**l.ntuKubernetes Thesis Notes**

* Install Raspberry Pi 4 operation system. Install Ubuntu 22.04 LTS.
  1. Install Raspberry Imager to mac computer

Download raspberry Pi Imager

Double click imager and move it to Applications Folder

Run Raspberry Pi Imager from Application Folder

Select Raspberry Pi model

Select OS : other options

Select SDCARD

Edit Settings and configure:

Hostname: rasppi

Username:dimitra

Password:dimitra

ssh enable

WiFi settings and choose a wifi to connect to computer and raspberry. If you select a certain wifi then it won’t connect when it will be far. Choose wisely.

Accept settings and install Ubuntu 22.04 LTS to SD Card

* 1. As long as installation completed remove microSD, put it on raspberry pi and switch on by connecting to power
  2. Find the raspberry pi IP address to connect to wifi. Therefore, install Angry IP Scanner
  3. Connect to home wifi : cosmote-185564 with that password
* Connect with ssh from MacBook Air to Raspberry Pi
  1. Trying to connect with ssh with usually command line
     + ssh [dimitra@192.168.1.3](mailto:dimitra@192.168.1.3) or dimitra@rasppi and cannot connect due to error port error connection 22
  2. use command
     + ssh rasppi
     + ssh 192.168.1.3
     + login and ask for username / password : dimitra/dimitra
* Install dependencies for application

First Things First

* + Sudo apt update
  + Sudo apt upgrade
  + Sudo apt autoremove
  + Sudo reboot

Install Apache

* + Sudo apt install apache2 -y
  + Sudo usermod -a -G www-data pi
  + Sudo chown -R -f www-data:www-data /var/www/html

Install Php 8.2

Because of ‘package not found’, install from third-party repository to install php

Install Java JDK in Raspberry Pi 4

* + Sudo apt update
  + Sudo apt upgrade
  + sudo apt install curl gnupg ca-certificates
  + curl -s https://repos.azul.com/azul-repo.key | sudo gpg --dearmor -o /usr/share/keyrings/azul.gpg
  + echo "deb [arch=arm64 signed-by=/usr/share/keyrings/azul.gpg] https://repos.azul.com/zulu/deb stable main" | sudo tee /etc/apt/sources.list.d/zulu.list
  + sudo apt update
  + sudo apt install zulu21-jdk-headless

install MongoDB4.2

* + sudo apt update
  + sudo apt upgrade
  + curl -s https://www.mongodb.org/static/pgp/server-4.2.asc | sudo tee /usr/share/keyrings/mongodb-archive-keyring.asc >/dev/null
  + echo "deb [signed-by=/usr/share/keyrings/mongodb-archive-keyring.asc arch=arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
  + sudo apt update

because of a gpg error did the following:

curl -fsSL https://pgp.mongodb.com/server-6.0.asc | \

sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/mongodb-server-6.0.gpg

echo "deb [ arch=amd64,arm64 signed=/etc/apt/trusted.gpg.d/keyrings/mongodb-server-6.0.gpg ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

and then I did:

* + sudo apt install mongodb-org
  + sudo systemctl enable mongod

*Do not run apt install mongodb on your Raspberry Pi, or indeed any Linux computer! The versions of MongoDB shipped with Linux distributions are very out of date. They won't run as well, and some of them are so old they're no longer supported.*

*All of these are problem installing mongodb on Raspberry Pi 4. Nevertheless, there is an installation procedure from mongo.org site with details installing mongo, but also explaining that it is wiser to use Mongo Atlas rather than installing MongoDB on raspberry pi 4. Although it has a good processor it can not administrating and taking all to mongo. So, the site says is better for using MongoDB Atlas.*

…..

Install docker

* + sudo apt update
  + sudo apt upgrade
  + sudo curl -sSL https://get.docker.com | sh
  + sudo usermod -aG docker $USER
  + logout to check that docker added to groups
  + login again and type groups
  + test with the command: docker run hello-world

Install K3s on raspberry pi as a worker node for raspberry meaning connect to cluster with certain ip and master node.

This means that Kubernetes must installed running as master node to an external other system/physical computer

This can be done by installing a VM with linux for installing kubernetes and docker in that VM. To this VM must be connected the raspberry Pi k3s as a worker node. So the VM sees the node (raspberry pi) and in this node run containers from existing applications.

Unistall docker, k3s-agent

Make static IP in raspberry. Create an /etc/dhcpd.conf file and write the following. Reboot and check file with static IP and is working.

cat /etc/dhcpd.conf

interface wlan0

static\_routers=192.168.1.1

static domain\_name\_servers=127.0.0.53

static ip\_address=192.168.1.6/24

**Create a VM ubuntu22.04 LTS to install Kubernetes to integrate with a K3s on Raspberry Pi 4**

Multipass installation: brew install –cask multipass

multipass launch 22.04 --disk 30G --memory 3G --cpus 2 --name kubernetes --network name=en0,mode=manual,mac="52:54:00:4b:ab:cd"

multipass shell kubernetes (in to the VM ubunut)

ip address: get the IP of the instance 192.168.64.3

and the mac address: 52:54:00:4b:ab:cd

give a static IP address to instance

multipass exec -n kubernetes -- sudo bash -c 'cat << EOF > /etc/netplan/10-custom.yaml

network:

version: 2

ethernets:

extra0:

dhcp4: no

match:

macaddress: "52:54:00:4b:ab:cd"

addresses: [192.168.64.101/24]

EOF'

multipass exec -n kubernetes -- sudo netplan apply

multipass info kubernetes | grep IPv4 -A1

this shows the original IP of the instance 192.168.64.3 and the static IP given 192.168.64.101

***after configuring the VM, must get into the VM to configure to install Kubernetes.***

*~~Multipass shell kubernetes : getting into the vm~~*

*~~Declare the hostnames and Ips to see the Kubernetes and is the last setting for vm to continue with installing Kubernetes~~*

*~~Sudo vi /etc/hosts~~*

*~~192.168.64.101 kubernetes kubernetes~~*

*~~192.168.1.3 rasppi rasppi~~*

*~~cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf~~*

*~~overlay~~*

*~~br\_netfilter~~*

*~~EOF~~*

*~~sudo modprobe overlay~~*

*~~sudo modprobe br\_netfilter~~*

*~~# sysctl params required by setup, params persist across reboots~~*

*~~cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf~~*

*~~net.bridge.bridge-nf-call-iptables = 1~~*

*~~net.bridge.bridge-nf-call-ip6tables = 1~~*

*~~net.ipv4.ip\_forward = 1~~*

*~~EOF~~*

*~~# Apply sysctl params without reboot~~*

*~~sudo sysctl --system~~*

*~~# Verify that the br\_netfilter, overlay modules are loaded by running the following commands:~~*

*~~lsmod | grep br\_netfilter~~*

*~~lsmod | grep overlay~~*

*~~#Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-nf-call-ip6tables, and net.ipv4.ip\_forward system variables are set to 1 in your sysctl config by running the following command:~~*

*~~sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip\_forward~~*

*~~because of problem with connecting with github from vm I installed ssh to VM, so to send files to vm through scp~~*

*~~sudo apt update~~*

*~~sudo apt upgrade~~*

*~~sudo apt-get install openssh-client~~*

*~~to connect to VM via ssh I created a public key ---- ssh-keygen -o --- and copy that to authorized-keys file in VM~~*

*~~then I did an scp to copy containerd tar file to install containerd~~*

*~~curl -LO https://github.com/containerd/containerd/releases/download/v1.7.14/containerd-1.7.14-linux-arm64.tar.gz~~*

*~~sudo tar Cxzvf /usr/local containerd-1.7.14-linux-arm64.tar.gz~~*

*~~curl -LO https://raw.githubusercontent.com/containerd/containerd/main/containerd.service~~*

*~~sudo mkdir -p /usr/local/lib/systemd/system/~~*

*~~sudo mv containerd.service /usr/local/lib/systemd/system/~~*

*~~sudo mkdir -p /etc/containerd/~~*

*~~sudo containerd config default | sudo tee /etc/containerd/config.toml > /dev/null~~*

*~~sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml~~*

*~~sudo systemctl daemon-reload~~*

*~~sudo systemctl enable --now containerd~~*

*~~#Check that containerd service is up and running~~*

*~~systemctl status containerd~~*

*~~install runc~~*

*~~curl -LO https://github.com/opencontainers/runc/releases/download/v1.1.12/runc.arm64~~*

*~~sudo install -m 755 runc.arm64 /usr/local/sbin/runc~~*

*~~install cni~~*

*~~curl -LO https://github.com/containernetworking/plugins/releases/download/v1.4.1/cni-plugins-linux-arm64-v1.4.1.tgz~~*

*~~sudo mkdir -p /opt/cni/bin~~*

*~~sudo tar Cxzvf /opt/cni/bin cni-plugins-linux-arm64-v1.4.1.tgz~~*

*~~install kubelet,kubeadm,kubectl~~*

*~~sudo apt-get update~~*

*~~sudo apt-get install -y apt-transport-https ca-certificates curl gpg~~*

*~~curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg~~*

*~~echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list~~*

*~~sudo apt-get update~~*

*~~sudo apt-get install -y kubelet kubeadm kubectl~~*

*~~sudo apt-mark hold kubelet kubeadm kubectl~~*

*~~configure crictl to work with containerd~~*

*~~sudo crictl config runtime-endpoint unix:///var/run/containerd/containerd.sock~~*

*~~in VM Kubernetes run the master node so everything is configured only in the Kubernetes vm~~*

*~~sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=192.168.64.101~~*

*~~the following is the command for connecting nodes. This will be executed in every worker node to connect to cluster~~*

***~~kubeadm join 192.168.64.101:6443 --token p2quhp.7xms1ubai740joiy --discovery-token-ca-cert-hash sha256:d2a6482057a84fe7226c6497565cc63d329419e5b4858b0ef587ecd0c09e0cd8~~***

*~~To make kubectl to work for non-root user~~*

*~~mkdir -p $HOME/.kube~~*

*~~sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config~~*

*~~sudo chown $(id -u):$(id -g) $HOME/.kube/config~~*

*~~install a network pod to connect each pod when running. This Kubernetes installation uses calico~~*

*~~kubectl create -f~~* [*~~https://raw.githubusercontent.com/projectcalico/calico/v3.28.1/manifests/tigera-operator.yaml~~*](https://raw.githubusercontent.com/projectcalico/calico/v3.28.1/manifests/tigera-operator.yaml)

**Installing Kubernetes on VM machine ubuntu 22.04**

multipass launch 22.04 --disk 30G --memory 3G --cpus 2 --name kubemaster --network name=en0,mode=manual,mac="52:54:00:4b:ab:cd"

multipass exec -n kubemaster -- sudo bash -c 'cat << EOF > /etc/netplan/10-custom.yaml

network:

version: 2

ethernets:

extra0:

dhcp4: no

match:

macaddress: "52:54:00:4b:ab:cd"

addresses: [192.168.64.101/24]

EOF'

multipass exec -n kubemaster -- sudo netplan apply

multipass info kubemaster | grep IPv4 -A1

this shows the original IP of the instance 192.168.64.4 and the static IP given 192.168.64.101

sudo apt update

sudo apt install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

sudo apt upgrade

sudo apt install docker-ce -y

sudo systemctl status docker

sudo usermod -aG docker ${USER}

install cri-dockerd **which is a requirement for a container runtime to work with Kubernetes**

VER=$(curl -s https://api.github.com/repos/Mirantis/cri-dockerd/releases/latest|grep tag\_name | cut -d '"' -f 4|sed 's/v//g')

wget https://github.com/Mirantis/cri-dockerd/releases/download/v${VER}/cri-dockerd-${VER}.amd64.tgz

tar xvf cri-dockerd-${VER}.amd64.tgz

sudo mv cri-dockerd/cri-dockerd /usr/local/bin/

wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packaging/systemd/cri-docker.service

wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packaging/systemd/cri-docker.socket

sudo mv cri-docker.socket cri-docker.service /etc/systemd/system/

sudo sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service

sudo systemctl daemon-reload

sudo systemctl enable cri-docker.service

sudo systemctl enable --now cri-docker.socket

stop installing K8s

**rasppi Raspberry Pi 4 – installing k3s agent to connect to k8s**

curl -sfL https://get.k3s.io | K3S\_URL=https://192.168.64.101:6443 K3S\_TOKEN=p2quhp.7xms1ubai740joiy sh –

Continuing with notes for installing Kubernetes.

Searching shown that K8s can’t work with K3s. This means that I can’t have a K8s cluster integrate a worker node running K3s. There isn’t compatibility in join command for worker node.

So either install K8s in VM and raspberry, or install K3s in VM and raspberry.

Choose to install K3s. Why K3s and not K3d, or KubeEdge.

K3d, Minikube are for testing environments. I want production environments.

Now KubeEdge can integrate with K8s. So, that is an option which we have to discuss with the professor.

Problem with subnet. VM and raspberry are in different subnets. This is a problem. Must connect them to a same network.

**Create VM**

multipass launch 22.04 --disk 30G --memory 3G --cpus 2 --name kubemaster --network name=en0,mode=manual,mac="52:54:00:4b:ab:cd"

now must create a bridge to connect to that the vm

according to some sites I tried to create a bridge, to make network adjustments to connect VM with raspberry. Finally, I managed to do that, but without knowing which actions worked.

After that, having VM and rasp in the same subnet I installed K3s and K3s-agent on rasp and create a cluster.

First I create a pod running in rasp using NodePort and it did and I saw that running in raspberry.

Then following Nana’s instructions and create a cluster where a web application is running and a MongoDB.

**Create Java web app for test use**

Want to run this application in raspberry, but initially would like to run it in VM as Kubernetes deployment.

So make a web app, create an image of it, this image must be

installed in a container – pod and deployed by Kubernetes.

When a pod is destroyed or deleted a new one is created by Kubernetes.

So, in order to delete a pod permanently and its replicas you must delete the deployment.

Kubectl get deployment

Kubectl delete deployment <name-of-deployment>

Create a pod – container in master node and running in rasp. I tried to create container in rasp but didn’t make it. Nevertheless, expose the apache pod and see apache on the browser.

A simple web app as a container running in Kubernetes

A diagram of a software system

Description automatically generated

To create an image of the product after creating the exe file in Dockerfile: is

Docker build . -t kube-app

Kube-app is the name of the image

So, after that we can run a container:

Docker run kube-app

Ok, but have to expose the port

Docker run -p 3030:8080 kube-app

Everything is running on the foreground and we can stop it with Ctrl+C

To run as daemon is

Docker run -d -p 3030:8080 kube-app I have a container and run in the background and everything ok

Tp create an image and push it to docker hub:

Docker tag kube-app:latest dimidk/testkube:latest

Docker push dimidk/testkube:latest

And now I want to pull that image and create a pod in Kubernetes

*Just for checking:*

*Must restart kubemaster vm and raspberry and see if there would be new IP assignment on VM and on raspberry.*

* **Comments on 23/9/2024**