National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Anjalee
Student ID: x19107803

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anjalee
Student ID:	x19107803
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	Anjalee
Date:	16th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anjalee
x19107803

1 Create Cluster on Digital Ocean

The steps for creating a Kubernetes cluster on DigitalOcean using Control Panel are:

Step 1: From the **Create** menu, select **Clusters** option. The page will be shown as:

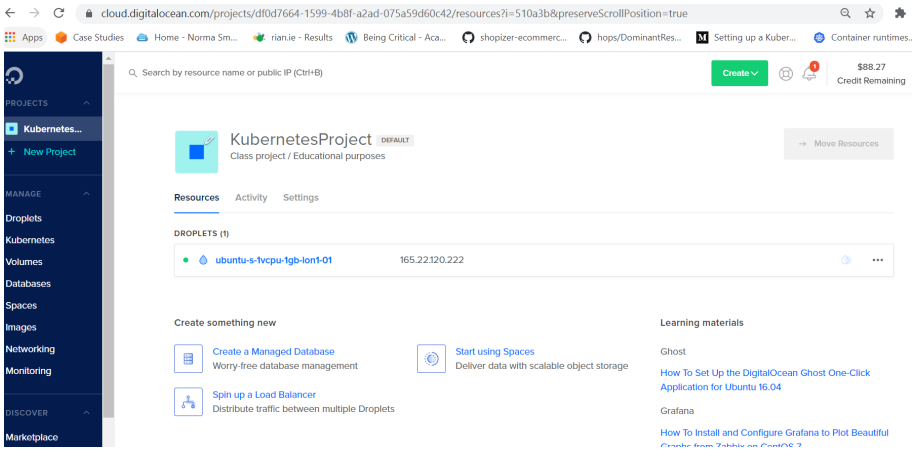
1

[1] Creating Clustershttps://www.digitalocean.com/docs/kubernetes/how-to/create-clusters/

Step 2: Then, select the latest Kubernetes version and nearset datacenter region.

Step 3: Select the VPC network related to the cluster.According to the requirement, the cluster capacity is selected. The K8s, cluster ID and resource type tags are by default to the worker nodes.

Step 4: Finally select the **Create Cluster** button. The control panel will show the cluster as :



2 Kubectl and Doctl set up in Ubuntu

[2] The command for installing doctl for Digital Ocean in ubuntu machine are:

#sudo snap install doctl

Then, download the doctl and copy the URL to get the file in home directory using curl.

cd curl -OL https://github.com/digitalocean/doctl/releases/download/v1.46.0/doctl-1.46.0-linux-amd64.tar.gz

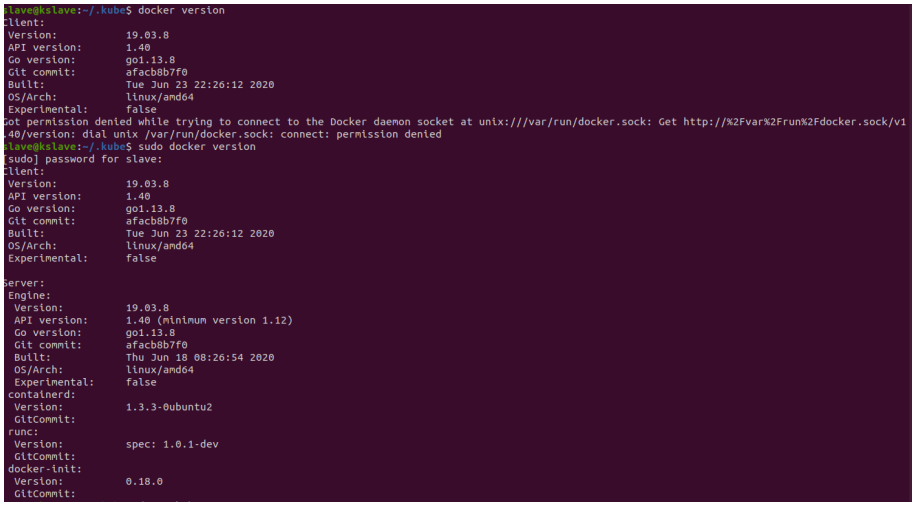
For Extracting use : **tar xf /doctl-1.46.0-linux-amd64.tar.gz**
Then, check if the docker is configured or not.

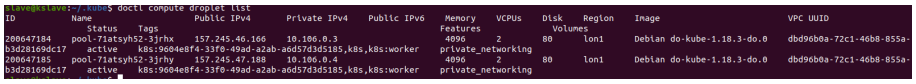
Authenticate the connection with the command :

 # doctl auth init

Then, the generated token is used to authenticate and connect to the DO account. To verify if doctl is working use the below command:

 # doctl compute droplet list

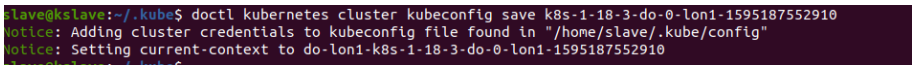




For Kubernetes setup, following commands were used:

#sudo snap connect doctl:kube-config __mkdir -p $HOME.kube__ sudo #chown $(id -u):$(id -g) $HOME/.kube

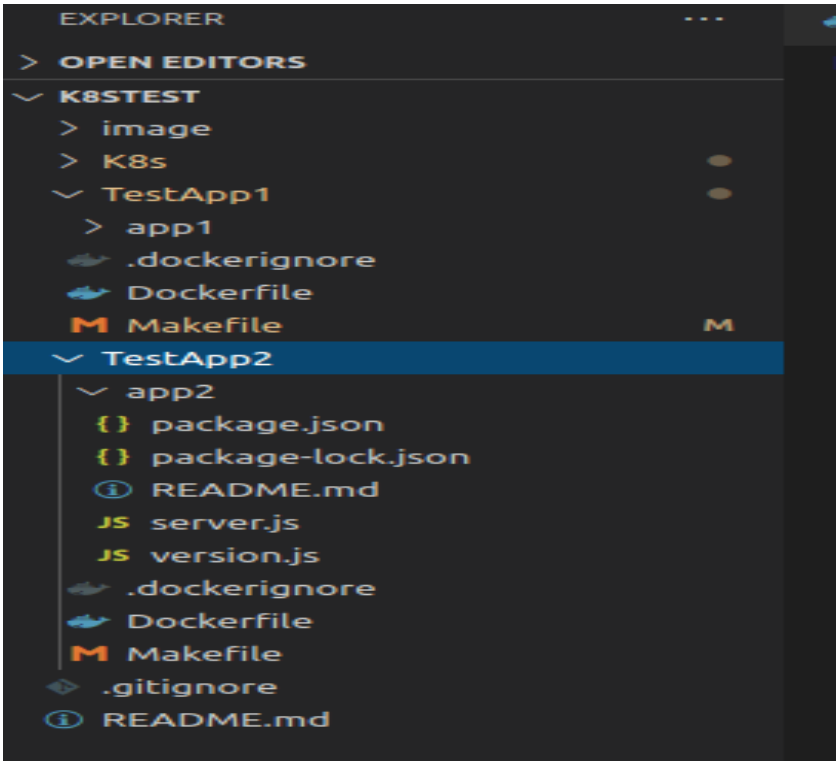
Then directory is created for storing configuration.
#mkdir -p $HOME/.kube__sudo chown $(id -u):$(id -g) $HOME/.kube
With doctl, Kubernetes configuration can be saved.
#doctl kubernetes cluster kubeconfig save k8s-1-18-3-do-0-lon1-1595187552910
[3]



3 Install Docker in Ubuntu and Containerize the app

Step 1: Create folder "app1" and "app2" that will contain the web app TestApp1 and TestApp2 respectively. To containerize the app, "Dockerfile" is created as shown in the below figure. Also, "Makefile" is created to automate the commands.

[2] https://kubectl.docs.kubernetes.io/
[3] Doctl Set uphttps://github.com/digitalocean/doctl



Step 2: To containerize the app, following commands should be used.

This command will build the image. **# sudo make build**

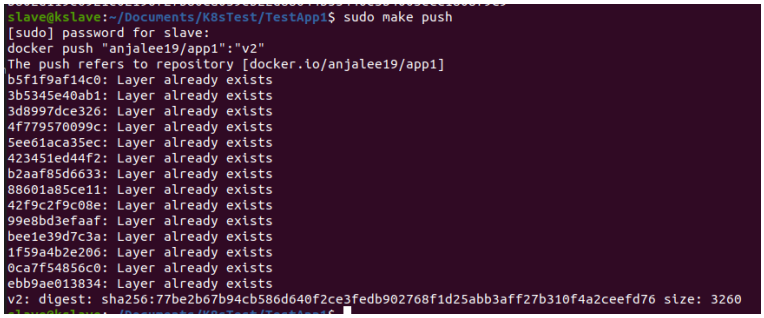
```
slave@kslave:~/Documents/K8sTest/TestApp1$ sudo make build
docker build -t "anjalee19/app1":"v2" .
Sending build context to Docker daemon  25.09kB
Step 1/8 : FROM node:8
 ---> 8eeadf3757f4
Step 2/8 : RUN apt update
 ---> Using cache
 ---> 672d821c8647
Step 3/8 : WORKDIR /opt/apis
 ---> Using cache
 ---> 544016c4939e
Step 4/8 : COPY app1/package.json /opt/apis
 ---> Using cache
 ---> b5d1f88a91b6
Step 5/8 : RUN npm install
 ---> Using cache
 ---> b150b53f761a
Step 6/8 : COPY app1/ ./
 ---> Using cache
 ---> 60800cd5fdc4
Step 7/8 : EXPOSE 3000
 ---> Using cache
 ---> aa8dfaaeb04a
Step 8/8 : CMD [ "node", "server.js" ]
 ---> Using cache
 ---> 7607a552e8d3
Successfully built 7607a552e8d3
Successfully tagged anjalee19/app1:v2
```

Once the build is done properly, run the command from the Makefile created. **# sudo make run**

```
slave@kslave:~/Documents/K8sTest/TestApp1$ sudo make run
docker run -p 3000:3000 --name "app1" -d "anjalee19/app1":"v2"
b802d119e6921e02196f27b80ca039eb22d88644b55440e3b4003ecc1808f9c9
```

Push the image to docker hub repo, the repo created on DockerHub as anjalee19/app1. Now, the image can be pulled and run using Kubernetes. Similar process should be done to push app2 image to DockerHub.

 # sudo make push



4

4 Creation of Pods

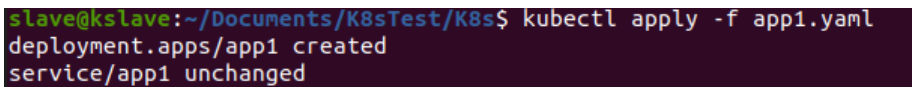
Step 1: The doctl set up is done and cluster is running with 2 nodes, but there is no container running in it. The containerized "app1" and "app2" should be executed to create pods. Kubernetes resource configs are used to create pods.

Step 2: The secret.yaml is created, which has the docker hub credential.

Step 3: Login with docker hub credential. **# sudo docker login**

Step 4: The /.docker/config.json will be created in the home directory. The auth key used in the secret.yaml is generated by following command:
 # sudo cat /.docker/config.json — base64 -w0
The name "dockerpullsecret" will be used for deployment as shown in the figure.

Step 5: The deployment and the service will be applied for app1 and app2 by executing file app1.yaml and app2.yaml respectively. Use below commands to apply app1.yaml app2.yaml:
 # kubectl apply -f app1.yaml
 # kubectl apply -f app2.yaml



[4] https://cloud.google.com/kubernetes-engine/docs/concepts/pod/

```
slave@kslave:~/Documents/K8sTest/K8s$ kubectl apply -f app2.yaml
deployment.apps/app2 created
service/app2 created
```

Step 6: Then with **#kubectl get all** command will show all the pods and services in the Kubernetes.

```
slave@kslave:~/Documents/K8sTest/K8s$ kubectl get all
NAME                        READY   STATUS    RESTARTS   AGE
pod/app1-bd4d77f77-crg8z    1/1     Running   0          42s
pod/app1-bd4d77f77-p27lj    1/1     Running   0          42s
pod/app2-5bb54d57fd-x6rtx   1/1     Running   0          7m33s
pod/app2-5bb54d57fd-zcpkj   1/1     Running   0          7m33s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/app1         ClusterIP   10.245.230.192   <none>        80/TCP    2d16h
service/app2         ClusterIP   10.245.147.108   <none>        80/TCP    20h
service/kubernetes   ClusterIP   10.245.0.1       <none>        443/TCP   13d

NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/app1   2/2     2            2           43s
deployment.apps/app2   2/2     2            2           7m33s

NAME                              DESIRED   CURRENT   READY   AGE
replicaset.apps/app1-bd4d77f77    2         2         2       43s
replicaset.apps/app2-5bb54d57fd   2         2         2       7m33s
```

As shown in the figure, the Cluster IPs 10.245.230.192 and 10.245.147.108 are used for the service. The app1 and app2 are available internally at 10.245.230.192 and 10.245.147.108 respectively at port 80. The container ports are 3000 and 3001. [5]

5 Setting Up Nginx Ingress controller

Step 1: First step is to create resources needed by this controller, the maintenance is done by Nginx. The command to create such resources are:
#kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/nginx-0.30.0/deploy/static/mandatory.yaml

Step 2: Ingress Controller Service is created that will balance load. The cloud-generic.yaml file contains the definition of service. The **externalTrafficPolicy: Cluster**, the command to apply this config file is:
kubectl apply -f cloud-generic.yaml

Step 3: Verify the Controller pods are started with the command:
kubectl get pods –all-namespaces -l app.kubernetes.io/name=ingress-nginx

```
slave@kslave:~/Documents/K8sTest/K8s$ kubectl get pods --all-namespaces -l app.kubernetes.io/name=ingress-nginx
NAMESPACE       NAME                                        READY   STATUS    RESTARTS   AGE
ingress-nginx   nginx-ingress-controller-5bb8fb4bb6-7wk9p   1/1     Running   0          44h
```

The command **kubectl get svc –namespace=ingress-nginx** will give the ports 80 and 443.

[5] https://stackoverflow.com/a/54322869

```
slave@kslave:~/Documents/K8sTest/K8s$ kubectl get svc --namespace=ingress-nginx
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)                      AGE
ingress-nginx   LoadBalancer   10.245.29.16   188.166.137.37   80:32342/TCP,443:32595/TCP   44h
```

Step 4: Then generate Ingress for app1 and app2 resource. Apply app1app2_ingress.yaml file to Kubernetes.
kubectl apply -f app1app2_ingress.yaml

```
slave@kslave:~/Documents/K8sTest/K8s$ kubectl describe ingress
Name:             app1app2-ingress
Namespace:        default
Address:          188.166.137.37
Default backend:  default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
TLS:
  tls-secret terminates app1.clenet.tech,www.app1.clenet.tech,app2.clenet.tech,www.app2.clenet.tech
Rules:
  Host                 Path  Backends
  ----                 ----  --------
  app1.clenet.tech
                          app1:80 (10.244.0.109:3000,10.244.0.135:3000)
  www.app1.clenet.tech
                       /              app1:80 (10.244.0.109:3000,10.244.0.135:3000)
                       /app1_default  app1:80 (10.244.0.109:3000,10.244.0.135:3000)
  app2.clenet.tech
                          app2:80 (10.244.0.246:3001,10.244.0.4:3001)
  www.app2.clenet.tech
                       /              app2:80 (10.244.0.246:3001,10.244.0.4:3001)
                       /app2_default  app2:80 (10.244.0.246:3001,10.244.0.4:3001)
Annotations:           cert-manager.io/cluster-issuer: letsencrypt-production
                       kubernetes.io/ingress.class: nginx
                       nginx.ingress.kubernetes.io/rewrite-target: /
```

6 Install and Configure Cert-Manager

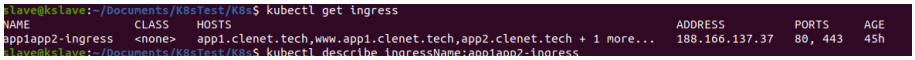
Step 1: Before installing Cert-Manager, first create Namespace to run it. The command used is: **kubectl create namespace cert-manager**

Step 2: Then, install Custom Resource Definitions related to cert-manager with the command:
kubectl apply –validate=false -f https://github.com/jetstack/cert-manager/releases/
manager.yaml
Verify the installed cert-manager with the command:
kubectl get pods –namespace cert-manager

```
slave@kslave:~/Documents/K8sTest/K8s$ kubectl get pods --namespace cert-manager
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-85db5c4c87-mlkdv              1/1     Running   0          28s
cert-manager-cainjector-7959549c78-mvw7z   1/1     Running   0          28s
cert-manager-webhook-5c8696f555-rk4xj      1/1     Running   0          28s
```

7 Implement Custom Scheduler- FFMRA

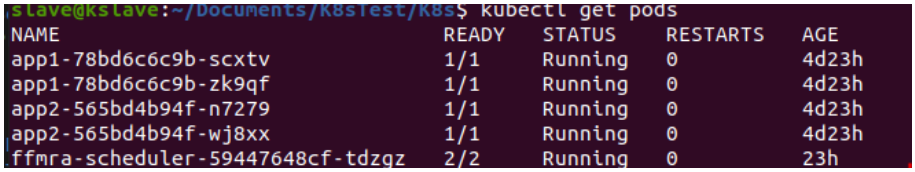
Step 1: The **scheduler.yaml** is created in the directory K8sTest defining the name of the custom scheduler and the pods in the container to be deployed. The **ffmra.go** is created where the custom scheduler logic is written.



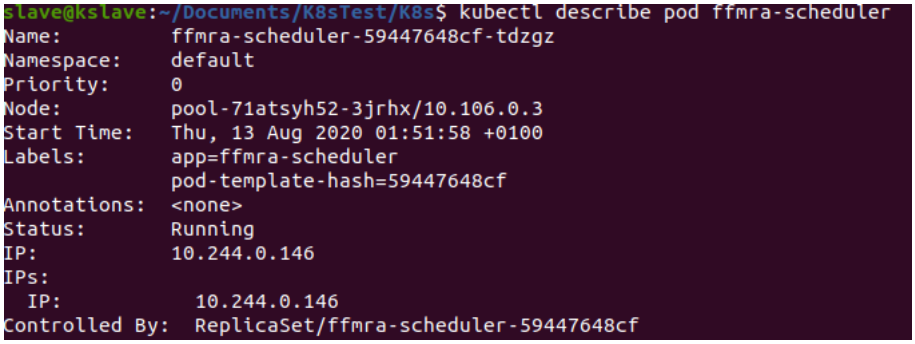
Step 2: With Dockerfile, scheduler's container image is build.

Step 3: Create a deployment with the custom scheduler code using the command: **kubectl apply -f scheduler.yaml**

Step 4: Verify if the scheduler is working on the pods. **kubectl get pods** The output will be as:



Step 5: The scheduler is running and using **kubectl describe pod ffmra-scheduler** to verify the status of the running scheduler.



Step 6: The status can be checked on Kubernetes dashboard. [6]

8 Kubernetes Monitoring Stack Set up Digital Ocean

[7] [8]

Step 1: The Digital Ocean monitoring stack is the integration of Grafana and Prometheus. These helps in monitoring the cluster, applications with graphs, charts, etc. Install Monitoring stack from the DO control panel and then follow the next steps.

[6] https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/
[7] https://kubernetes.github.io/ingress-nginx/user-guide/monitoring/
[8] https://www.digitalocean.com/community/tutorials/how-to-set-up-a-kubernetes-monitoring-stack-wit

Step 2: Download the config file and save it to the local machine's Downloads folder. Then, copy the file to kubectl directory. The commands used are:
 # cp /.kube/config /.kube/config.bkup
 # cp /Downloads/k8s-1-18-3-do-0-lon1-1595187552910-kubeconfig.yaml /.kube/config

Step 3: To verify run **# kubectl get pods -A**, the result will be as:

```
prometheus-operator   alertmanager-prometheus-operator-alertmanager-0          2/2     Running   0
prometheus-operator   prometheus-operator-grafana-6dbf66d75b-pt2ws             2/2     Running   0
prometheus-operator   prometheus-operator-kube-state-metrics-69fcc8d48c-dz8l9  1/1     Running   0
prometheus-operator   prometheus-operator-operator-6895d99c87-w9296            2/2     Running   0
prometheus-operator   prometheus-operator-prometheus-node-exporter-lwwf6       1/1     Running   0
prometheus-operator   prometheus-operator-prometheus-node-exporter-w9p72       1/1     Running   0
prometheus-operator   prometheus-prometheus-operator-prometheus-0              3/3     Running   1
```

Step 4: Set up Grafana as IP is not accessible publicly, so port-forwarding can be used.
 # kubectl -n prometheus-operator get pods — grep prometheus-operator-grafana
#kubectl port-forward prometheus-operator-grafana-6dbf66d75b-pt2ws -n prometheus-operator 8080:3000

Step 5: Then, login to Grafana dashboard at 127.0.0.1:8080 with username as **"admin"** and password as **"prom-operator"**. The dashboard will look as: