**Lab: KubeEdge - A Kubernetes Native Edge Computing Framework**

In this lab, we will learn KubeEdge. **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. Both edge and cloud parts are now opensourced.

The advantages of Kubeedge include mainly:

* **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.

## Components

KubeEdge is composed of these components:

* [**Edged**](https://kubeedge-docs.readthedocs.io/en/latest/modules/edge/edged.html)**:** an agent that runs on edge nodes and manages containerized applications.
* [**EdgeHub**](https://kubeedge-docs.readthedocs.io/en/latest/modules/edge/edgehub.html)**:** 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**](https://kubeedge-docs.readthedocs.io/en/latest/modules/cloud/cloudhub.html)**:** A web socket server responsible for watching changes at the cloud side, caching and sending messages to EdgeHub.
* [**EdgeController**](https://kubeedge-docs.readthedocs.io/en/latest/modules/cloud/controller.html)**:** an extended kubernetes controller which manages edge nodes and pods metadata so that the data can be targeted to a specific edge node.
* [**EventBus**](https://kubeedge-docs.readthedocs.io/en/latest/modules/edge/eventbus.html)**:** an MQTT client to interact with MQTT servers (mosquitto), offering publish and subscribe capabilities to other components.
* [**DeviceTwin**](https://kubeedge-docs.readthedocs.io/en/latest/modules/edge/devicetwin.html)**:** responsible for storing device status and syncing device status to the cloud. It also provides query interfaces for applications.
* [**MetaManager**](https://kubeedge-docs.readthedocs.io/en/latest/modules/edge/metamanager.html)**:** the message processor between edged and edgehub. It is also responsible for storing/retrieving metadata to/from a lightweight database (SQLite).

## Architecture

Diagram

Description automatically generated

**References:**

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/docs/setup/keadm/>

<https://github.com/kubeedge/kubeedge/blob/master/docs/setup/keadm.md>

KubeEdge documentation and installation:

<https://kubeedge-docs.readthedocs.io/en/latest/setup/kubeedge_install_keadm.html>

**My successful installation using the blog on github**

**(**[**https://github.com/JingruiLea/blogs/blob/master/%E5%AE%89%E8%A3%85kubeedge.md**](https://github.com/JingruiLea/blogs/blob/master/%E5%AE%89%E8%A3%85kubeedge.md)**)**

**1. Create Nodes**

1. On GCP, create two VM instances (ips are: 10.128.0.71 and 10.128.0.72)
2. Both are Ubuntu 20.0 LTS, 2 vCPU, 4GB mem
3. **Base Image install**

**Note:** should install on both VM machines. Otherwise, we can also install on one machine, save as machine image, and then spin up two additional VMs using the image to serve as cloud and edge.

1. On both VMs, do

$ sudo apt-get -y update

$ sudo apt-get -y upgrade

$ sudo apt-get -y install wget net-tools gcc make vim openssh-server docker.io

1. Check docker installation

$ sudo docker --version

1. Setup Root SSH

$ sudo nano /etc/ssh/sshd\_config

Change “PermitRootLogin without-password” to “PermitRootLogin yes”

Change “PasswordAuthentication no” to “PasswordAuthentication yes”

Save, and restart SSHD

$ sudo systemctl restart sshd

1. Setup root password

$ sudo passwd

1. Install snap and Kubernetes suite

$ sudo apt-get -y install snap

$ sudo apt-get -y install snapd

$ sudo snap install kubectl --classic

$ sudo snap install kubelet --classic **(DO NOT install this on edge node)**

$ sudo snap install kubeadm --classic

1. Check Kubernetes installation (this step will show error messages since we do not have a cluster yet)

$ kubectl version

1. Install golang:

$ sudo su

$ wget <https://golang.org/dl/go1.15.7.linux-amd64.tar.gz>

$ tar -C /usr/ -xzf /root/go1.15.7.linux-amd64.tar.gz

$ nano /etc/bash.bashrc

Append the following at the end

**Setting 1: (used to install libraries and go on both nodes, then keep setting 1 on cloud node, and switch to setting 2 on RPi after)**

export PATH=$PATH:/snap/bin:/usr/go/bin

export GOPATH=/usr/go

export GOBIN=$GOPATH/bin

export PATH=$PATH:$GOBIN:$GOROOT/bin

export GO111MODULE=auto

**Setting 2:**

export PATH=$PATH:/snap/bin:$GOROOT/bin:$GOPATH/bin

export GOPATH=/root/go

export GOROOT=/usr/go

export GOBIN=/usr/go/bin

export GO111MODULE=auto

$ source /etc/bash.bashrc

1. Download Kubeedge git repo

$ git clone <https://github.com/kubeedge/kubeedge> $GOPATH/src/github.com/kubeedge/kubeedge

1. **Cloud node preparation**
2. Install Keadm using the downloaded source file

$ cd $GOPATH/src/github.com/kubeedge/kubeedge

$ make all WHAT=keadm

1. **Copy Keadm binary to path**

$ cp ./\_output/local/bin/keadm /usr/bin/

1. Setup SSH login from cloud node to edge node

$ ssh-keygen

$ ssh-copy-id root@173.22.148.144

$ ~~scp ./\_output/local/bin/keadm~~ [~~root@173.22.148.144:/usr/bin/~~](mailto:root@173.22.148.144:/usr/bin/) ~~(Don’t run this if the edge core is Raspberry Pi since it has a different architecture, and the binary copied from the core will not work on it)~~

Note: the path to keadm may be different, we can copy /usr/bin/keadm instead

1. Install go kind

$ cd /root/

$ GO111MODULE="on" go get sigs.k8s.io/kind@v0.7.0

$ kind version

1. Download kindest

$ docker pull kindest/node:v1.17.2

1. Configure kindest

$ sudo tee /root/kind.yaml <<-'EOF'

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

networking:

apiServerAddress: "127.0.0.1"

apiServerPort: 6443

nodes:

- role: control-plane

image: kindest/node:v1.17.2

extraPortMappings:

- containerPort: 5000

hostPort: 5000

EOF

1. Create KubeEdge cluster using kind

$ kind create cluster --config=/root/kind.yaml

1. If you have restarted the Cloud node, need to restart the control plane docker container:

$ docker restart kind-control-plane

1. Check nodes (it might take a minute or so for it to be ready)

$ kubectl get nodes

Text

Description automatically generated with low confidence

1. Create Kubeedge cloud node **(this step needs to re-run every time the cluster is restarted, and also advertise address is required if joining edge node with public IP!!!)**

$ keadm init --advertise-address=129.114.25.26 --kubeedge-version=1.6.0 --kube-config=/root/.kube/config

1. Check node log

$ tail -f /var/log/kubeedge/cloudcore.log

1. Create certificates and keys, and copy to edge node

$ cd $GOPATH/src/github.com/kubeedge/kubeedge/build/tools

$ ./certgen.sh genCertAndKey edge

$ scp -r /etc/kubeedge/certs root@173.22.148.144:/etc/kubeedge/

$ scp -r /etc/kubeedge/ca root@173.22.148.144:/etc/kubeedge/

1. **Edge node preparation**
2. Get Kubeedge secret token (on Cloud node)

$ kubectl get secret -nkubeedge tokensecret -o=jsonpath='{.data.tokendata}' | base64 -d

1. Join Kubeedge cluster

$ sudo su

On Edge1:

$ keadm join --cloudcore-ipport=129.114.25.26:10000 --token=5c1160b875dd1c91e55870b46c93cc2286b9f203d3ae748ec7f50ba272a99ad4.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MTczMTM0NDF9.\_SOJoPNXuwyE1FCBCCErntIKZnWrx6dHG9LifaeOdFg --edgenode-name=edge1 --kubeedge-version=1.6.0

On Edge2:

$ keadm join --cloudcore-ipport=129.114.25.26:10000 --token=5c1160b875dd1c91e55870b46c93cc2286b9f203d3ae748ec7f50ba272a99ad4.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MTczMTM0NDF9.\_SOJoPNXuwyE1FCBCCErntIKZnWrx6dHG9LifaeOdFg --edgenode-name=edge2 --kubeedge-version=1.6.0

1. Check log

$ journalctl -u edgecore.service -b

1. **Deploy kubeedge-web-app on cloud node (root user)**
2. Check for edge node in the cluster

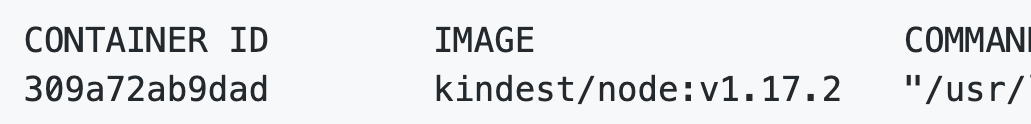
$ kubectl get nodes

**Text

Description automatically generated**

1. Check docker container

$ docker ps



1. Download example web-app git repo

$ git clone https://github.com/kubeedge/examples $GOPATH/src/github.com/kubeedge/examples

1. Compile example web app

$ cd $GOPATH/src/github.com/kubeedge/examples/web-demo/kubeedge-web-app/

$ go mod vendor

$ make all

$ docker pull centos:7.6.1810

$ docker build . -t kubeedge/kubeedge-web-app:v2.6

1. Update web-demo code

$ cd $GOPATH/src/github.com/kubeedge/examples/web-demo/kubeedge-web-app/

$ nano views/content.html

* Modify the list of songs and their names in the html file

$ nano Dockerfile

|  |
| --- |
| # Based on centos  FROM kubeedge/kubeedge-web-app:v2.6  LABEL maintainers="KubeEdge Authors"  LABEL description="KubeEdge Web App"  # Copy from build directory  COPY kubeedge-web-app /kubeedge-web-app  COPY static /static  COPY views /views  # Update  RUN yum -y update  # Define default command  ENTRYPOINT ["/kubeedge-web-app"]  # Run the executable  CMD ["kubeedge-web-app"] |

$ nano utils/kubeclient.go

|  |
| --- |
| package utils  import (  "errors"  "fmt"  "io/ioutil"  "k8s.io/client-go/rest"  certutil "k8s.io/client-go/util/cert"  "net"  "os"  )  var KubeQPS = float32(5.000000)  var KubeBurst = 10  var KubeContentType = "application/vnd.kubernetes.protobuf"  var ErrNotInCluster = errors.New("unable to load in-cluster configuration, KUBERNETES\_SERVICE\_HOST and KUBERNETES\_SERVICE\_PORT must be defined")  func InClusterConfig() (\*rest.Config, error) {  const (  tokenFile = "/var/run/secrets/kubernetes.io/serviceaccount/token"  rootCAFile = "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt"  )  host, port := os.Getenv("KUBERNETES\_SERVICE\_HOST"), os.Getenv("KUBERNETES\_SERVICE\_PORT")  if len(host) == 0 || len(port) == 0 {  return nil, ErrNotInCluster  }  token, err := ioutil.ReadFile(tokenFile)  if err != nil {  return nil, err  }  tlsClientConfig := rest.TLSClientConfig{}  if \_, err := certutil.NewPool(rootCAFile); err != nil {  fmt.Errorf("Expected to load root CA config from %s, but got err: %v", rootCAFile, err)  } else {  tlsClientConfig.CAFile = rootCAFile  }  return &rest.Config{  // TODO: switch to using cluster DNS.  Host: "https://" + net.JoinHostPort(host, port),  TLSClientConfig: tlsClientConfig,  BearerToken: string(token),  }, nil  }  // KubeConfig from flags  func KubeConfig() (conf \*rest.Config, err error) {  kubeConfig, err := InClusterConfig()  if err != nil {  return nil, err  }  kubeConfig.QPS = KubeQPS  kubeConfig.Burst = KubeBurst  kubeConfig.ContentType = KubeContentType  return kubeConfig, err  } |

$ nano deployments/kubeedge-speaker-instance.yaml

|  |
| --- |
| apiVersion: devices.kubeedge.io/v1alpha2  kind: Device  metadata:  name: speaker-01  labels:  description: 'Speaker'  manufacturer: 'test'  spec:  deviceModelRef:  name: speaker-model  nodeSelector:  nodeSelectorTerms:  - matchExpressions:  - key: ''  operator: In  values:  - test1 |

$ nano deployments/kubeedge-web-app.yaml

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  labels:  k8s-app: kubeedge-web-app  name: kubeedge-web-app  namespace: default  spec:  selector:  matchLabels:  k8s-app: kubeedge-web-app  template:  metadata:  labels:  k8s-app: kubeedge-web-app  spec:  nodeSelector:  node-role.kubernetes.io/master: ""  hostNetwork: true  containers:  - name: kubeedge-web-app  image: kubeedge/kubeedge-web-app:v2.7  imagePullPolicy: IfNotPresent  restartPolicy: Always |

1. Create docker image

$ ­­­­cd $GOPATH/src/github.com/kubeedge/examples/web-demo/kubeedge-web-app/

$ make clean && make all

$ docker build . -t kubeedge/kubeedge-web-app:v2.7

$ docker images

Text

Description automatically generated

1. Change cluster rbac privilege:

$ sudo tee /root/fabric8-rbac.yaml <<-'EOF'

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: fabric8-rbac

subjects:

- kind: ServiceAccount

# Reference to upper's `metadata.name`

name: default

# Reference to upper's `metadata.namespace`

namespace: default

roleRef:

kind: ClusterRole

name: cluster-admin

apiGroup: rbac.authorization.k8s.io

EOF

1. Deploy rbac

$ kubectl create -f /root/fabric8-rbac.yaml

1. Load docker image to kind

$ kind load docker-image kubeedge/kubeedge-web-app:v2.7

1. ~~Create device models to be used for deployments:~~

~~$ cd /etc/kubeedge/crds/devices~~

~~$ kubectl create -f devices\_v1alpha2\_devicemodel.yaml~~

~~$ kubectl create -f devices\_v1alpha2\_device.yaml~~

1. Create deployments

$ cd $GOPATH/src/github.com/kubeedge/examples/web-demo/kubeedge-web-app/deployments

$ kubectl create -f kubeedge-speaker-model.yaml

$ kubectl create -f kubeedge-speaker-instance.yaml

$ kubectl create -f kubeedge-web-app.yaml

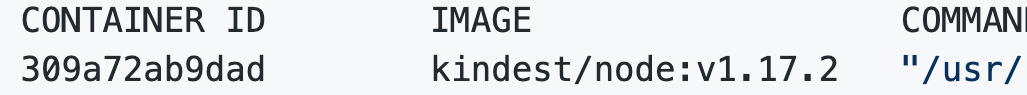
1. Check deployments

$ kubectl get nodes

Text

Description automatically generated with medium confidence

1. Check docker container

****$ docker ps

1. Check crds

$ kubectl get crds

**­­­** **Graphical user interface, text

Description automatically generated**

1. Check deployments

$ kubectl get deployments

1. Check pods

$ kubectl get pods

Take down pod ID

1. Check IP address of web app

$ kubectl describe pod “your-pod-ID” | grep IP

Take down IP address of web app on cloud node

1. SSH tunneling to connect to the web page on the cloud node from your local computer (if desktop is not available on the cloud node)

**Note:** change the local IP with your local IP you got from step 16); and the public IP of your cloud node. Change 8888 to a different number if your port on your local computer is currently used.

**(if the below command does notwork, you need to copy and paste the public key to the authorized\_keys file in the /root/.ssh/ directory of the host machine**)

$ ssh -i cyberrange-gce-key -N -f -L localhost:8888:172.17.0.2:80 [root@](mailto:root@130.211.223.135)35.184.151.234

**Note:** for this command to work, need to copy the public key of cyberrange and paste into authorized\_key in the /root/.ssh dir on the host machine

1. View the web app page:

In your web browser, put localhost:8888 and enter

**Graphical user interface, text, application

Description automatically generated**

1. **Install mqtt on edge node to listen to commands from cloud node**
2. On edge node, create mqtt client

$ sudo su

$ cd /root/

$ touch /root/main.go

|  |
| --- |
| package main  import (  "encoding/json"  "fmt"  "github.com/kubeedge/kubeedge/cloud/pkg/devicecontroller/types"  "github.com/yosssi/gmq/mqtt"  "github.com/yosssi/gmq/mqtt/client"  )  func main() {  fmt.Println("Get music list successfully")  cli := client.New(&client.Options{  // Define the processing of the error handler.  ErrorHandler: func(err error) {  fmt.Println(err)  },  })  fmt.Println("Create mqtt client successfully")  stopchan := make(chan int)  // Terminate the Client.  defer cli.Terminate()  // Connect to the MQTT Server.  err := cli.Connect(&client.ConnectOptions{  Network: "tcp",  Address: "localhost:1883",  ClientID: []byte("receive-client"),  })  if err != nil {  panic(err)  }  fmt.Println("Connect mqtt client successfully")  err = cli.Subscribe(&client.SubscribeOptions{  SubReqs: []\*client.SubReq{  {  TopicFilter: []byte(`$hw/events/device/speaker-01/twin/update/document`),  QoS: mqtt.QoS0,  // Define the processing of the message handler.  Handler: func(topicName, message []byte) {  Update := &types.DeviceTwinDocument{}  err := json.Unmarshal(message, Update)  if err != nil {  fmt.Println("Unmarshal error", err)  fmt.Printf("Unmarshal error: %v\n", err)  }  fmt.Printf("%+v", Update)  },  },  },  })  fmt.Println("Subscribe mqtt topic successfully")  <-stopchan  if err != nil {  panic(err)  } else {  fmt.Println("Connection successfully")  }  } |

1. Run mqtt

$ go get github.com/yosssi/gmq/mqtt/client

$ go get github.com/yosssi/gmq/mqtt

$ go run /root/main.go

**Text

Description automatically generated**

1. **Dummy test of cloud-edge connection**
2. On cloud node, check kubeedge log

$ kubectl logs -f “Your-pod-ID”

1. On your local computer, keep the web page running
2. On edge node, run mqtt client code

$ go run main.go

1. Open up another edge node terminal, and run check kubeedge log (this may not work)

$ tail -f /var/log/kubeedge/edgecore.log

1. From the web page, click any song to play, or stop playing, etc. On cloud node, your should see output that shows your actions:

Text

Description automatically generated

1. On edge node, you should see output from mqtt client:

Text

Description automatically generated with low confidence

1. **Web-demo real deployment**
2. Install mplayer on RPi

$ apt-get -y install mplayer

1. Clone example applications:

$ cd

$ git clone <https://github.com/kubeedge/examples>

1. Modify code to use mplayer instead of omxplayer:

$ cd examples/web-demo/pi-player-app

$ nano main.go

|  |
| --- |
| package main  import (  "encoding/json"  "fmt"  "math/rand"  "os"  "os/exec"  "reflect"  "strings"  "github.com/yosssi/gmq/mqtt"  "github.com/yosssi/gmq/mqtt/client"  "github.com/kubeedge/kubeedge/cloud/pkg/devicecontroller/types"  )  var musicDir = "/home/pi/music/"  func main() {  // Get files  files, err := os.Open(musicDir)  if err != nil {  panic(err)  }  var m = make(map[string]string)  list, \_ := files.Readdirnames(0)  fmt.Printf("Files read dirnames result: %v\n", list)  for \_, track := range list {  trackWithoutSuffix := strings.TrimSuffix(track, ".mp3")  fmt.Printf("Loading track key: %s value: %s\n", trackWithoutSuffix, track)  m[trackWithoutSuffix] = track  }  fmt.Println("Get music list successfully")  cli := client.New(&client.Options{  // Define the processing of the error handler.  ErrorHandler: func(err error) {  fmt.Println(err)  },  })  fmt.Println("Create mqtt client successfully")  stopchan := make(chan int)  // Terminate the Client.  defer cli.Terminate()  // Connect to the MQTT Server.  err = cli.Connect(&client.ConnectOptions{  Network: "tcp",  Address: "localhost:1883",  ClientID: []byte("receive-client"),  })  if err != nil {  panic(err)  }  fmt.Println("Connect mqtt client successfully")  err = cli.Subscribe(&client.SubscribeOptions{  SubReqs: []\*client.SubReq{  {  TopicFilter: []byte(`$hw/events/device/speaker-01/twin/update/document`),  QoS: mqtt.QoS0,  // Define the processing of the message handler.  Handler: func(topicName, message []byte) {  Update := &types.DeviceTwinDocument{}  err := json.Unmarshal(message, Update)  if err != nil {  fmt.Println("Unmarshal error", err)  fmt.Printf("Unmarshal error: %v\n", err)  }  cmd := exec.Command("pkill", "-9", "mplayer")  cmd.Run()  trackToPlay := \*Update.Twin["track"].CurrentState.Expected.Value  fmt.Printf("Receive expected track: %s\n", trackToPlay)  // Stop music  if trackToPlay == "stop" {  return  }  \_, ok := m[trackToPlay]  if !ok {  fmt.Printf("Could not find song %s in playlist\n", trackToPlay)  trackToPlay = MapRandomKeyGet(m).(string)  fmt.Printf("Selected random track %s to play\n", trackToPlay)  }  fmt.Printf("Playing track: %s\n", musicDir+trackToPlay+".mp3")  cmd = exec.Command("mplayer", musicDir+trackToPlay+".mp3")  err = cmd.Run()  if err != nil {  fmt.Printf("Error while playing track: %v\n", err)  }  },  },  },  })  fmt.Println("Subscribe mqtt topic successfully")  <-stopchan  if err != nil {  panic(err)  } else {  fmt.Println("Connection successfully")  }  }  func MapRandomKeyGet(mapI interface{}) interface{} {  keys := reflect.ValueOf(mapI).MapKeys()  return keys[rand.Intn(len(keys))].Interface()  } |

1. Complile binary for pi-player-app:

$ go build -o pi-player-app main.go

1. Run binary

$ /root/examples/web-demo/pi-player-app/pi-player-app

1. **Try out music playing on the web UI, it should play and stop music on the edge node:**

**Text

Description automatically generated**

**Congratulations on a successful web-demo installation on KubeEdge!**

**Now we are ready to move on to the ai example application:**

**KubeEdge – ai application setup**

1. On Cloud core, follow the instruction doc
2. Activate camera at Raspberry Pi edge:
3. Install motion following this page:

<https://hackernoon.com/spy-your-pet-with-a-raspberry-pi-camera-server-e71bb74f79ea>

1. Enable authentication following the same page, but may need to also follow this page:

<https://www.raspberrypi.org/forums/viewtopic.php?t=68038>

1. Now the Raspberry Pi camera can stream video
2. Install edge core on the Pi:
3. Snap install kubectl wont work and it says it does not support
4. To install Ubuntu 20.04 on Raspberry pi, follow this page:

<https://ubuntu.com/download/raspberry-pi>

1. To setup wifi connection, follow this page:

<https://raspberrypi.stackexchange.com/questions/111722/rpi-4-running-ubuntu-server-20-04-cant-connect-to-wifi>

1. Setup root login, add password, and install all edge core libraries – follow the KE install GCP document.
2. Install opencv on RPi
3. <https://linuxize.com/post/how-to-install-opencv-on-ubuntu-20-04/>
4. Install picamera streaming
5. This was difficult: <https://raspberrypi.stackexchange.com/questions/111852/ubuntu-19-10-enabling-and-using-raspberry-pi-camera-module-v2-1>

From Cloud Core:

1. Install ai example application:

* Face-recog Dockerfile has issue: looks to me the architecture should be the cloud core (amd64) instead of using RPi’s architecture (arm64)

|  |
| --- |
| FROM sixsq/opencv-python:master-amd64  WORKDIR /code  COPY . /face-recongition  WORKDIR /face-recongition  RUN pip install –upgrade pip  RUN pip --default-timeout=10000 install -r requirements.txt -i https://mirrors.aliyun.com/pypi/simple  RUN pip install -U scikit-learn  ENTRYPOINT ["python3","-u","/face-recongition/face\_recong.py"] |

* To run the devicemodel, need to go to /etc/kubeedge/crds/devices and run

Kubectl apply -f devices\_v1alpha2\_device.yaml

And Kubectl apply -f devices\_v1alpha2\_devicemodel.yaml

To get the crd definition ready,

Then go to /usr/go/src/github.com/kubeedge/examples/ ai app /crds/ and run the two devices yaml

For cloud node to find the edge node as a worker, need to do the following on the cloud node:

“$ kubectl label nodes test1 nodePool=cluster”

And then in the faceRecogDeployment.yaml, add

nodeSelector:

nodePool: cluster

If need to schedule pod on the master node:

“$ kubectl label nodes kind-control-plane nodePool=master”

To build and run arm based docker images on x86 machines:

<https://matchboxdorry.gitbooks.io/matchboxblog/content/blogs/build_and_run_arm_images.html>

**To mimic arm64 from x86 machine:** [**https://www.stereolabs.com/docs/docker/building-arm-container-on-x86/**](https://www.stereolabs.com/docs/docker/building-arm-container-on-x86/)