Configuration Manual

MSc Research Project
Cloud Computing

Manisha Chandra
Student ID: x21117284

School of Computing
National College of Ireland

Supervisor: Dr. Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Manisha Chandra
Student ID:	x21117284
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Aqeel Kazmi
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	2000
Page Count:	9

Signature:	C Manisha
Date:	14th December 2022

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.	☐

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

Configuration Manual

Manisha Chandra
x21117284

1 Introduction

This tutorial offers a straightforward explanation of how to construct a Kubernetes cluster on an Ubuntu server. In part 2, we will discuss the instructions and technologies that are necessary in order to construct the cluster. By using the idea that several schedulers are simultaneously operating, which will be covered in the next section, we will have the ability to determine how to check both the custom scheduler and the default scheduler. We are going to walk over the code to have a better understanding of how it operates in accordance with the suggested scheduler. At long last, we will look at the process of configuring the monitoring tools prometheus and node exporter.

2 Tools and Technologies Required

The table that follows provides us with the prerequisite criteria that must be met before we can go on with the implementation of the proposed scheduler.

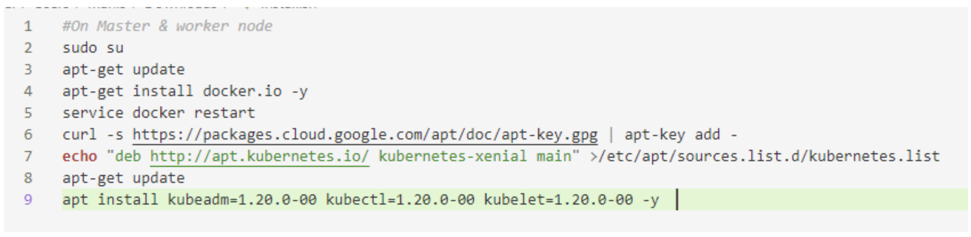
Design-Specification	
Tools & Technologies	Specification
Clustering Platform	AWS EC2
Application-Container	Ngnix
Operating System (OS)	Ubuntu Server 20.04 LTS
Container software	Docker version 20.10.12
Container orchestrator Platform	Kubernetes 1.20.0
tools for Monitoring	Prometheus and node exporter
Number of CPUs for Slave & Master	1 and 2 respectively
Coding language used	GoLang 1.18
Storage	minimum 4GiB memory
communication between pods and nodes	YAML

3 Clustering using Kubernetes

For this research , I am using AWS EC2 services in order to take the advantages provided by the cloud services.
I have decided to utilize Ubuntu Server 20.02 LTS since it will facilitate the formation of kubernetes clusters. I would ask that you refrain from using version 22.02 because it is not compatible with this endeavor.

3.1 Nodes Creation

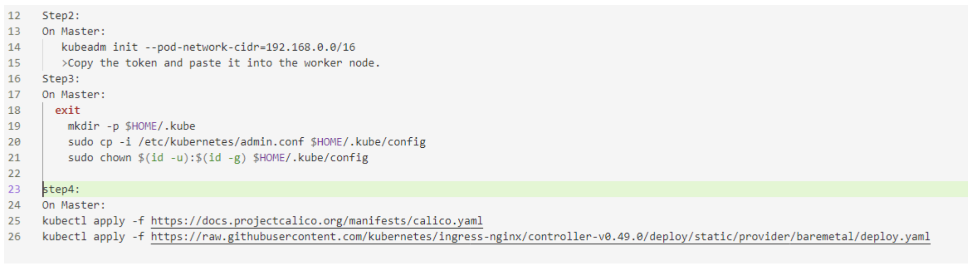
We will require one master node in addition to two slave nodes in order to carry out this investigation. Since there is a significant amount of processing work involved, the master node will be constructed using a T2.Medium (which has 2 CPUs), while the slave nodes will be formed using a T2.micro, which has just one CPU. In the below figure we can the commands to install Docker and kubernetes and also kubeadm in order to form the cluster. this commands can be written in a single shell script to make things easy. This



```
#On Master & worker node
sudo su
apt-get update
apt-get install docker.io -y
service docker restart
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" >/etc/apt/sources.list.d/kubernetes.list
apt-get update
apt install kubeadm=1.20.0-00 kubectl=1.20.0-00 kubelet=1.20.0-00 -y
```

Figure 1: Shell script code for installing prerequisites

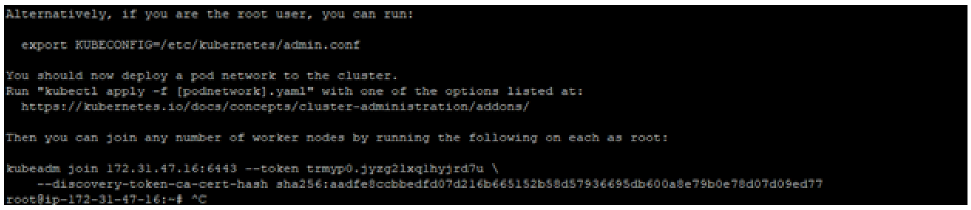
file needs to be run in all the nodes including master in order to form the cluster. first we need to run the below command.
sudo -i
the we need to run the shell script.After we need to run the commad provided in the figure 2.
Step-2 which will generate kubead token which needs to be used in the slave nodes.The token will be generated as shown in the figure 3



```
Step2:
On Master:
   kubeadm init --pod-network-cidr=192.168.0.0/16
   >Copy the token and paste it into the worker node.
Step3:
On Master:
  exit
    mkdir -p $HOME/.kube
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
    sudo chown $(id -u):$(id -g) $HOME/.kube/config

step4:
On Master:
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.49.0/deploy/static/provider/baremetal/deploy.yaml
```

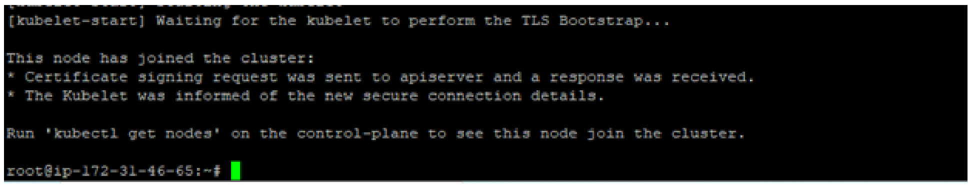
Figure 2: Steps for creating cluster that needs to be executed in master node

After this, we are required to copy the token, and we must ensure that this process is completed as rapidly as possible since the token has a time limit.
Step-3
The token is then executed on the slave nodes, after which we will be presented with the command that we have to execute on the master node in order to validate that the cluster has been created.This can be shown in the figure 4.



```
Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.47.16:6443 --token trmyp0.jyzg21xqlhyjrd7u \
    --discovery-token-ca-cert-hash sha256:aadfe8ccbbedfd07d216b665152b58d57936695db600a8e79b0e78d07d09ed77
root@ip-172-31-47-16:~# ^C
```

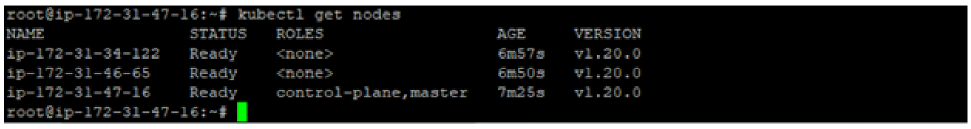
Figure 3: kubeadm token in order to form cluster



```
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@ip-172-31-46-65:~#
```

Figure 4: The nodes have joined the cluster

Step-4:
Before running the command
Kubectl get nodes
we need to fallow the steps 3 and 4 provided in the master node so that we give permission for these nodes to communicate and form the cluster. After these steps are executed when we provide the get nodes command we will the nodes have been joined the cluster this is shown in the below figure 5.



```
root@ip-172-31-47-16:~# kubectl get nodes
NAME               STATUS   ROLES                  AGE     VERSION
ip-172-31-34-122   Ready    <none>                 6m57s   v1.20.0
ip-172-31-46-65    Ready    <none>                 6m50s   v1.20.0
ip-172-31-47-16    Ready    control-plane,master   7m25s   v1.20.0
root@ip-172-31-47-16:~#
```

Figure 5: kubeadm command to get the nodes from the master node

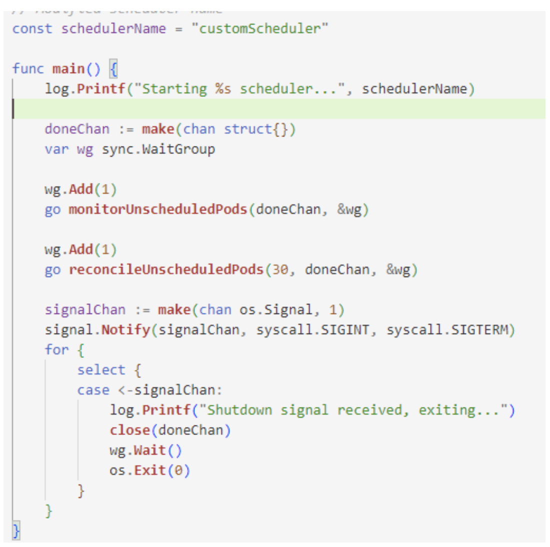
3.2 Kubernetes UI Dashboard

The Kubernetes portal is where the clusters node bindings, deployments binding, and applications binding are located.
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.6.1/aio/deploy/recommended.yaml
Utilizing the kubectl command line interface and executing the command line, you will be able to authorize access to a Dashboard. **kubectl proxy**
The dashboard will be available at the below link.
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/ services/https:kuberne
dashboard:/proxy/
If facing any issues kindly check the fallowing website:[1]

[1]https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

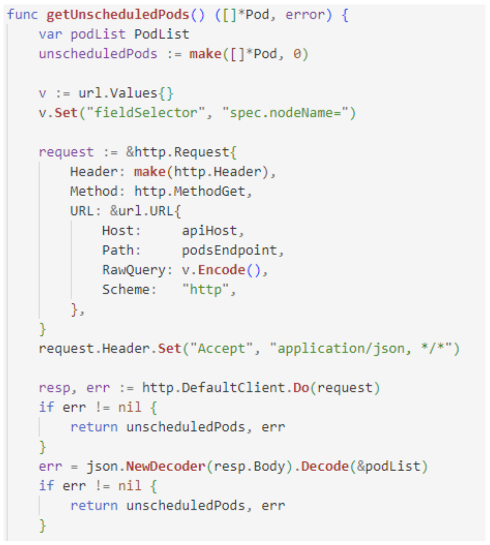
4 Scheduler Code Explanation

Once we clone the scheduler folder present in the code artifacts we can see the fallowing folders of files which are implemented to perform scheduling according to the proposed way.
The execution of the scheduler will start with main.go file first as it does the monitoring part. **monitorUnscheduledpods()** is a method that can be found inside the main.go file (Kindly check the code artifacts) transfer the file to . After constantly monitoring for every requests which have been generated by the user, it will set up a pod as well as assign a nodes to it. After all this, the pods will be ready for usage. The code can be shown in the figure 6.



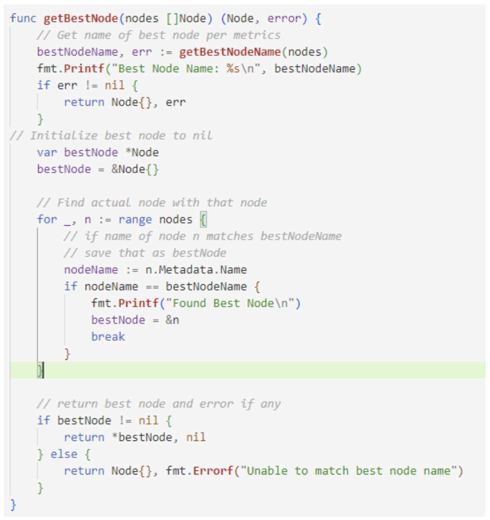
```go
const schedulerName = "customScheduler"

func main() {
    log.Printf("Starting %s scheduler...", schedulerName)

    doneChan := make(chan struct{})
    var wg sync.WaitGroup

    wg.Add(1)
    go monitorUnscheduledPods(doneChan, &wg)

    wg.Add(1)
    go reconcileUnscheduledPods(30, doneChan, &wg)

    signalChan := make(chan os.Signal, 1)
    signal.Notify(signalChan, syscall.SIGINT, syscall.SIGTERM)
    for {
        select {
        case <-signalChan:
            log.Printf("Shutdown signal received, exiting...")
            close(doneChan)
            wg.Wait()
            os.Exit(0)
        }
    }
}
```

Figure 6: main.go file present in the scheduler folder



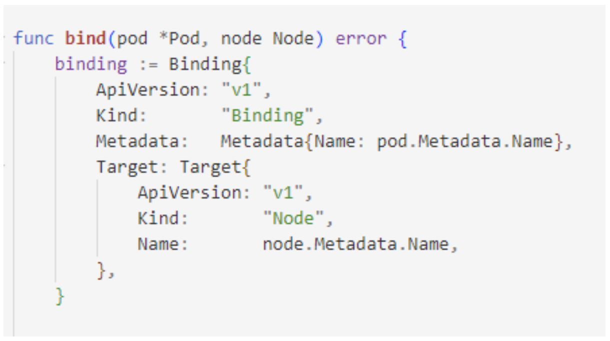
```go
func getUnscheduledPods() ([]*Pod, error) {
    var podList PodList
    unscheduledPods := make([]*Pod, 0)

    v := url.Values{}
    v.Set("fieldSelector", "spec.nodeName=")

    request := &http.Request{
        Header: make(http.Header),
        Method: http.MethodGet,
        URL: &url.URL{
            Host:      apiHost,
            Path:      podsEndpoint,
            RawQuery: v.Encode(),
            Scheme:    "http",
        },
    }
    request.Header.Set("Accept", "application/json, */*")

    resp, err := http.DefaultClient.Do(request)
    if err != nil {
        return unscheduledPods, err
    }
    err = json.NewDecoder(resp.Body).Decode(&podList)
    if err != nil {
        return unscheduledPods, err
    }
```

Figure 7: Code for getUnscheduled-Pods() method



```go
func getBestNode(nodes []Node) (Node, error) {
    // Get name of best node per metrics
    bestNodeName, err := getBestNodeName(nodes)
    fmt.Printf("Best Node Name: %s\n", bestNodeName)
    if err != nil {
        return Node{}, err
    }
// Initialize best node to nil
    var bestNode *Node
    bestNode = &Node{}

    // Find actual node with that node
    for _, n := range nodes {
        // if name of node n matches bestNodeName
        // save that as bestNode
        nodeName := n.Metadata.Name
        if nodeName == bestNodeName {
            fmt.Printf("Found Best Node\n")
            bestNode = &n
            break
        }
    }

    // return best node and error if any
    if bestNode != nil {
        return *bestNode, nil
    } else {
        return Node{}, fmt.Errorf("Unable to match best node name")
    }
}
```

Figure 8: code for getBestNode() method



```go
func bind(pod *Pod, node Node) error {
    binding := Binding{
        ApiVersion: "v1",
        Kind:       "Binding",
        Metadata:   Metadata{Name: pod.Metadata.Name},
        Target: Target{
            ApiVersion: "v1",
            Kind:       "Node",
            Name:       node.Metadata.Name,
        },
    }
```

Figure 9: code for bind() method

Next comes the step of getting all the unscheduled list pods, here annotations are used to match the pods to the response. check the figure 7 for the implementation.
The process from here goes to the **SchedulePods** method which in turn calls the **SchedulePod** where the input is given as each of the pods which needs to be scheduled. This method will in turn call the **getBestNode** and **Bind** methods in the figures 9 and 9 The fit() method is composed of two different functional components: first, it will accept the input, which is a pod that has been given by the schedulePod() method, conduct the proper filtering, and determine the set of nodes that is most suited.
This list of nodes is then sent along as an input to the getBestNode() method, which is the implementation of how to get the metric data of the nodes from the prometheus server API. Based on the collected metric data, this strategy chooses the node that is deemed to confer the most benefit onto the network as a whole.
Bind() This function takes as arguments both the pod that has been chosen by the schedulerPod() method and the best node that has been chosen by the getBestNode() method. The method then binds the pod to the best node that has been chosen. This concludes the procedure that was previously followed.

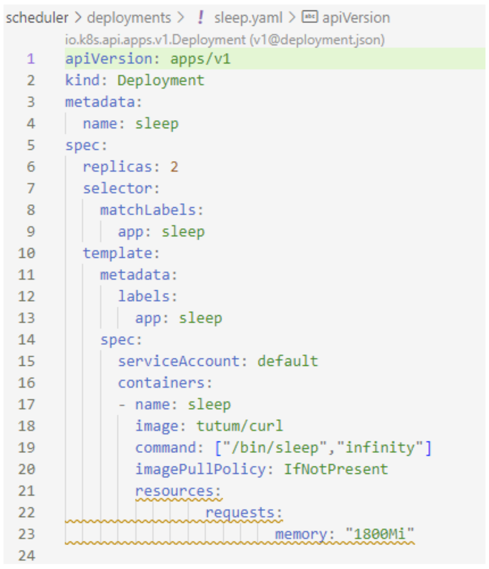
In order to run the scheduler we need to download the go language in the server of AWS EC2.
sudo apt install golang-go

5 Deployment of the pods

We install five pods in order to compare the performance of the custom scheduler to that of the default scheduler in Kubernetes. These pods are named as follows: Sleep (two replicas), Nginx (one replica) on both the default scheduler and the custom scheduler, including sys (1 replica).
In order to create a Pod, we need to declare the API type as V1, and the pod's type ought to be as depicted in the image below. The 'name' field in metadata is what's utilized for identifying purposes, regardless of the specification. The customScheduler scheduler has to be specified as the schedulerName.
As we see the figure 10 and 11, these two yaml files are used to test the working of the schedulers. The resting time. yaml is indeed a pod which demands 1800Mi of storage and has an unlimited capacity for sleeping. In all, we need to deploy two of sleep pods and make a request of 3,600 megabytes of RAM. The default for the test. yaml triggers the execution of the Nginx app using the standard scheduler as well as by using testcustoms.yaml. The Nginx app is executed by yaml just on user-defined scheduler.

5.1 Steps to be fallowed in order to create pods

Following the formation of pod, we can generate pods by using **kubectl create -f file/podname.yaml** command.
If we run the command **kubectl get pods** the current condition of a pods is displayed


scheduler > deployments > ! sleep.yaml > apiVersion
io.k8s.api.apps.v1.Deployment (v1@deployment.json)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sleep
  template:
    metadata:
      labels:
        app: sleep
    spec:
      serviceAccount: default
      containers:
      - name: sleep
        image: tutum/curl
        command: ["/bin/sleep","infinity"]
        imagePullPolicy: IfNotPresent
        resources:
                requests:
                        memory: "1800Mi"

```

Figure 10: sleep.yaml

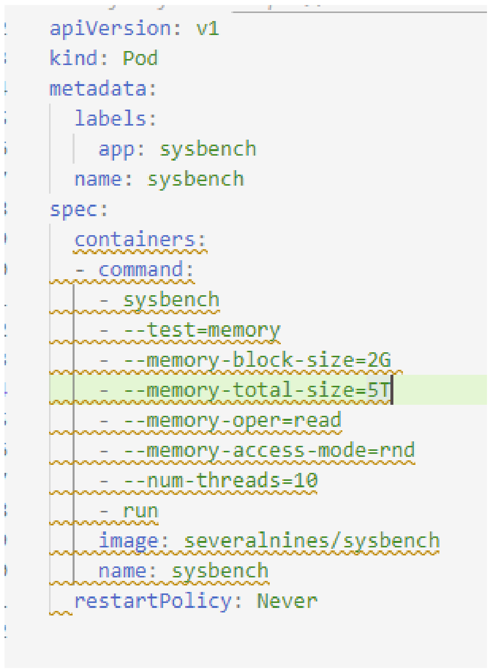

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: sysbench
  name: sysbench
spec:
  containers:
  - command:
    - sysbench
    - --test=memory
    - --memory-block-size=2G
    - --memory-total-size=5T
    - --memory-oper=read
    - --memory-access-mode=rnd
    - --num-threads=10
    - run
    image: severalnines/sysbench
    name: sysbench
  restartPolicy: Never
```

Figure 11: sys.yaml

6 Executing multiple schedulers

The default schedule that comes with Kubernetes is outlined in this article. You are free to create your custom scheduler in the event that the one that comes as standard does not meet your requirements. In addition, you are able to run numerous schedulers concurrently in addition to a default scheduler, and you may direct Kubernetes to utilize a specific scheduler for each the pods. This feature is available with the default scheduler.

6.1 Put the scheduler into a package.

We need to execute the fallowing commands:
git clone
https://github.com/kubernetes/kubernetes.git
cd kubernetes
make
Make that the kube scheduler binary is included in the docker container that you create. The Dockerfile that will be used to construct the image:
FROM busybox
ADD ./_output/local/bin/linux/amd64/kube-scheduler /usr/local/bin/kube-scheduler
Now that the configuration has been created, we will need to launch it on the Kubernetes cluster.

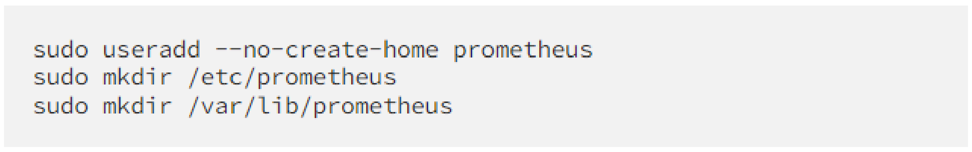
6.2 Start the cluster's secondary scheduling service

Create a deployment described inside the config above for a Kubernetes cluster in order to execute your scheduler there:
kubectl create -f my-scheduler.yaml

Make sure the scheduler pod is active by doing the following:
kubectl get pods –namespace=kube-system
Check the fallowing kubernetes official document for further assistance [2]

7 Configuration of Monitoring Tools

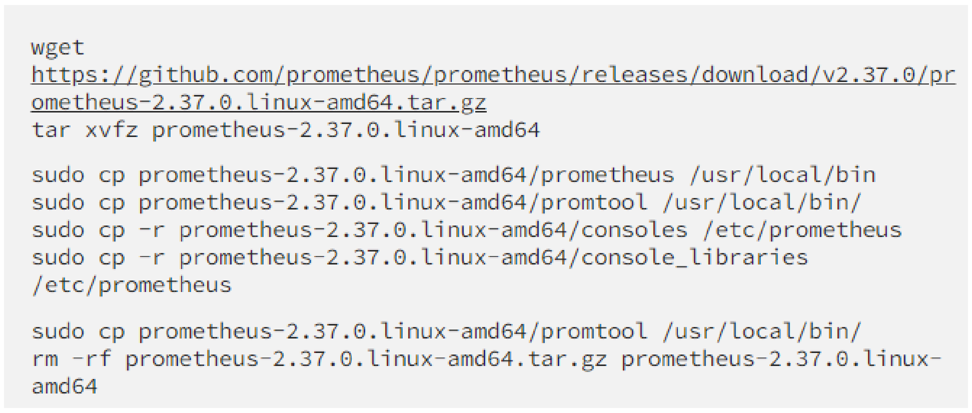
EC2 Instance Launch
In this step while creating the EC2 instance we need to add port number **9090** in order to allow the traffic of prometheus.
Prometheus Install When it comes to running certain services, it is advised that a user other than root be created. This will assist in isolating Prometheus as well as adding an additional layer of defense to the system. In addition to this, we have to create two directories: one to store the setup for Prometheus, but another to store the data it generates.



```
sudo useradd --no-create-home prometheus
sudo mkdir /etc/prometheus
sudo mkdir /var/lib/prometheus
```

Figure 12: Prometheus command for making directories

Installing the Prometheus is the next step now.

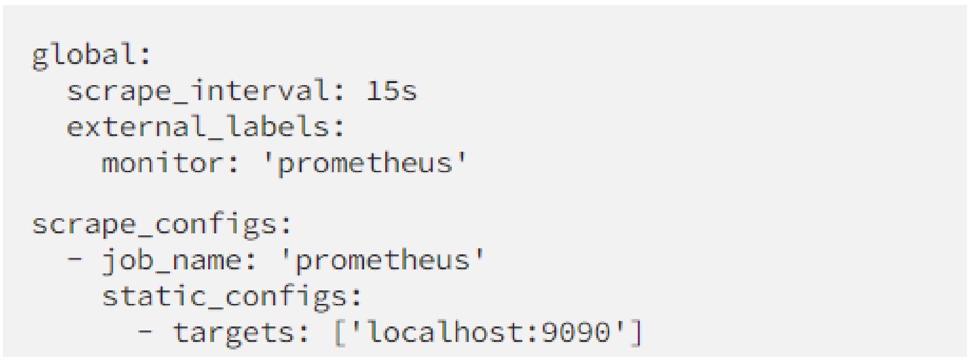


```
wget
https://github.com/prometheus/prometheus/releases/download/v2.37.0/pr
ometheus-2.37.0.linux-amd64.tar.gz
tar xvfz prometheus-2.37.0.linux-amd64

sudo cp prometheus-2.37.0.linux-amd64/prometheus /usr/local/bin
sudo cp prometheus-2.37.0.linux-amd64/promtool /usr/local/bin/
sudo cp -r prometheus-2.37.0.linux-amd64/consoles /etc/prometheus
sudo cp -r prometheus-2.37.0.linux-amd64/console_libraries
/etc/prometheus

sudo cp prometheus-2.37.0.linux-amd64/promtool /usr/local/bin/
rm -rf prometheus-2.37.0.linux-amd64.tar.gz prometheus-2.37.0.linux-
amd64
```

Figure 13: Prometheus Installing command

In the beginning, as an evidence of concept, we are able to enable Prometheus constantly monitor itself; but, in order to do so, we need to build or alter the data of **/etc/prometheus/prometheus.yml**

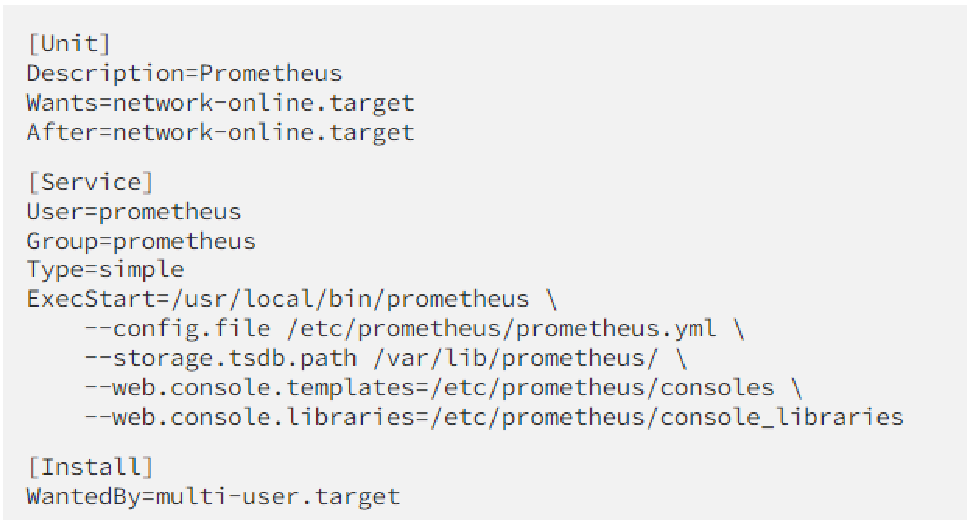
[2] `https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/`



```
global:
  scrape_interval: 15s
  external_labels:
    monitor: 'prometheus'

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

Figure 14: prometheus.yml

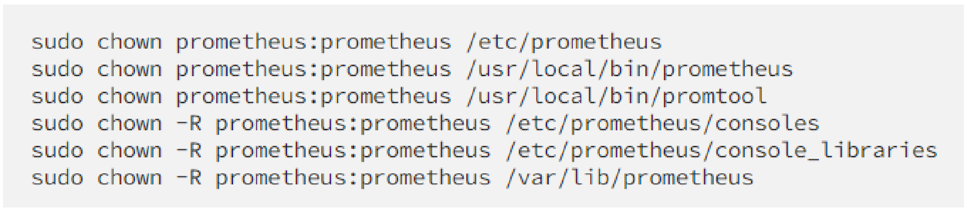
In this case, it would be helpful if Prometheus could be made accessible like a service. In order for Prometheus to launch the operating system each time the system is restarted. First, write the prometheus.service file in /etc/systemd/system. Now that we've



```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target
```

Figure 15: prometheus.service

introduced new directories, documents, and binary to our system, we need to make a few modifications to the permissions associated with those components.



```
sudo chown prometheus:prometheus /etc/prometheus
sudo chown prometheus:prometheus /usr/local/bin/prometheus
sudo chown prometheus:prometheus /usr/local/bin/promtool
sudo chown -R prometheus:prometheus /etc/prometheus/consoles
sudo chown -R prometheus:prometheus /etc/prometheus/console_libraries
sudo chown -R prometheus:prometheus /var/lib/prometheus
```

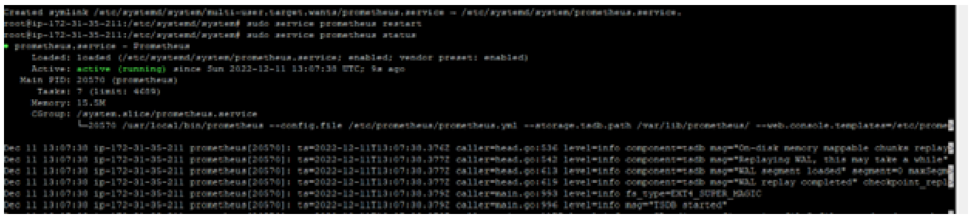
Figure 16: Modification to the permissions

Currently, the only thing left to do is setup the systemd.
In order to run the prometheus we need to restart it

```
sudo systemctl daemon-reload
sudo systemctl enable prometheus
```

Figure 17: Setting up the systemd

sudo service prometheus restart
sudo service prometheus status



```
Created symlink /etc/systemd/system/multi-user.target.wants/prometheus.service → /etc/systemd/system/prometheus.service.
root@ip-172-31-35-211:/etc/systemd/system# sudo service prometheus restart
root@ip-172-31-35-211:/etc/systemd/system# sudo service prometheus status
● prometheus.service - Prometheus
     Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2022-12-11 13:07:38 UTC; 5s ago
   Main PID: 20570 (prometheus)
      Tasks: 7 (limit: 4689)
     Memory: 15.5M
     CGroup: /system.slice/prometheus.service
             └─20570 /usr/local/bin/prometheus --config.file /etc/prometheus/prometheus.yml --storage.tsdb.path /var/lib/prometheus/ --web.console.templates=/etc/prome

Dec 11 13:07:38 ip-172-31-35-211 prometheus[20570]: ts=2022-12-11T13:07:38.376Z caller=head.go:536 level=info component=tsdb msg="On-disk memory mappable chunks replay
Dec 11 13:07:38 ip-172-31-35-211 prometheus[20570]: ts=2022-12-11T13:07:38.377Z caller=head.go:542 level=info component=tsdb msg="Replaying WAL, this may take a while"
Dec 11 13:07:38 ip-172-31-35-211 prometheus[20570]: ts=2022-12-11T13:07:38.377Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=0 maxSegm
Dec 11 13:07:38 ip-172-31-35-211 prometheus[20570]: ts=2022-12-11T13:07:38.377Z caller=head.go:619 level=info component=tsdb msg="WAL replay completed" checkpoint_repl
Dec 11 13:07:38 ip-172-31-35-211 prometheus[20570]: ts=2022-12-11T13:07:38.379Z caller=main.go:993 level=info fs_type=EXT4_SUPER_MAGIC
Dec 11 13:07:38 ip-172-31-35-211 prometheus[20570]: ts=2022-12-11T13:07:38.379Z caller=main.go:996 level=info msg="TSDB started"
```

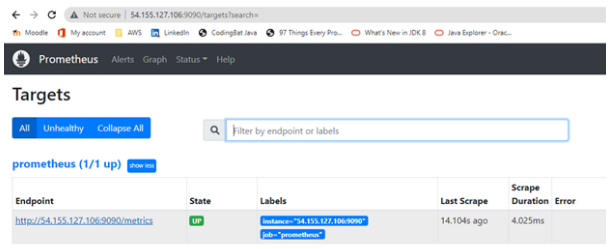
Figure 18: After we start the prometheus Server

Node Exporter installing and running The Node Exporter is indeed a single binary file that is static and can be installed using a tarball. After you have obtained it via the Prometheus download website and saved it to your computer, unzip this, and then execute it:

```
wget https://github.com/prometheus/node_exporter/releases/download/v*/node_exporter-*.*-amd64.tar.gz
tar xvfz node_exporter-*.*-amd64.tar.gz
cd node_exporter-*.*-amd64
./node_exporter
```

Figure 19: Node exporter installing command

This has to be done in the two slave nodes in order to get the nodes value in the master prometheus tool. After this is done we will be able to see the that nodes have been connected to the prometheus.



Targets

prometheus (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://54.155.127.106:9090/metrics	UP	instance="54.155.127.106:9090" job="prometheus"	14.104s ago	4.025ms	

Figure 20: Prometheus dashboard