CS 4530/7530 Cloud Computing I

Lab – Edge Computing with KubeEdge

Developed by: Dr. Songjie Wang (wangso@missouri.edu)

Contents

1 Overview **3**

2 References to guide lab work **3**

3 KubeEdge background **3**
3.1 Advantages of KubeEdge . . . 3
3.2 KubeEdge Architecture and Components . . . 4

4 Lab Topology **5**

5 Github Repo **5**

6 VM instance Preparation on GCP **5**
6.1 Log into your Google GCP account console . . . 6
6.2 Setup firewall policies for the networks in your project . . . 6
6.3 Create VM instances for each of the node in the topology setup . . . 7
6.4 Create VM instances for each of the node in the topology setup . . . 8
6.5 Once all the VM instances have been created and are up running, please note down the following information: . . . 9

7 Node pre-installation **9**
7.1 SSH login to the nodes using your SSH private key. . . . 9
7.2 Switch to sudo user . . . 9
7.3 Clone KubeEdge installation git repo . . . 9
7.4 Setup Root SSH login with password . . . 11
7.5 Setup root user password . . . 12

8 KubeEdge Installation on All Nodes **13**
8.1 Cloud core node installation . . . 13
8.2 Edge core node installation . . . 14
8.3 Setup remote login from Cloud node to Edge nodes, and copy certificates to Edge nodes . . . 14
8.4 Obtain KubeEdge secret token . . . 15
8.5 Join Edge core nodes to the KubeEdge cluster . . . 16

9 Deploy an Volunteer Edge Computing with Trusted Bioinformatics application, the OnTimeURB, in the KubeEge cluster **16**
9.1 OnTimeURB Overview . . . 16
9.2 Github repository for the OnTimeURB application . . . 16
9.3 Example workflow - Pgen workflow . . . 16
9.4 Labeling Edge core nodes . . . 17
9.5 Create deployments for example workflow . . . 18

10 Homework Problems **18**

11 Lab Report **18**

1 Overview

In this lab, we will explore KubeEdge system, an open source system for extending native containerized application orchestration capabilities to hosts at Edge. It is built upon kubernetes and provides fundamental infrastructure support for network, app. deployment and metadata synchronization between cloud and edge.

In this lab, we will learn how to setup KubeEdge capability on a public cloud, Google GCP. We will create a KubeEdge cluster that has a master node, and a few Edge nodes with different computing power to mimic different edge devices in a real world scenario. Once the cluster is setup, we will leverage a use case application - volunteer edge computing with Trusted bioinformatics, and show how we can leverage the KubeEdge cluster to orchestrate HPC workflows on selected edge nodes in the cluster.

2 References to guide lab work

Please use the links below to learn more about KubeEdge and to help you understand the lab contents.

- KubeEdge – *https://kubeedge.io/en/*
- KubeEdge documentation – *https://kubeedge-docs.readthedocs.io/en/latest/modules/kubeedge.html*
- KubeEdge git repo – *https://github.com/kubeedge/kubeedge/releases*
- Keadm installation – *https://kubeedge.io/en/*

For this lab, we assume that you already have some knowledge/experience with Docker and Kubernetes. Here are some references for these pre-requisites:

- Docker – *https://docs.docker.com*
- Docker Hub – *https://hub.docker.com/*
- Kubernetes – *https://kubernetes.io/*
- Kubectl overview – *https://kubernetes.io/docs/reference/kubectl*

3 KubeEdge background

KubeEdge is an open source system extending native containerized application orchestration and device management to hosts at the Edge. It is built upon Kubernetes and provides core infrastructure support for networking, application deployment and metadata synchronization between cloud and edge. It also supports MQTT and allows developers to author custom logic and enable resource constrained device communication at the Edge. KubeEdge consists of a cloud part and an edge part. The cloud part normally consists one master node and the edge part can include as many edge nodes based on user's requirement.

3.1 Advantages of KubeEdge

- Edge Computing With business logic running at the Edge, much larger volumes of data can be secured processed locally where the data is produced. This reduces the network bandwidth requirements and consumption between Edge and Cloud. This increases responsiveness, decreases costs, and protects customers' data privacy.
- Simplified development Developers can write regular http or mqtt based applications, containerize these, and run them anywhere - either at the Edge or in the Cloud - whichever is more appropriate.
- Kubernetes-native support With KubeEdge, users can orchestrate apps, manage devices and monitor app and device status on Edge nodes just like a traditional Kubernetes cluster in the Cloud

- Abundant applications It is easy to get and deploy existing complicated machine learning, image recognition, event processing and other high level applications to the Edge.

3.2 KubeEdge Architecture and Components

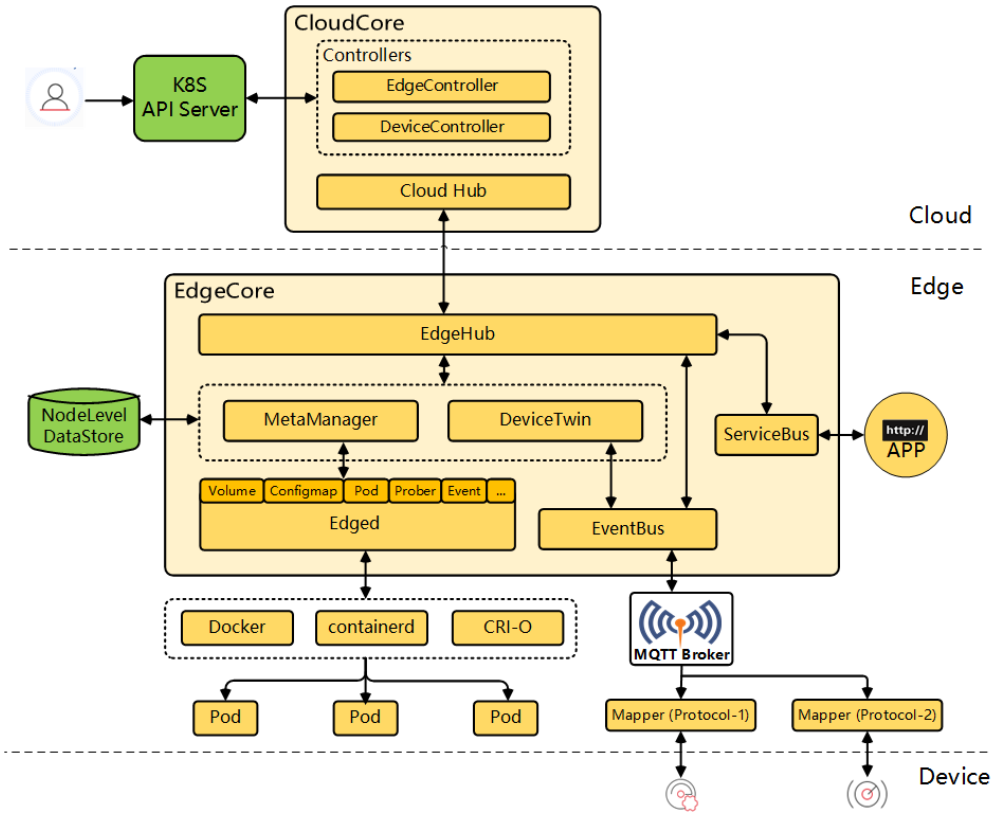


Figure 1: KubeEdge system architecture

KubeEdge is composed of these components:

- Edged: an agent that runs on edge nodes and manages containerized applications.
- EdgeHub: a web socket client responsible for interacting with Cloud Service for edge computing (like Edge Controller as in the KubeEdge Architecture). This includes syncing cloud-side resource updates to the edge and reporting edge-side host and device status changes to the cloud.
- CloudHub: A web socket server responsible for watching changes at the cloud side, caching and sending messages to EdgeHub.
- EdgeController: an extended kubernetes controller which manages edge nodes and pods metadata so that the data can be targeted to a specific edge node.
- EventBus: an MQTT client to interact with MQTT servers (mosquitto), offering publish and subscribe capabilities to other components.
- DeviceTwin: responsible for storing device status and syncing device status to the cloud. It also provides query interfaces for applications.

- MetaManager: the message processor between edged and edgehub. It is also responsible for storing/retrieving metadata to/from a lightweight database (SQLite).

4 Lab Topology

In this lab, we will setup a topology that has a Cloud core node, two edge sites each with three different VM instances with three different configuration of computing resources, i.e., small, medium, and large. The Cloud core node has 4 CPU and 4 GB of memory. On the Edge core sites, the mall VM instances have 1 CPU, and 1GB of memory, medium VM instances have 4 CPU, and 4GB of memory, and large VM instances have 16 CPU, and 16GB of memory.

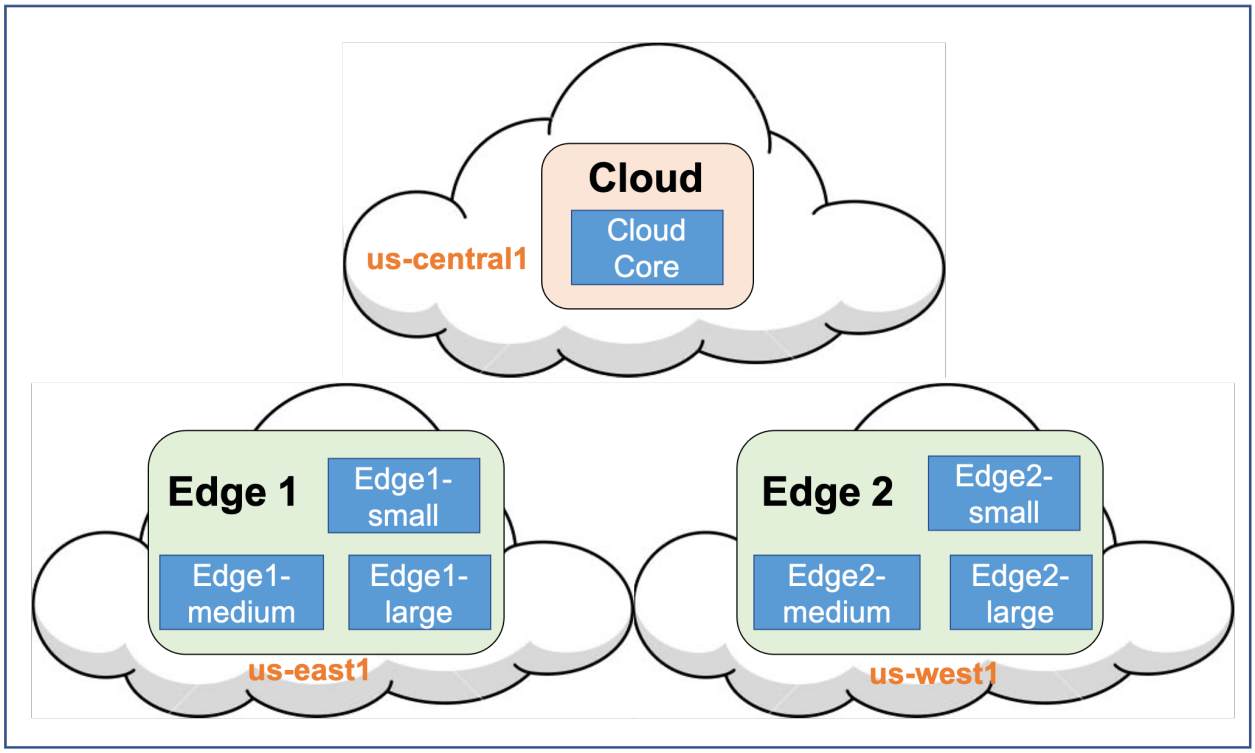


Figure 2: Lab topology

5 Github Repo

For the purpose of the lab setup, we have created a Github repository that has the scripts to install and configure KubeEdge on the Cloud/Edge core nodes. The link to the repo is at *https://github.com/wangso/KubeEdge-install.git*

6 VM instance Preparation on GCP

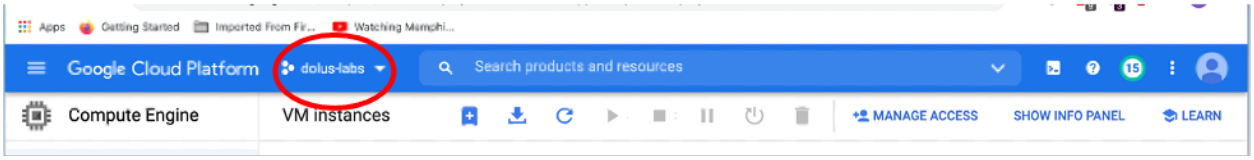
The first step of this lab to configure GCP cloud access. For this, you need to have

- A Google account
- A Google project

- (optional) Google SDK to allow command line access

6.1 Log into your Google GCP account console

- Navigate to your web console on GCP in the project:



6.2 Setup firewall policies for the networks in your project

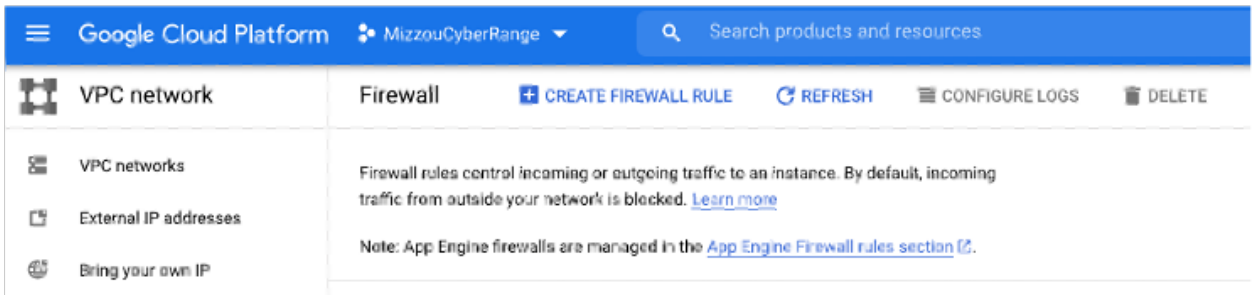
Warning

In this lab, it is essential that you configure your firewall policy in your VPC network. If you don't configure the policies correctly, your SDN network routing will not work!

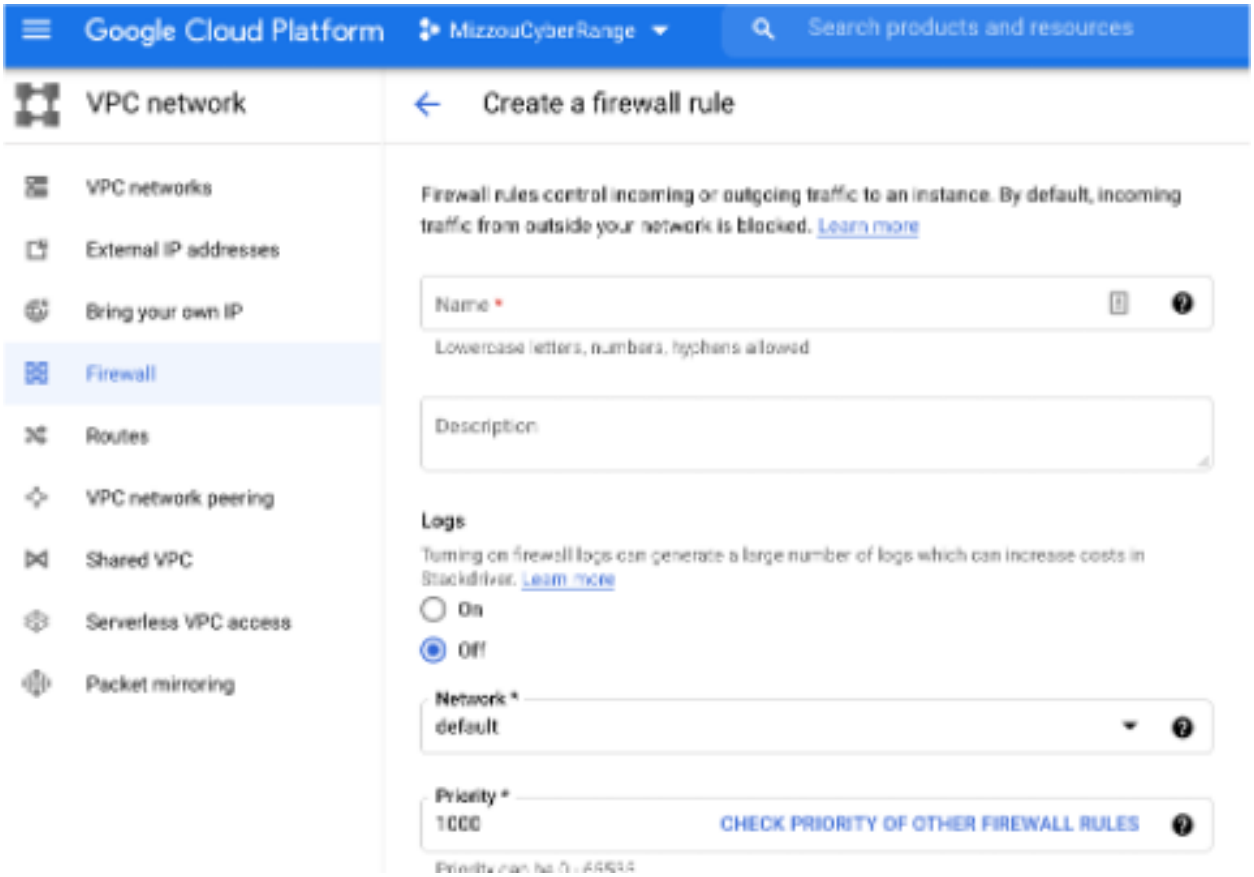
Note

In this lab, we will use the "default" VPC and subnet in your GCP account.

- Click the hamburger symbol, and select Networking $->$ Firewall.



- Click "Create Firewall Rule"
- Give the firewall rule a name, leave others as default, and fill out the Targets as indicated in the screen shot below and define source IP ranges as "0.0.0.0/0" to allow all IPs on the network to access the VPC network.
- Configure protocols and ports as indicated in the screen shot to open firewall ports required in the Dolus topology setup, i.e. ports 22, 80/8080, 5000 and 6643:
- Click "Create" to create the firewall policy.



6.3 Create VM instances for each of the node in the topology setup

After configuring VPC firewall policies, create the VM instances for KubeEdge topology setup.

Note

In this section, we guide you to create one of the VM instances (the Cloud core node) in the topology, and you will use repeat the steps with similar settings to create all the other instances.

- Click the hamburger symbol, and select Compute Engine $->$ VM instances.
- From the console, click the symbol to create new VM instance.
- Give an appropriate name to the VM instance. We recommend you to use the following naming convention: “username-nodename”, to easily remember what the instance is, e.g., “wangso-cloudcore”. Keep the default VM instance configuration as shown in the screenshot.
- In the ”Boot disk”, click Change. Leave the default Disk image as Debian GNU/Linux 10, and change Persistent disk size to 20GB. Click Select.
- Leave the Identity and API access to ”Allow full access to all Cloud APIs”, and check firewalls to allow both HTTP and HTTPs traffic.

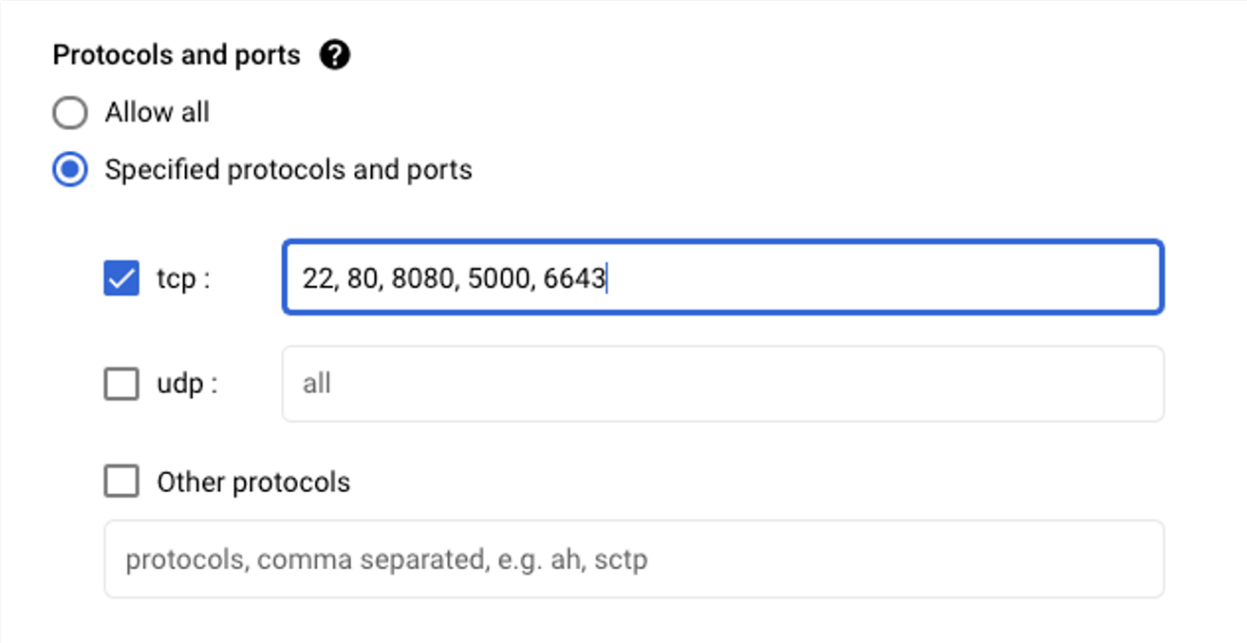
- Keep default settings for Management, security, disks, networking, sole tenancy
- Click Create.

6.4 Create VM instances for each of the node in the topology setup

Repeat the above Section 6.3 to create all other VM nodes.

Warning

Make sure to select the correct Region of the Edge core nodes, i.e., Edge 1 on US East, and Edge 2 on US West. Zone can be selected from any of the Zone choices. Also make sure the CPU/Mem configurations are according to the VM instances requirements.



6.5 Once all the VM instances have been created and are up running, please note down the following information:

- Public IP of the Cloud core node
- Public IPs of all the Edgecore nodes

Then we are going to install KubeEdge on all the nodes and setup the KubeEdge cluster.

7 Node pre-installation

This section guides you to perform pre-installation setup on all the VM instances nodes, including the Cloud core node and all the Edge core nodes. Opening up a separate terminal for each of the node to perform the following setups.

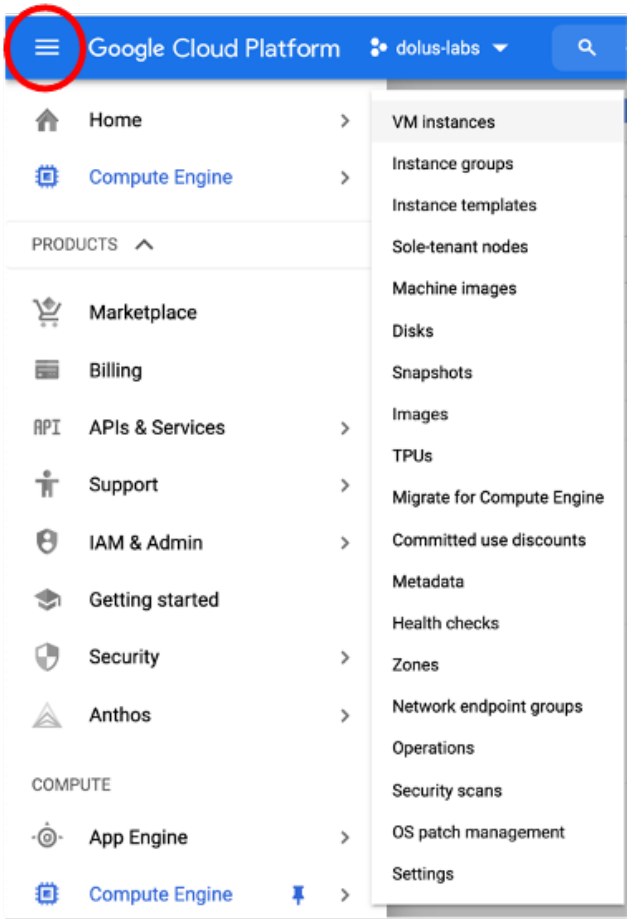
7.1 SSH login to the nodes using your SSH private key.

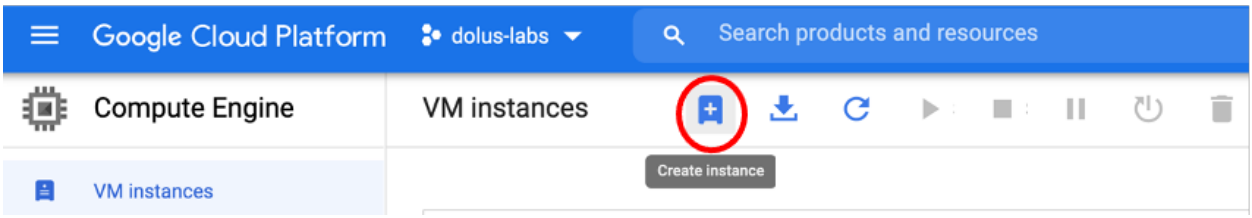
7.2 Switch to sudo user

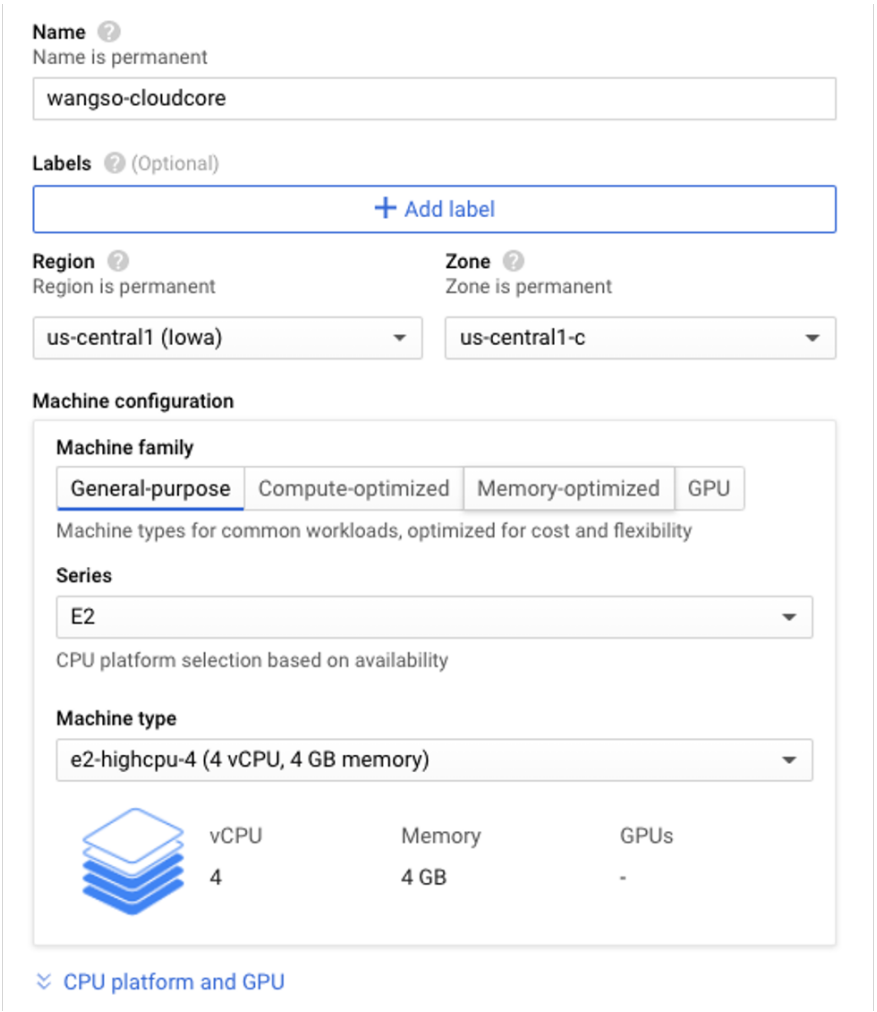
```
$ sudo su
```

7.3 Clone KubeEdge installation git repo

```
$ cd /root
$ git clone https://github.com/wangso/KubeEdge-install.git
```

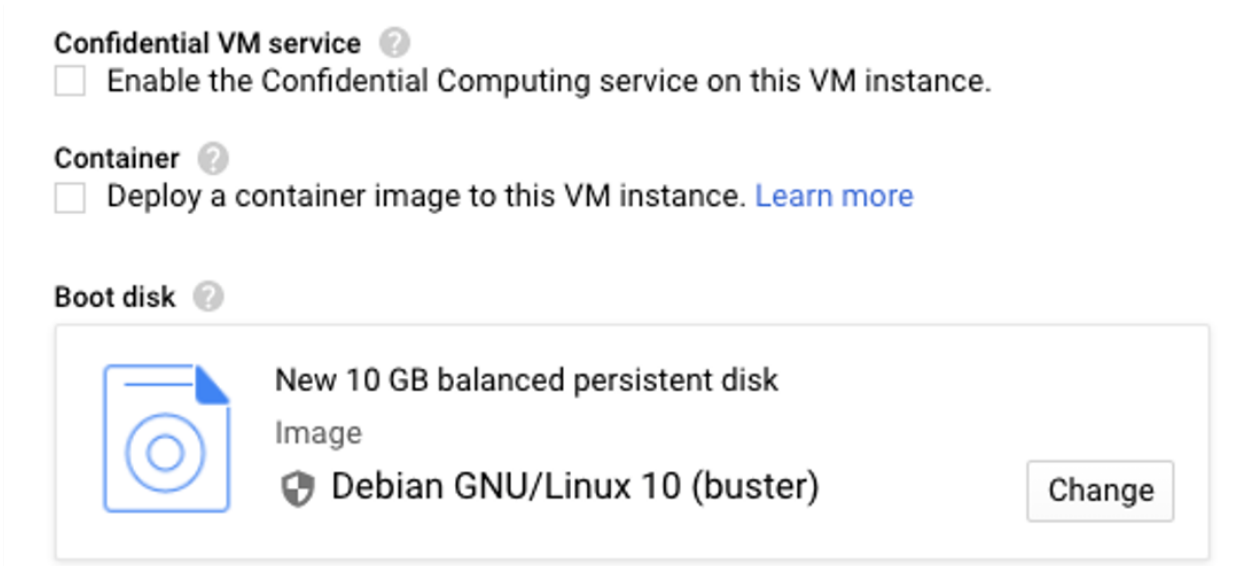

Google Cloud Platform
dolus-labs
Home
Compute Engine
PRODUCTS
Marketplace
Billing
APIs & Services
Support
IAM & Admin
Getting started
Security
Anthos
COMPUTE
App Engine
Compute Engine
VM instances
Instance groups
Instance templates
Sole-tenant nodes
Machine images
Disks
Snapshots
Images
TPUs
Migrate for Compute Engine
Committed use discounts
Metadata
Health checks
Zones
Network endpoint groups
Operations
Security scans
OS patch management
Settings

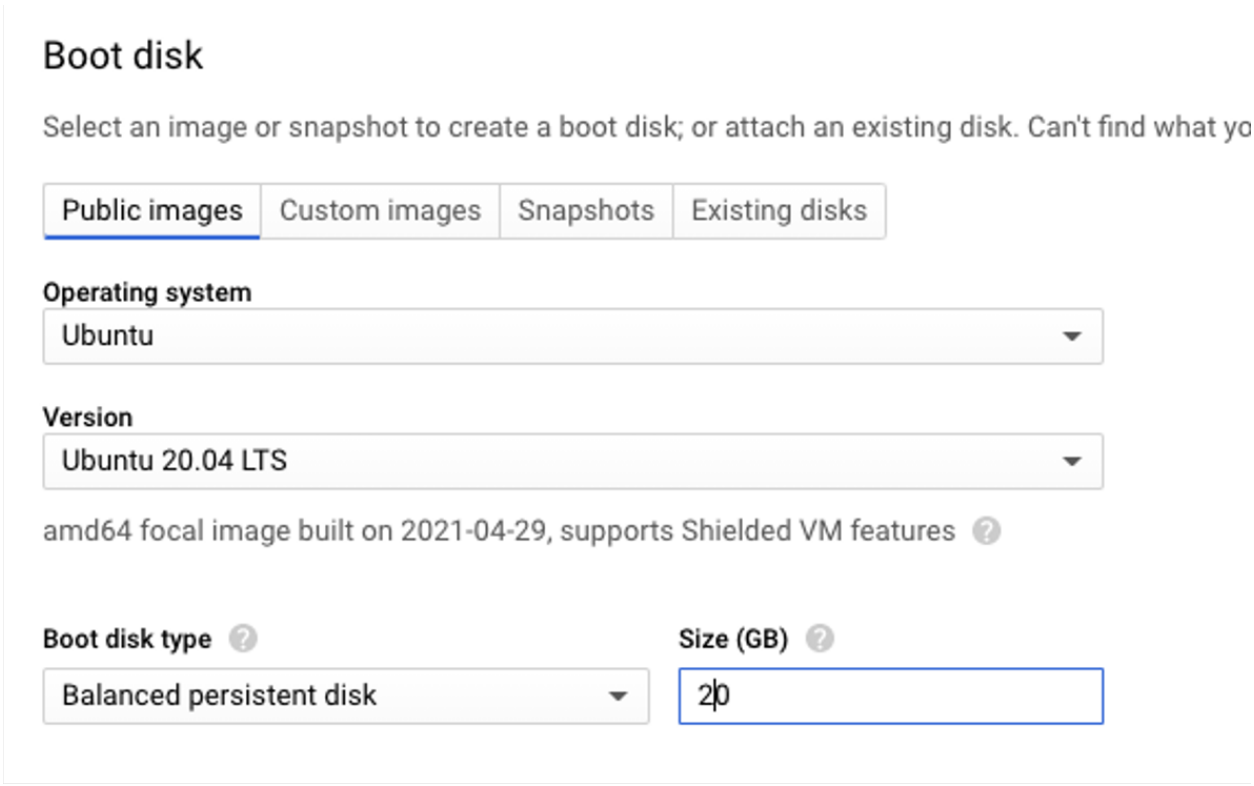




7.4 Setup Root SSH login with password

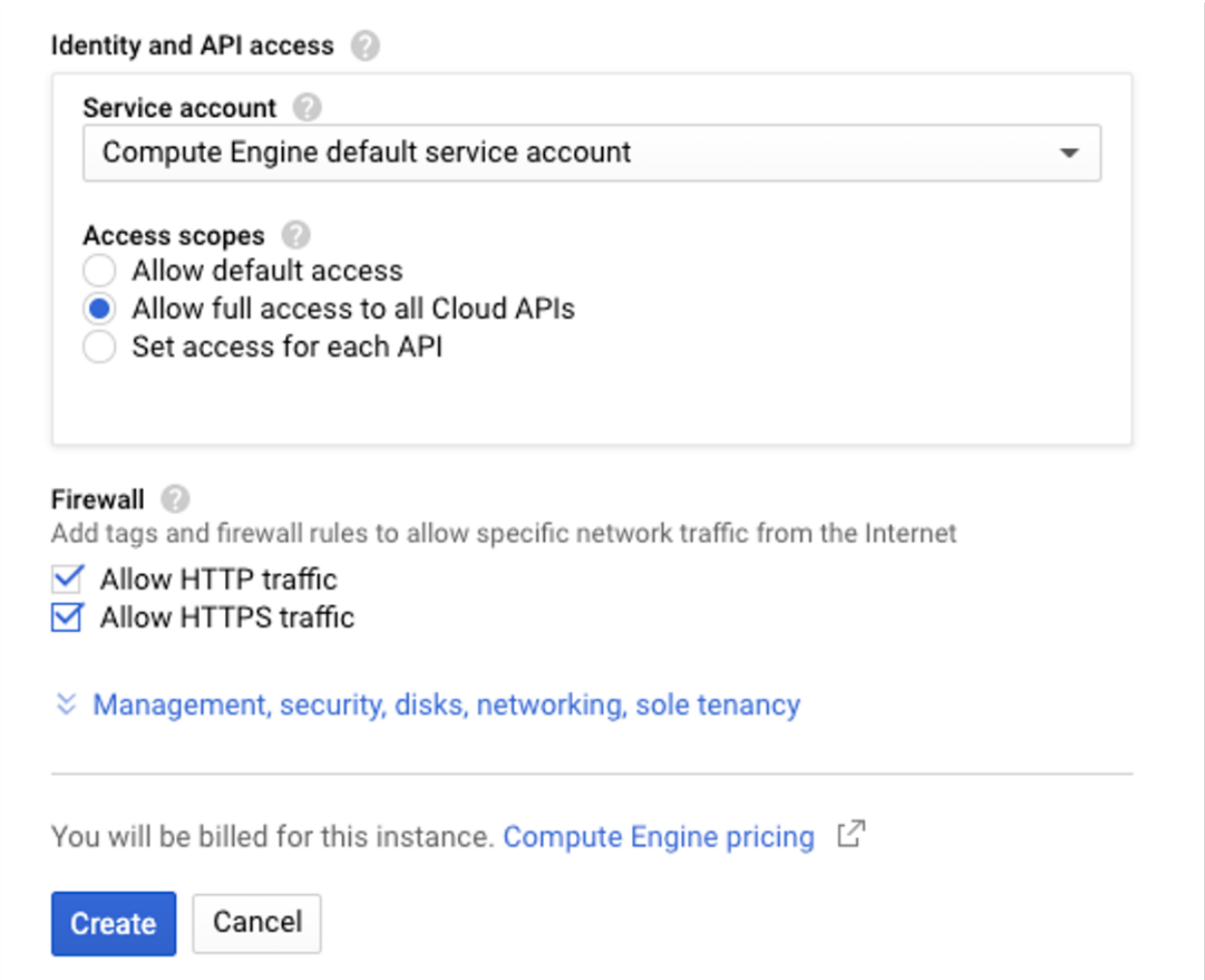
```
$ sed -i "s/#PermitRootLogin prohibit-password/PermitRootLogin yes/g"
  /etc/ssh/sshd_config
$ sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g"
  /etc/ssh/sshd_config
$ systemctl restart sshd
```





7.5 Setup root user password

```
$ passwd
```



And create your root password by following screen prompts.

8 KubeEdge Installation on All Nodes

8.1 Cloud core node installation

On your Cloud core node, run the following commands to install KubeEdge.

```
$ cd /
$ bash /KubeEdge-install/kubeedge-cloud-install/cloud-core-install.sh
  "Cloud_core_Public_IP/"
```

Example:

```
$ bash /KubeEdge-install/kubeedge-cloud-install/cloud-core-install.sh
  34.121.207.23
```

8.2 Edge core node installation

On each of your Edge core nodes, run the following commands to install KubeEdge.

```
$ cd /
$ bash /KubeEdge-install/kubeedge-edge-install/edge-core-install.sh
```

After the installation, you will have created a KubeEge cluster, with the Cloud core running on the same VM instance. Run the following commands to check for your installation:

```
$ kubectl get nodes
```

You will see the following output:

KubeEdge-lab/img/kubeedge-master.png

The entire cluster is controlled by a Docker container running on the Cloud core node named ”control-plane”. Run the following command to check for the Docker container:

```
$ docker ps -a
```

8.3 Setup remote login from Cloud node to Edge nodes, and copy certificates to Edge nodes

For the Cloud core to communicate with the Edge core nodes, you need to setup remote login from the Cloud node to any Edge core node. For each of your Edge core node, run the following commands:

```
root@trust:/home/wangso# docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                    NAMES
8104a88248fb        kindest/node:v1.17.2   "/usr/local/bin/entr…"   3 days ago          Up 3 days           0.0.0.0:5000->5000/tcp, 1
27.0.0.1:6443->6443/tcp   kind-control-plane
root@trust:/home/wangso#
```

```
$ cd /
$ bash /KubeEdge-install/kubeedge-cloud-install/add-edge-core.sh
  "Edge-core-node-Public-IP"
```

Example: $ bash /KubeEdge-install/kubeedge-cloud-install/add-edge-core.sh 34.70.155.106

8.4 Obtain KubeEdge secret token

The secret token is used for Edge core nodes to join to the Cloud core node as worker nodes. Run the following command to obtain the secret token and paste to the commands on the Edge nodes later.

```
$ bash /KubeEdge-install/kubeedge-cloud-install/obtain-token.sh
```

KubeEdge-lab/img/kubeedge-token.png

8.5 Join Edge core nodes to the KubeEdge cluster

On each of the Edge core nodes, run the following command to join the node to the KubeEdge cluster running on the Cloud core node VM, using the Cloud core node IP address, and the token obtained from the previous step, and a unique node_name you want to use in the cluster.

Warning

Make sure you use a distinct "node_name" for each of the Edge core nodes, otherwise you will have conflicts between nodes and wont be able to see all the Edge nodes in the cluster. In this lab, we will use the names defined in the lab topology, i.e., Edge1-small, Edge1-medium, Edge1-large, Edge2-small, Edge2-medium, Edge2-large, for the Edge core nodes.

```
$ bash /KubeEdge-install/kubeedge-cloud-install/edge-core-join.sh
  "cloud_node_IP" "the_token" "node_name"
```

Example:
$ bash /root/KubeEdge-rpi-dev/Chameleon-version/kubeedge-edge-install/edge-core-join.sh 34.121.207.23 3b64449c7f0c3d289a392361c11b344503510fa3cd2801537432db66f5bd143d.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MTc4MDI3Mjd9.TBqjhdTlhqHzX3biQkQ42D18oubKfwJYnFYvIjenjlU Edge1-small

9 Deploy an Volunteer Edge Computing with Trusted Bioinformatics application, the OnTimeURB, in the KubeEge cluster

9.1 OnTimeURB Overview

OnTimeURB software enables researchers in better utilizing compute resources available with them at local servers as well resources allocations on public cloud platforms. It gives researchers flexibiltiy in creating hybrid cyber infrastruture for their applications requirements. The software have the capapbility of deducing in real time the ideal compute resources allocations for application execution and stages the tasks on those resources.

9.2 Github repository for the OnTimeURB application

We have created github repository for the application, which includes various bioinformatics workflow that can be run inside Docker containers. The link to the github repo is: *https://github.com/wangso/OnTimeURB.git*

From your Cloud core node terminal, run the following commands:

```
$ cd /
$ git clone https://github.com/wangso/OnTimeURB.git
```

9.3 Example workflow - Pgen workflow

PGen workflow (*https://github.com/wangso/OnTimeURB/tree/master/Docker_Containers/Bioinformatics_Workflows/PGen*) is used to efficiently facilitate large-scale next generation sequencing (NGS) data analysis to identify genomic variations. The workflow allows users to identify single nucleotide polymorphisms (SNPs) and insertion-deletions (Indels), perform SNP annotation and conduct copy number variation (CNV)

analyses on multiple resequencing datasets. PGen workflow is developed using many widely accepted open-source NGS tools for alignment of reads, variants calling, variants filtration, VCF merging and others. The workflow starts by aligning either paired-end or single-end FASTQ reads against the organisms reference genome using BWA. Picard Tools is also used at this step to locate duplicate molecules and assign all reads into groups with the default parameters. After alignment, SNPs and Indels are called using the Haplotype Caller algorithm from Genome Analysis Toolkit (GATK). Filtering criteria are defined in INFO filed in vcf file, where QD stands for quality by depth, FS is Fisher strand values and MQ is mapping quality of variants. Detected variants were then filtered using criteria QD $<$ 26.0 || FS $>$ 60.0 || MQ $<$ 40.0 for SNPs and DQ $<$ 26.0 || FS $>$ 200.0 || MQ $<$ 40.0 for indels. Custom criteria can also be applied by the user. Outputs are generated as BAM and VCF standard formats. In the filtering step, only the SNPs and Indels are considered; however other types of variants can extracted from the generated unfiltered files.

We have pre-built the Docker container image for the PGen workflow, which can be accessed at *https://hub.docker.com/r/apfd6/pgen_wf* and downloaded using the below command:

```
$ docker pull apfd6/pgen_wf:latest
```

In this lab, we will configure KubeEdge Pod to automatically download the Docker image to the specified Edge core node that we will run the PGen workflow on. The downloading process will take several minutes when its the first time the Pod runs the PGen workflow. After the first running, the Docker image will be cached on the same node, and thus it will take much shorter time to initialize a pod to run the same container.

9.4 Labeling Edge core nodes

Our volunteer edge computing with Trusted bioinformatics application requires automatic deployment of workflows on a user-defined Edge core node. For this purpose, we need to give a label to each of the Edge core nodes in the cluster, so we can select on which node to run workflows based on the labels. The node labels will be selected using KubeEdge cluster configuration files.

To label the Cloud core node, run:

```
$ kubectl label nodes kind-control-plane nodePool=master
```

To label the Edge core nodes, for each of the Edge core node, run the following. Replace Node-Name with the correct node name you used when joining the Edge nodes to the cluster. Replace NodePool-Name with the name you want to use in the cluster configuration. The Node-Name and NodePool-Name can be the same to avoid confusion.

```
$ kubectl label nodes Node-Name nodePool=NodePool-Name
```

Example: *$ kubectl label nodes Edge1-small nodePool=Edge1-small*

Depending on where you want to have Kubernetes pod deployed, the configuration will look like below in your Kubenetes configuration file:

```
nodeSelector:
  nodePool: Edge1-small
```

9.5 Create deployments for example workflow

From your Cloud core node terminal,

10 Homework Problems

1) Describe pod, deployment, service, and what they are used for.
2) Describe the different between ClusterIP, NodePort and Load Balancing, and what they are best suited for.
3) Describe the different between port, target port and node port, and what they are best used for.
4) Describe the difference between Ingress and port-forwarding, and what they are best used for.

11 Lab Report

1) Submit the required screen shots in the lab steps
2) Short assay answers to the homework problems