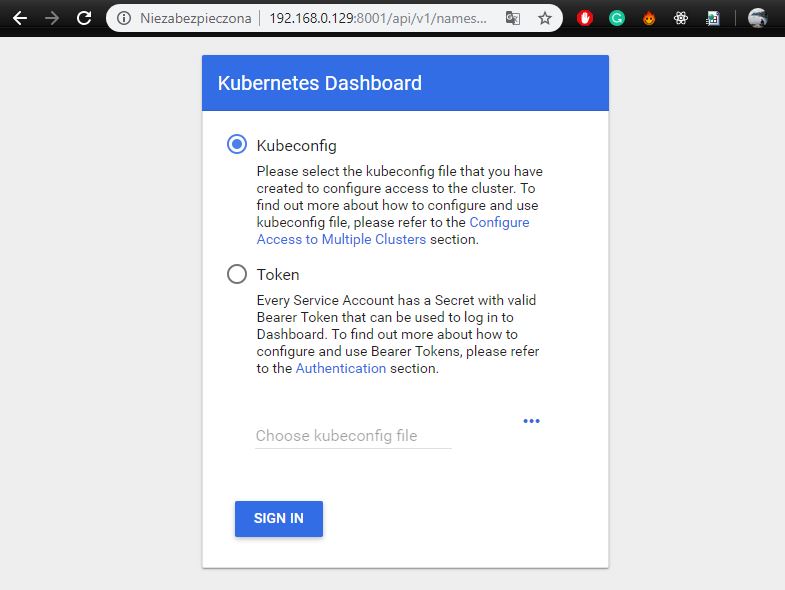
### **Task 12:** **Generate token for dashboard login in task 11**

figure out how to generate a token to login to the dashboard and publish the procedure to the gitlab.

As I failed to login to the Dashboard so I find some related material that can deploy Kubernetes Dashboard. Reference: <https://www.devdiaries.net/blog/Single-Node-Kubernetes-Cluster-Part-3/>

If everything goes well, when go to http://[your VM’s inet IP]:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login in your browser, it supposes to popup the following window.

|  |
| --- |
| kubectl proxy --address your\_VM’s\_inet\_IP --accept-hosts '^.\*$' --port**=**8001 |



Next is to creating a service account and a role binding in Kubernetes. To login to the dashboard, a service account and a role binding are needed. To create a service account, use the following command:

|  |
| --- |
| kubectl create serviceaccount --namespace kube-system admin-user |

And for role binding:

|  |
| --- |
| kubectl create clusterrolebinding admin-user --clusterrole**=**cluster-admin --serviceaccount**=**kube-system:admin-user |

Next, sign in to Kubernetes Dashboard. There are several ways to log into the dashboard. We can genererate JWT token:

|  |
| --- |
| kubectl -n kube-system describe secret **$(**kubectl -n kube-system get secret | grep admin-user | awk '{print $1}'**)** |

Copy generated token and use it when logging in to the dashboard:

A screenshot of a cell phone

Description automatically generated

Dashboard is only available via localhost. We can not log into the dashboard from the host system

A picture containing table, bird

Description automatically generated

So, we can create SSH tunnel or enable skip login. Here shows the second option:

|  |
| --- |
| kubectl edit deployment/kubernetes-dashboard --namespace**=**kube-system |

Add --enable-skip-login to the deployment’s command line arguments:

|  |
| --- |
| containers:  - args:  - --auto-generate-certificates  - --enable-skip-login *# <-- add this line* |

Now refresh the dashboard and click on the “skip” button. You will be correctly logged into the dashboard:

A screenshot of a social media post

Description automatically generated

### **Task 13: Publish your work**

push all files/procedures in your local gitlab repo to remote github repo (e.g. <https://github.com/your_github_id/go-web-hello-world>)

please send it to bo.cui@ericsson.com, no later than two calendar days after the interview.