ZSUT

Zakład Sieci i Usług Teleinformatycznych

Kubernetes laboratory

Lab part 1: Installing k3s on Raspberry Pi cluster

Prepared by: dr inż. Dariusz Bursztynowski

ZSUT. Zakład Sieci i Usług Teleinformatycznych
Instytut Telekomunikacji
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Last update March 2023

Table of contents

1. Introduction

We are going to install k3s on our Raspberry Pi cluster. K3s is a lightweight Kubernetes distribution from Rancher especially suitable for computing ARM platforms. The installation procedure is based on both bash scripts and Ansible playbooks. This mix of techniques is adopted to compare purely imperative management automation, here represented by the scripts, to more declarative automation tool being Ansible in our case. A side effect of this approach for those who do not now Ansible is to learn basics of this tool.

The overall procedure consists of four main steps:

- preparing the management host that will be used for launching major installation procedures
- installing the OS (Ubuntu in our case) on our Raspberry Pi-s
- installing k3s on RbPi-s – this step will include also additional tasks (in bash) to fine tune the configuration of Raspberry Pi-s
- post-install configuration of the management host to enable *kubectl* access to the cluster.

A complete HW setup of our lab consists of the cluster itself (four RbPi-s nodes connected to a PoE switch) and a WRT54GL router, as shown in Figure 1 (the names of cluster nodes in the figure are exemplary).

Figure 1. Physical setup of the lab.

It is assumed that the cluster switch WAN port will be directly connected to WRT54GL to separate the cluster subnetwork from your local network and provide flexibility in reserving/configuring IP addresses in the cluster. WAN port of WRT54GL will be connected to your local network and can receive IP address form its DHCP server. Your laptop with the management host (see next section) should be connected to the cluster network segment (to a LAN port of WRT54GL). WRT54GL should be configured with DHCP server enabled and the CIDR different from that of your local network.

The separation using WRT54GL is optional and you can connect directly to your local network if you prefer. However, the inclusion of WRT54GL, apart from allowing to avoid any changes to your local network, makes it possible to work in Internet-disconnected mode provided that the images of needed containers had earlier been stored locally in your cluster. The latter can be achieved by deploying appropriate Pods once in Internet-connected mode using the option `imagePullPolicy: IfNotPresent` or `imagePullPolicy: Always` (default) and then using the option `imagePullPolicy: IfNotPresent` while in Internet-disconnected mode.

The configuration/installation steps of the cluster are described in the following sections.

2. Preparing the management host

The management host is the machine where you will run scripts and Ansible playbooks to install k3s on the cluster and issue kubectl commands to control the k3s cluster. This setup is symbolically shown in Figure 2.

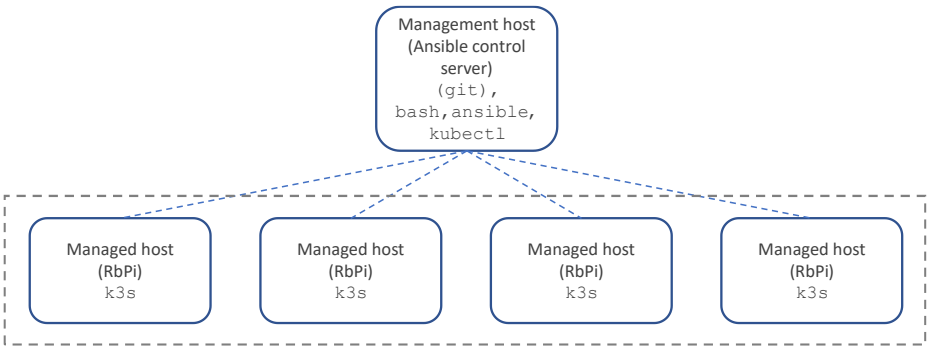

Figure 2. Management node (host) and managed nodes (hosts).

We assume the management host runs under Linux. It can be bare metal or virtual machine. The commands included in this guideline apply to a Debian distribution (we used Ubuntu). In case of using other distribution of Linux appropriate adaptation of the commands may be needed.

- Install `net-tools` and `nmap` utility on the management host.

  ```
  sudo apt-get update
  sudo apt-get install net-tools && sudo apt-get install nmap
  ```

- Install Ansible on the management host (using Ansible terminology, our management host is *Ansible control node*). All installations should be done according to Ansible guidelines. If you happen to follow instructions where configurations of Ansible managed hosts (i.e., hosts to be automatically configured using Ansible) are considered you can skip these parts for the moment unless you already have some test hosts running. Example Ansible installation guides are as follows (but you may want to use other guides – feel free to use a guide of your choice):

  https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-and-upgrading-ansible

  https://learnubuntu.com/install-ansible-in-ubuntu/

  https://www.cyberciti.biz/faq/how-to-install-and-configure-latest-version-of-ansible-on-ubuntu-linux/

  https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-20-04

- Install `kubectl` command line tool on the management node using a method of your choice:

  https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

  https://pwittrock.github.io/docs/tasks/tools/install-kubectl/#install-with-snap-on-ubuntu

  Important: the version of kubectl client has to match the version of Kubernetes control plane according to the rule specified under the first link above. We recommend to install v1.25 client of `kubectl` on the management node, and v1.25 control plane (i.e., k3s v.25 in our case) on the cluster. Although higher version of Kubernetes is available, we recommend 1.25 because it complies with the compatibility matrix for kube-prometheus stack for monitoring we will use in our third lab (cf. https://github.com/prometheus-operator/kube-prometheus#compatibility).

- Upgrade kubectl (for any case)

  https://gist.github.com/qaiserali/18926b5bd9ca7a0551195d449bf31eb6

http://www.mtitek.com/tutorials/kubernetes/install_kubectl.php
https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

3. Preparing Raspberry Pi hosts and local DHCP

3.1. Preparing RbPi hosts

Our cluster contains four Raspberry Pi 4B boards and a local switch which serves as a "TOR" switch and a PoE (Power over Ethernet) power source for the RbPi-s. There are two RbPi boards with 4GB RAM (1, 2 - counting from left to right) and two RbPi-s with 8GB RAM (3, 4 – counting form left to right). As we will see later, RbPi #1 will be used as a master (control) node of the cluster – 4GB RAM is sufficient for this purpose while the nodes with more RAM can be dedicated for the workloads.

- Install Ubuntu on RbPi-s
  - the recommended way is using *Raspberry Pi Imager* application (google to find it)
  - in Raspberry Pi Imager, select: *Other general-purpose OS -> Ubuntu -> Ubuntu 20.04 64bit server ARM*

    Note: you can select Ubuntu 22.04 LTS server, 64bit ARM BUT according to this https://github.com/k3s-io/k3s/issues/5443 Ubuntu 22.04 / 21.10 on Raspberry Pi needs to install EXTRA KERNEL MODULES (over 100MB of additional code), e.g., using the command before starting k3s:

    ```
    # sudo apt install linux-modules-extra-raspi
    ```

    Otherwise k3s will keep restarting. So, **we recommend installing Ubuntu 20.04 LTS**.

  - before starting burning the image set appropriate options (see also Figure 3):
    - *set host name*: Linux host name to be used (also, to be used in ansible hosts.yaml file); we recommend to preserve a common prefix for the node name and add a variable suffix (e.g., 1, 2, …)
      - otherwise you will have to change node names manually by loging *via* ssh to each host to set its name:

        ```
        # ssh <user-name>@<pi-ip-addess>
        # hostnamectl set-hostname <hostname>
        ```

        (for the first time, the passwd will as you set it in *RbPi Imager*, otherwise it will be *ubuntu* and you will be prompted to change it during the first loging)
    - *Enable ssh -> Use password authentication* (checked); this will be used, e.g., by the configuration bash script to upload ssh credentials to the cluster nodes
    - *Set user name and password*: to be used by a bash script to preconfigure the node (you can leave the default values pi/ubuntu – up to your choice)
    - Check locale-settings (time zone, keyboard)

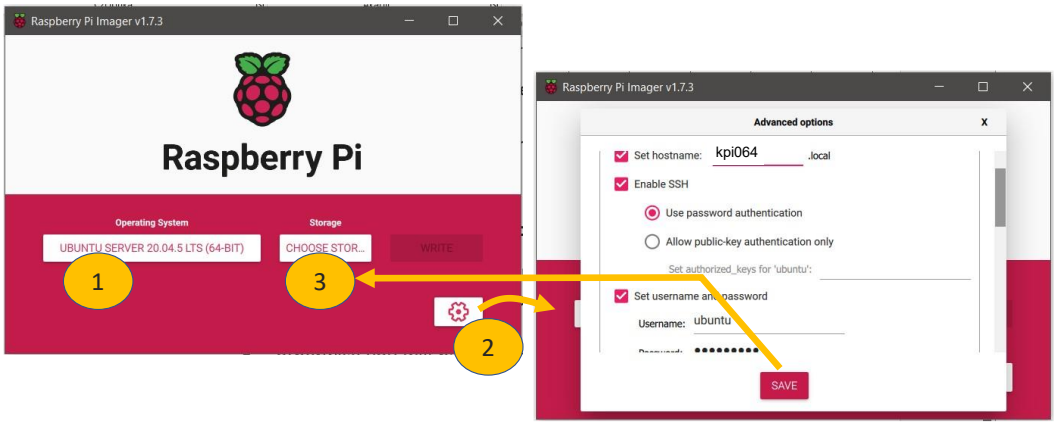

Figure 3. Raspberry Pi Imager settings.

Note: In case of reinstalling the cluster (burning the images again, etc.) and when you want to reuse old host names/addresses then you need to remove those old names/addresses from the `known_hosts` file on your management host as follows (below, kpi091, … are host names from an exemplary installation):

`# ssh-keygen -R <hostname> (or ssh-keygen -R <IP-address>)` – for each host

or in a script form:

```
#!/bin/bash
ssh-keygen -R kpi091
ssh-keygen -R kpi092
ssh-keygen -R kpi093
ssh-keygen -R kpi094
```

Moreover, it is also recommended to delete (by editing) those hosts from file ~/.ssh/config otherwise hosts can occur multiple times in ~/.ssh/config (this is not critical but multiple occurrences can be confusing).

3.2. Configuring local DHCP for using the cluster

Actually, two configurations should be done:

- Reserving fixed IP addresses for the RbPi hosts to be used in the cluster.
- Reserving a suitable address range for dynamic use in the cluster (needed for Kubernetes Services of type LoadBalancer).

Respective steps are described in the remainder of this section.

Note1: Do not connect the cluster to WRT54GL with DHCP enabled before starting the following procedure (this may save your time waiting for the expiration of DHCP lease time or renewing address allocation for the RbPi boards).

Note2: before starting the procedure make sure your WRT54GL is connected to your local network and: 1) has its WAN port (in panel Basic/Network) set to Type=DHCP and DNS server=auto, and 2) has its IP address (panel Basic/Network/LAN) assigned form a DIFFERENT subnet than the outer (external, e.g., your home) subnetwork (the one from which WAN port gets its address).

Note3: Important to note is that cluster configuration scripts we are going to use will order cluster nodes according to the value of their IP address – the higher the address (value) the higher index the node will receive. In particular, the master node (say, node1) of the cluster gets the lowest IP address from among our RbPi-s (say, 192.168.1.80). Successive nodes will receive addresses in increasing order (say, node 2 -> 192.168.1.81, etc.). ***Therefore, by assigning the IP addresses in the DHCP as described in the following we determine the "ordering" (and the role) of the RbPi-s boxes in our cluster.*** *Actually, the names of RbPi-s assigned in the image burning step will not matter for the final naming (**Linux hostname**) of the nodes. Keep it in mind while assigning addresses to your RbPi-s.*

The nodes of the cluster (RbPi nodes) have to be assigned fixed IP addresses. This can be achieved in various ways. To limit the number of configurations on cluster nodes themselves, below we assume the DHCP server in the local network

can be accessed and we can assign static IP addresses using DHCP. An example procedure to achieve that is described in the following.

1. Make sure the cluster switch is unplugged from the power source (all RbPi-s are switched-off) and all RbPi-s are correctly Ether-connected to the cluster switch.
2. Log to WRT54GL and set a short address lease time (e.g. 5 minutes) in the Basic/Network/LAN section.
3. Configure dynamic address range in the WRT54GL DHCP server (Basic/Network/LAN) so that a suitable subset of addresses is reserved for static and dynamic use in your cluster. Static addresses from this pool will be applied to the cluster nodes (four hosts) while the remaining ones will be available for allocation to the cluster's load balancer which in turn will assign them to the services deployed in the cluster and exposed to the outside as type External IP. For the latter purpose (load balancer), a couple of addresses (say, 6) will be perfectly sufficient in our case. **Accordingly, reserving 10 addresses in total for the cluster will suffice**.
4. Switch on the cluster (power on the cluster switch). The management host should be connected as in Figure 1.
5. Make sure no other Raspberry Pi devices except your cluster's RbPi-s are active in the cluster (i.e. WRT54GL) network segment (not critical but makes it easier to discover your RbPi-s in the next step). Run the following command from the management host terminal to verify the IP and MAC addresses of all RbPi-s in your cluster (adjust the CIDR/mask to your environment):

   ```
   $ sudo nmap -sP 192.168.1.0/24 | awk
   '/^Nmap/{ipaddress=$NF}/28:CD:C1|B8:27:EB|DC:A6:32|E4:5F:01/{print ipaddress, $3}' |
   tr -d "()"
   ```

6. Execute the following steps **for each** of the RbPi-s listed (use the credentials you set while burning the images):
   - `ssh <your_user>@<pi_ip_address>`
   - annotate the name of the host (the RbPi to which you have logged in) visible in the terminal
   - in the Basic/DHCP reservation panel of WRT54GL set appropriate static IP reservation for the given RbPi (insert its MAC and the desired IP address). A good practice is to keep some order between RbPi-s' IP addresses and a correlation between the address and the name of a given RbPi; save the setting (bottom of the panel)
   - shut down current RbPi from the terminal: `$ sudo poweroff`.
7. Once step 6 has been completed for all RbPi-s switch-off the cluster switch (unplug it from the power source) and plug it on again. The nodes of the cluster should now boot and receive their static IP addresses that will be used throughout the rest of our experiments. In particular, they will be used in the automatic configuration procedures in the next section.
8. To be sure everything is fine check the connected devices in the DHCP server (panel Status/Device list on WRT54GL) to verify that new IP addresses have been assigned as expected. Now you can make corrections if something went wrong.
9. If everything is as expected set the preferred value of the address lease time in the server (typically 1440 minutes).

Note: when you ssh to a host for the first time then it's data (e.g., name or IP address) is being registered in the file `known_hosts` on the management host. If for any reason you want to reuse this name/IP address for another (remote) host (e.g., after reinstalling the OS on the remote host) than you should **remove this (old) host name/addresses from the `known_hosts` file**. To this end you can execute the command `ssh-keygen -R <hostname>` (or `ssh-keygen -R <IP-address>`) from the management node.

4. Installing k3s and configuring kubectl client

An intended side-effect of running the installation procedure is to get acquainted with modern **network automation tools** using (as an example of such a tool) Ansible. To make the learning process more complete, a (small) part of the configuration work is done using traditional bash scripting. The latter will allow to compare imperative form of bash scripts to much more declarative form of Ansible templates. In our case, the bash script starts with a set of initial

(imperative) configurations applied to the cluster nodes (like installing the authorization keys on the nodes) while at the end it invokes Ansible and passes it a playbook being responsible for the actual deployment of k3s on the nodes.

One of the tasks to be done by students is to analyse bash and Ansible files involved and pay attention to the "style" of defining automation tasks and recognise the differences between imperative and (more) declarative configuration specification using bash and Ansible, respectively. Notice that bash and Ansible files contain inline comments that explain the role of respective parts of the specification – use them to comprehend the overall workflow of operations.

4.1. Installing k3s

The first step consists in adjusting the installation script and selected Ansible templates to match your environment. Respective adjustments are as follows:

Bash script settings

- **File** …**`/pi-cluster-install/install.sh`**

  This script sets the names of RbPi boxes. These names will later be assigned to the boxes as their Liux *hostname*. Moreover, authorization keys are uploaded to the RbPi boxes and local ssh files (known_hosts, config) on the management host are updated with the information on the RbPi-s. Finally, this script invokes ansible playbook responsible for the actual installation of k3s in the cluster nodes.

  The convention used for assigning the name to our RbPi-s is that the name consists of a fixed (common across the cluster) prefix string and a variable suffix being an integer from the sequence 1, 2, …, with the master node indexed with 1 and being the one with the lowest IP address in our cluster. The RbPi box with the next lower IP address will become a node with index 2, and so on.

  Below, we present the parameter section of the script. Commented parameters with symbol ← should be adjusted manually according to your setup while uncommented ones should be left unchanged. Read also the comments at the beginning of the file.

```
NETWORK="$1"                                cluster CIDR/mask, script parameter
USER_NAME="ubuntu"                          ← your user name on all RbPi-s
PASSWORD="raspberry"                        ← your password on all RbPi-s
HOST_FILE="./cluster"                       auxiliary file for IPs addresses of hosts
INVENTORY_FILE="inventory/hosts.ini"        Ansible inventory file
CONFIG_FILE="$HOME/.ssh/config"             ssh config file (to ssh to the RbPi-s)
ERROR_FILE="/tmp/ssh-copy_error.txt"        error log file
PUBLIC_KEY_FILE="$HOME/.ssh/db_id_rsa"      ← your ssh public key
MASTER_GROUP="master"                       Ansible group of nodes (one node for us)
MASTER_NODE="kpi061"                        ← the name of the control (master) node
WORKER_GROUP="node"                         Ansible group of nodes serving as worker
WORKER_NODE="kpi06"                         ← the prefix of the name of worker node
CLUSTER_GROUP="cluster"                     Ansible group of all cluster hosts
```

  End of this file: we recommend to update the name of the config file downloaded form the master node of the cluster to preserve its uniqueness. To this end, set it, e.g., using the date suffix as follows:

```
scp $USER_NAME@$MASTER_NODE:~/.kube/config ~/.kube/config-cluster-20230225
#export KUBECONFIG=~/.kube/config-cluster-20230225 ← this line can be left commented
```

  The data from file `~/.kube/config-cluster-2023022` shall be used to configure local (on the management host) *kubconfig* file for a remote use of the *kubectl* tool from the management host.

Ansible templates settings

- **File:** …**`/pi-cluster-install/inventory/group_vars/all.yaml`**

Settings

`k3s_version: v1.25.5+k3s2` ← recommended

`ansible_user: ubuntu` ← adjust to your user name (the one for RbPi hosts)

- **File:** …**`/pi-cluster-install/inventory/group_vars/all.yaml`**

  Settings

  `k3s_version: v1.25.5+k3s2` ← recommended[1)] [1](install kubectl client v.25 on the management host)

  `ansible_user: ubuntu` ← adjust to your user name (the one for RbPi hosts)

- **File:** …**`/pi-cluster-install/inventory/hosts.ini`**

  Settings

  In our case, this inventory file is created by the bash script from scratch so there is no need to touch it manually. It is however recommended to analyse and understand its contents after being created.

Notice: Selected hints regarding Ansible (can be helpful when analysing our Ansible playbooks):

- *Ansible facts (or playbook variables):*
  https://www.middlewareinventory.com/blog/ansible-facts-list-how-to-use-ansible-facts/
  https://www.digitalocean.com/community/tutorials/how-to-access-system-information-facts-in-ansible-playbooks
- *Ansible special variables:*
  https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html
- *Ansible_hostname and inventory_hostname:*
  https://www.middlewareinventory.com/blog/ansible-inventory_hostname-ansible_hostname-variables/#Inventory_hostname_variable_Introduction
- *Jinja2 syntax: https://documentation.bloomreach.com/engagement/docs/jinja-syntax*

Installation

On the management node, execute the script `./install.sh <cluster_CIDR-mask>` (cluster CIDR/mask according to DHCP settings from section 3.2, e.g., 192.168.1.0/24) and observe progress notifications that should be similar to the ones shown in the frame below. Successful installation is reported by the `failed 0` indication for each cluster node visible at the end. Note: guides for troubleshooting in case of errors is out of the scope of this document.

```
#start of the installation

xubuntu@xubulab:~/cluster-pi/pi-cluster-install$ ./install.sh 192.168.1.0/24   ← provide your subnet CIDR
[sudo] password for xubuntu:
Host kpi091
The authenticity of host &apos;192.168.1.38 (192.168.1.38)&apos; can&apos;t be established.
ECDSA key fingerprint is SHA256:XNMqIKnMyhhHPzMURgUSFoalg/Xfz2Raysl2/XEgoj0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
ubuntu@192.168.1.38&apos;s password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh -p &apos;22&apos; &apos;ubuntu@192.168.1.38&apos;"
and check to make sure that only the key(s) you wanted were added.

Public key successfully copied to 192.168.1.38

# . . .
# for each host similar output as above during the first install attempt; subsequent attempts can produce
# simpler output than above (only line: Public key successfully copied to 192.168.1.xy per host)

PLAY [cluster]
*********************************************************************************************************
```

[1] Actually, you can install a newer version (e.g., 26) of k3s and kubectl. There's a high chance everything will work fine.

```
TASK [Gathering Facts]
*********************************************************************************************
ok: [kpi093]
ok: [kpi092]
ok: [kpi091]
ok: [kpi094]

TASK [preparation : Update the /etc/hosts file with localhost name]
*************************************************
changed: [kpi093]
changed: [kpi091]
changed: [kpi094]
changed: [kpi092]

(. . . progress notifications)
# end of the installation

PLAY RECAP
*********************************************************************************************************************
kpi091                     : ok=20   changed=10   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
kpi092                     : ok=12   changed=8    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
kpi093                     : ok=12   changed=8    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
kpi094                     : ok=12   changed=8    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

config                                                                        100% 2964   764.3KB/s   00:00
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$
```

4.2. Configuring kubectl client

As an example, the content of a real `config` file is shown in the frame below (`DATA+OMITTED` and `REDACTED` stand for long strings being actually present in the file and serving as k3s certificates required for accessing the clusters with the kubectl commands). These (long) strings have to manually copied from the config file downloaded from the master node of the cluster. As can be seen, data for those three clusters is stored in this config file, two of which are RbPi clusters and the third one is of another origin.

After configuring the `congfig` file, to finalize this exercise, you can run a couple of popular kubectl commands to verify if the installation works properly (at least on a basic level). In case of noticeable misbehaviors it is recommended to reinstall the cluster from scratch (analysing possible mistakes in manual configurations that have been introduced in the bash script and in Ansible templates).

```
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://10.254.184.164:6443
  name: cluster.local
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.1.42:6443
  name: kpi03
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.1.38:6443
  name: kpi09
contexts:
- context:
    cluster: kpi03
    user: kpi03.admin
  name: admin@kpi03
- context:
    cluster: kpi09
    user: kpi09.admin
  name: admin@kpi09
- context:
    cluster: cluster.local
    user: kubernetes-admin
  name: kubernetes-admin@cluster.local
```

```
current-context: admin@kpi09
kind: Config
preferences: {}
users:
- name: kpi03.admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
- name: kpi09.admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$

# setting current context to work with
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$ kubectl config use-context admin@kpi09
Switched to context "admin@kpi09".

# using kubectl commands in current context that has been set above (you can run a coup[le of other
ones as "kubectl get pods -A" or similar to verify the installation works fine)
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$ kubectl get deployments -A
NAMESPACE     NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
kube-system   local-path-provisioner   1/1     1            1           83m
kube-system   coredns                  1/1     1            1           83m
kube-system   metrics-server           1/1     1            1           83m
kube-system   traefik                  1/1     1            1           82m
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$
```