ZSUT

Zakład Sieci i Usług Teleinformatycznych

Kubernetes laboratory

Lab part 1: Installing k3s on Raspberry Pi cluster

Prepared by: dr inż. Dariusz Bursztynowski

Acknowledgements: The author is grateful to the following students for their help in preparing the lab (in alfabetical order): Hubert Daniłowicz, Franciszek Dec, Jerzy Jastrzębiec-Jankowski, Maciej Maliszewski, Miłosz Marchewka, Adrian Osędowski, Cezary Osuchowski, Piotr Polnau, Jan Sosulski, Filip Wrześniewski, Marta Zielińska

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

Last update May 2024

Table of contents

Table of contents 2
1. Introduction 3
2. Preparing the management host 4
3. Preparing Raspberry Pi hosts and local DHCP 5
3.1. Preparing RbPi hosts 5
3.2. Configuring local (Linksys) DHCP for using the cluster 6
3.3. Enabling CPU temperature control on RbPi (optional) 8
3.4. Enabling VPN access to the cluster 8
4. Installing k3s and configuring kubectl client 8
4.1. Installing k3s 8
4.2. Configuring kubectl client and checking the liveness of Kubernetes 11
5. Homework 12

1. Introduction

WARNING: Before powering the devices you should make sure you are using the right power supply. The power supplies in your set have the same type of DC plug but they **DIFFER SIGNIFICANTLY in the output voltage** (Linksys device powers from 12V DC while TP-Link switch needs 53V DC). Powering Linksys device from power supply of TP-Link switch results in instantaneously damaging the Linksys.

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 Linux bash scripts, to more declarative automation tool being Ansible in our case. A side effect of this approach for those who have not used with Ansible is to learn the 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. In Internet-connected mode one either uses the option `imagePullPolicy: IfNotPresent` or the option `imagePullPolicy: Always` (default) depending on preferences. But in Internet-disconnected mode the option `imagePullPolicy: IfNotPresent` should always be used (and the images should be stored in a local repo).

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

Note: there are many ways to install Kubernetes and several known approaches to install k3s on Raspberry Pi cluster using Ansible. A good guide, although not the most recent one anymore, describing purely Ansible-based approach is available under the link http://www.pidramble.com/ and https://github.com/geerlingguy/raspberry-pi-dramble. One can learn a lot from there by observing the use of a number of interesting Ansible constructs.

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.

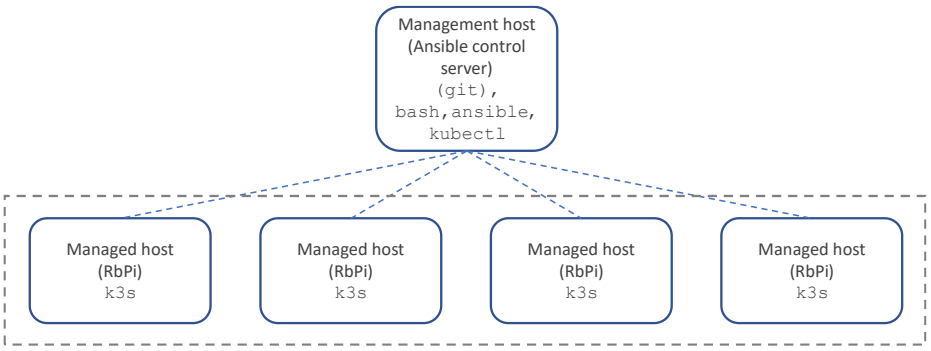

Figure 2. Management node (host) and managed nodes (hosts).

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 distributions (we use Ubuntu 22.04 LTS). In case of using other distribution of Linux appropriate adaptations of the commands may be needed.

- **If you are using a VM for the management host, configure its network interface as "bridged"** – this is mandatory for the pre-config bash script to run successfully.
- Generate SSH key – run the command as below on the management host; no need to set key password, etc., so just hit enter a couple of times. The key will be saved in file ~/.ssh/id_rsa (file ~/.ssh/id_rsa.pub will hold the public key).

  ```
  ssh-keygen
  ```

- Install `net-tools`[1] and `nmap` utilities 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 use a guide where configuration of Ansible managed hosts (i.e., hosts to be automatically configured using Ansible) is considered you can skip these parts for the moment unless you already have some test hosts running. Example Ansible installation guides are given in the list below (but you may want to use other guides – feel free to use a source of your choice):

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

  https://docs.ansible.com/ansible/latest/installation_guide/installation_distros.html#installing-ansible-on-specific-operating-systems

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

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

[1] Although many net-tools programs are obsolete today we still use some of them in this guide. You may skip this step and use respective commands from iproute2 package if you want. nmap is used in our script to detect Raspberry Pi hosts in the local network.

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

  https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-binary-with-curl-on-linux

  Important: Although there are newer versions of Kubernetes available, please follow the guidelines given in this note. We recommend installing v1.28 client of `kubectl` on the management node, and v1.28 control plane (i.e., k3s v.28 in our case) on the cluster. This is because the version of kubectl client has to match the version of Kubernetes control plane (according to the rule specified under the first link above) AND Kubernetes version has to comply with the compatibility matrix[2] of kube-prometheus stack for monitoring (that we will use in our third lab). Specify your version 1.28 in the curl command according to the *Note* under the link provided above.

- Note: Using kubectl command requires the existence of directory `~/.kube` where kubectl `config` file with credentials to access your clusters is stored. If `.kube` directory is not created during kubectl installation than it will be created by our k3s pre-install script `install.sh` (we will describe `install.sh` later in this document).

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 (slots 1, 2 – seen from left to right) and two RbPi-s with 8GB RAM (slots 3, 4). 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: if you think of installing Ubuntu 22.04 LTS server / 64bit ARM on your RbPi-s please note that, according to https://github.com/k3s-io/k3s/issues/5443, Ubuntu 22.04 / 21.10 on Raspberry Pi need installing EXTRA KERNEL MODULES (over 100MB of additional code), before starting k3s:

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

    Otherwise k3s will keep restarting. **Therefore we recommend installing Ubuntu 20.04 LTS on RbPis**.

  - before burning the image set appropriate options in Raspberry Pi Imager (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 logging *via* ssh to each host:

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

        (for the first time, the passwd will be as you set it in *RbPi Imager*, otherwise it will be *ubuntu* and you will be prompted to change it during the first loging)
    - *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)
    - *Enable ssh -> Use password authentication* (checked); this will be used by the configuration bash script to upload ssh credentials to the cluster nodes.

[2] See https://github.com/prometheus-operator/kube-prometheus#compatibility.

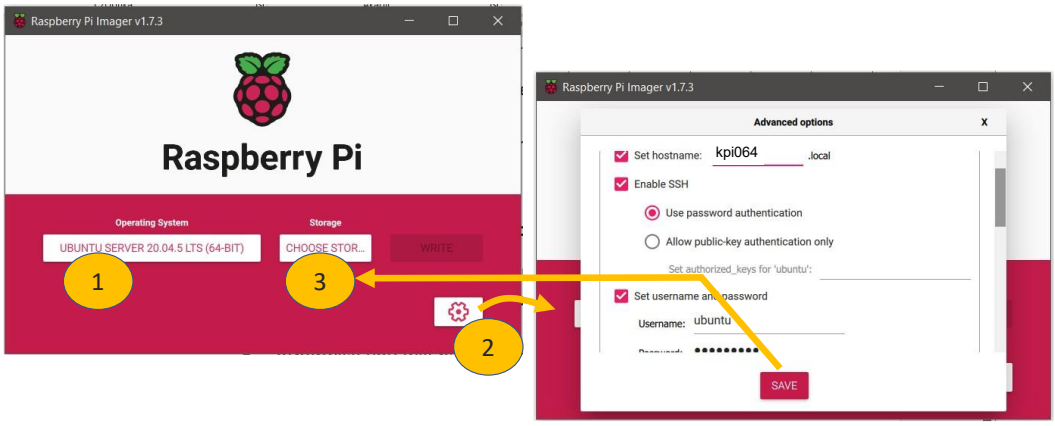

Figure 3. Raspberry Pi Imager settings (notice setting the hostname in our case). Caution: the newest version of Raspberry Pi Imager presents a separate panel SERVICES for SSH settings – check Enable SSH therein.

Note: In case of reinstalling the cluster from scratch (burning the images again, etc.), if 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 (adjust host names kpi091, … as below according to your environment):

`# 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, during every run, script `install.sh` adds hosts to the `~.ssh/config` file. It is therefore recommended to delete (by editing) those hosts from file `~/.ssh/config` or otherwise hosts can occur multiple times in `~/.ssh/config` (this is not critical but multiple occurrences can be confusing).

3.2. Configuring local (Linksys) DHCP for using the cluster

Actually, two configuration settings should be made:

- 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 actual "ordering" (and the role) of the RbPi-s boxes in our cluster.*** *In fact, the names of RbPi-s assigned while burning the image will not matter*

*for the final naming (**Linux hostname**) of the nodes. **Linux hostnames of our RbPis will actually be fixed for the use in the cluster by the bash installation script**. Keep it in mind while assigning addresses to your RbPi-s in DHCP.*

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, below we assume the DHCP server in the local network can be accessed and we can reserve static IP addresses for our RbPi-s using DHCP. An example procedure is as follows.

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 (or a little bit more) for the cluster will suffice perfectly**.
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 are assigned as expected (adjust the CIDR/mask to your environment):

   ```
   $ sudo nmap -sP 192.168.1.0/24 | awk '/Nmap scan report for /{ipaddress=$NF}/28:CD:C1|B8:27:EB|D8:3A:DD|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 adopt some rules for assigning RbPi-s' IP addresses and some dependency between the address and the name of a given RbPi[3]; 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 and IP addresses have been assigned as expected check the connected devices in the DHCP server (panel Status/Device list on WRT54GL). 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 1: When you ssh to a RbPi host for the first time then it's data (e.g., name or IP address) is registered in 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>`) on the management node.

Note 2: There are many ways to set the IP addresses for cluster nodes and another example is as follows: Log into the control panel of the router provided in the lab kit. Then, next to the nodes you are interested in, set a static IP address. For example, for a device with the reported hostname kpi091, set the IP address of 192.168.96.91. After restarting, the Raspberry Pi will retrieve the assigned address from the router, which will remain fixed afterwards.

[3] For example, my rule is to name RbPis with some constant prefix and integer suffix drawn from a set of consecutive numbers, install them in the physical case ordered from the left to the right according to ascending name suffix, and allocate consecutive IP addresses to them in a similar way – in ascending order from the left to the right.

3.3. Enabling CPU temperature control on RbPi (optional)

As an option, one can install software that enables RbPi board temperature. This is possible only for clusters with the PoE board containing a small display panel (visible in the upper part of the board). If you are interested in trying this option please refer to the following link where the sources and installation guide are available.

https://github.com/darkfence/PoE_HAT-B-temp-control

In case of Raspberry Pi 5 model(s), consult the case with the instructor.

3.4. Enabling VPN access to the cluster

It is highly recommended to setup a VPN to allow that all students can access the cluster and make experiments on their own. The procedure for setting the VPN with ZeroTier is described in our github repo in the file zt-manual.md. You can also use another VPN platform of your choice. In either case read the mentioned zt-manual.md as it also contains a description of how we can switch-off/switch-on the cluster remotely (to save energy/hardware and limit the noise form).

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 our lesson 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 cluster nodes.

IMPORTANT: One of the tasks to be completed by students is to analyse bash and Ansible files involved. Pay attention to the "style" in which automation tasks are defined 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 – read 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:

Before the main part

- Try to disable unattended upgrades on your RbPi-s. To this end ssh to each of them and run:

```
$ sudo systemctl stop unattended-upgrades
$ systemctl disable unattended-upgrades.service
```

Bash script settings

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

  This script sets the names of RbPi hosts. These names will be assigned to the hosts as their Linux *hostname*. Moreover, authorization keys are uploaded to RbPi hosts 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 (i.e., 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 assigned the lowest IP address in our cluster. The RbPi host with the next lowest 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
SSH_KEY_FILE="$HOME/.ssh/id_rsa"            ← your ssh key (will be created if missing)
MASTER_GROUP="master"                       Ansible group of nodes (one node for us)
MASTER_NODE="kpi091"                        ← the name of the control (master) node
WORKER_GROUP="node"                         Ansible group of nodes serving as worker
WORKER_NODE="kpi09"                         ← the prefix of the name of worker node
CLUSTER_GROUP="cluster"                     Ansible group of all cluster hosts
```

Last two lines of the file: we update the name of the `config` file each time it is downloaded from the master node of the cluster to preserve its uniqueness. This is achieved by setting the suffix to CURRENT_DATE=$(date +%Y%m%d):

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

The data from file `~/.kube/config-cluster-$CURRENT_DATE` shall be used by you to configure local (on the management host) *kubconfig* file named `config`. It is used by *kubectl* on the management host to access the cluster.

Note: script install/sh will insert user credentials (name, password) into Ansible inventory file in plain text. That is not a best practice and some encryption method should be used in production. In our lab, however, we take a minimalistic approach. More on secrets in Ansible, e.g.:
https://www.redhat.com/sysadmin/ansible-playbooks-secrets

Ansible templates settings

- **File:** …**/pi-cluster-install/inventory/group_vars/all.yaml**
  Settings
  `k3s_version: v1.28.9+k3s1` ← recommended[4] (install kubectl client v.28 on the management host)
  `ansible_user: ubuntu` ← adjust to your RbPi user name (authorized user name on your 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 running the script.

Note: Selected links that can help analyse 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

[4] We use k3s release 1.28 although 1.30 is already available. As of this writing, however, 1.28 is the highest release the kube-prometheus project we will use in the lab is compatible with.

- *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
(Note: remember to `ssh-keygen -R <host-IP >` for each host if cluster nodes have been reinstalled.)

On the management node, execute the script: **`$ ./install.sh <cluster_CIDR-mask>`** (set the cluster CIDR/mask according to your 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 in the REACP PLAY at the end. Note: detailed guides for troubleshooting in case of errors is out of the scope of this document. A simple one is to reinstall your cluster from the beginning sticking to the instructions.

```
#start of the installation

xubuntu@xubulab:~/cluster-pi/pi-cluster-install$ ./install.sh 192.168.1.0/24 ← set your Linksys 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/Xfz2Rays12/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]
*********************************************************************************************************

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$
```

If you happen to see something like below it means there is a conflict with unattended upgrades running on the hosts. This is not a well-solved issue on Ubuntu with Ansible (https://github.com/ansible/ansible/issues/51663). A simple workaround is to wait some time (e.g., 15-20 minutes) and run script install.sh once again.

```
TASK [upgrade : Upgrade all packages on server] ********************************************************************
fatal: [kpi091]: FAILED! => {"changed": false, "msg": "&apos;/usr/bin/apt-get dist-upgrade &apos; failed: E: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process 2781 (unattended-upgr)\nE: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?\n", "rc": 100, "stdout": "", "stdout_lines": []}
fatal: [kpi092]: FAILED! => {"changed": false, "msg": "&apos;/usr/bin/apt-get dist-upgrade &apos; failed: E: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process 2778 (unattended-upgr)\nE: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?\n", "rc": 100, "stdout": "", "stdout_lines": []}
fatal: [kpi093]: FAILED! => {"changed": false, "msg": "&apos;/usr/bin/apt-get dist-upgrade &apos; failed: E: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process 2784 (unattended-upgr)\nE: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?\n", "rc": 100, "stdout": "", "stdout_lines": []}
fatal: [kpi094]: FAILED! => {"changed": false, "msg": "&apos;/usr/bin/apt-get dist-upgrade &apos; failed: E: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process 2773 (unattended-upgr)\nE: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?\n", "rc": 100, "stdout": "", "stdout_lines": [
```

4.2. Configuring kubectl client and checking the liveness of Kubernetes

As an example, the content of a simple `config` file describing one cluster is shown in the frame below (`DATA+OMITTED` and `REDACTED` stand for long strings (the keys) contained in the file and serving as k3s certificates required to authorize kubectl commands executed on the management host). These (long) keys should either be manually copied from the `config` file downloaded from the master node of the cluster or locally stored in separate file(s) referred in our local `config` file. Notice that using this method allows one to register any number of clusters/contexts in the `config` file.

After preparing the `config` file, you can run a couple of basic kubectl commands to verify if the installation works properly (on a basic level). A healthy installation should at least report all pods in running state (give it a couple of minutes to stabilize; 3-5 min. is sufficient). In case of misbehaviours you can either uninstall the cluster using playbook_uninstall_k3s.yaml and install again[5] or reinstall it from scratch (you can analyse the bash script and/or Ansible templates for possible bugs you might have introduced).

```
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$ kubectl config view

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.96.91:6443
  name: kpi09
contexts:
- context:
    cluster: kpi09
    user: kpi09.spiw
  name: spiw@kpi09
current-context: spiw@kpi09
kind: Config
preferences: {}
users:
- name: kpi09.spiw
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
xubuntu@xubulab:~/cluster-pi/pi-cluster-install$

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

# CHECKING THE LIVENESS OF THE CLUSTER
```

[5] `$ ansible-palybook palybook_uninstall_k3s.yaml -i inventory/hosts.ini`
`$ ansible-palybook palybook_install_k3s.yaml -i inventory/hosts.ini`

```
# using very basic kubectl commands in current context ("kubectl get nodes" or "kubectl get pods -A
or kubectl get deployments -A, etc.) to verify if cluster responds.
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$
```

5. Homework

1. Propose (not necessarily implement and/or test) your own "policy" (method) of assigning names/IP addresses to cluster nodes.
2. Pay attention to/analyse the *roles* in our Ansible scripts. Explain how those roles relate to the notion of Ansible *plays*?
3. Assuming you managed to install k3s on the cluster, how would you suggest to improve the lab in the future? What aspects/capabilities/skills related to automation are missing or could be addressed more thoroughly, and which ones could be tackled to a lesser extent?