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 alphabetical 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: April 2025

Table of contents

1. Introduction

WARNING: Before powering up your devices, please make sure you are using **the correct power adapter**. The power adapters in the boxes have the same DC plug type, but they are **SIGNIFICANTLY DIFFERENT** in terms of output voltage (WiFi routers as Linksys are powered by 12V DC, while the TP-Link switch requires 53V DC). **Powering our WiFi router from TP-Link switch power adapter will immediately damage the router.**

***Note**: This guide was originally written under the assumption that the cluster was using **Raspberry Pi 4B boards and a Linksys WRT54GL router with Fresh Tomato firmware**. There are currently two cluster variants differentiated by the type of RPi board used: Raspberry Pi 4B and Raspberry Pi 5 (all clusters are homogeneous in terms of the type of RPi board). All RPi4 clusters are equipped with 4 RPi boards and a Linksys WRT54GL router. Clusters with RPi5 contain 3 RPi boards and either a WRT54GL router (5 clusters, admin username and password are listed on the sticker) or a TOTO Link router (6 clusters, user name is `admin`, the password[1] is either `admin` or `admin123`). Some of the WRT54GLs run Fresh Tomato firmware, and some run the original DD-WRT. For each WRT54GL, the admin username and password are listed on a sticker attached to the router case. Each kit with TOTO Link router includes a manual with enough information to perform all required configurations. All in all, the differences between both kits are actually minor, so we have decided not to produce multiple descriptions that would explicitly cover each router/firmware combination. As a result, sometimes the guidelines should be interpreted to figure out the settings required in each particular case.*

We are going to install k3s in our RPi cluster. K3s is a lightweight Kubernetes distribution from Rancher especially suitable for ARM platforms. The installation procedure is based on both a bash script and Ansible playbook. The goal of using the mix of techniques is to compare purely imperative configuration automation, here represented by a Linux bash script, to a more declarative automation that comes with Ansible in our case. A side effect of this approach for those who have not used Ansible is the opportunity to become familiar with the tool.

The overall procedure consists of four main steps:

- preparing the management host that will be used for running Ansible
- configuration of the management host to enable *kubectl*[2] access to the cluster
- installing the OS (Ubuntu in our case) on our Raspberry Pi-s
- installing k3s on RPi-s – this step will include also additional tasks to fine tune the configuration of RPi-s.

A complete HW setup[3] of the lab consists of the cluster itself (three or four RPi-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).

[1] Note that logging to TOTO Link additionally requires entering a verification code displayed in the logging panel.

[2] You are free to use your preferred Kubernetes user interface, e.g., K9s, etc. You will have to install one on your own.

[3] Keep in mind the *Note* given above.

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 the cluster 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 have been already stored in your cluster. In Internet-connected mode you can either use the option `imagePullPolicy: IfNotPresent` or the option `imagePullPolicy: Always` (default) depending on your preferences. In Internet-disconnected mode the option `imagePullPolicy: IfNotPresent` should always be used (and the images should be stored in your cluster).

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 quite recent, 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 regarding configuration automation 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.

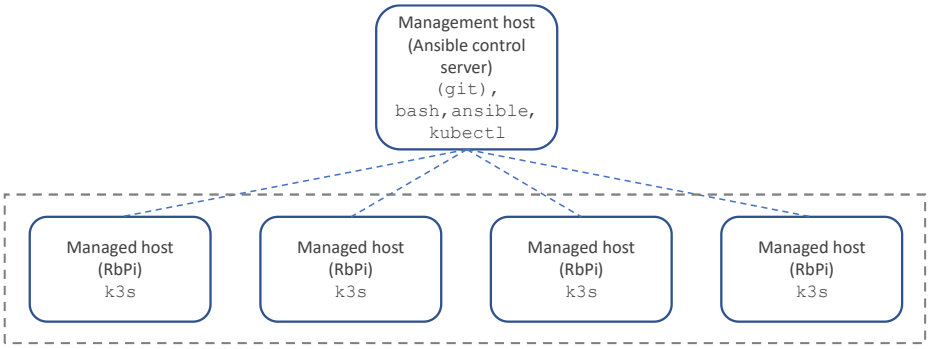

Figure 2. Management node (host) and the managed nodes (hosts).

We assume the management host runs under Linux. It can be bare metal or virtual machine. The commands included in this guide directly apply to Debian distributions (we use Ubuntu 24.04 LTS). In case of using other distribution of Linux appropriate adaptations of the commands may be needed.

- **If you are using a VirtualBox VM for the management host, configure its network interface as "bridged"** – this is mandatory for the pre-config bash script to run successfully (we use nmap utility and it needs to be run in a machine directly attached to the L2 network segment of your cluster).
- Optionally generate rsa SSH key – run the command as below on the management host (no need to set key password, etc., so just hit Enter button a couple of times). The key will be saved in a file, and by default it will be named `~/.ssh/id_ed25519` on Ubuntu 24.04 LTS (file `~/.ssh/id_ed25519.pub` will hold the public key). **Note**: this step can be skipped and the installation script will generate rsa key itself. It will also upload the public part of the key to RPi hosts. If you generate the key yourself and want it to be used, make sure the name of variable SSH_KEY_FILE in the installation script `install.sh` matches the name of your key.

```
$ ssh-keygen -t rsa -b 4096 -N ""      <== -N "" – no passphrase will be generated
```

- Install `net-tools`[4], `nmap` and `arping` utilities on the management host.

```
$ sudo apt-get update
$ sudo apt-get install net-tools -y && sudo apt-get install nmap -y |
 && sudo apt-get install iputils-arping -y
```

- Install Ansible on the management host (using Ansible terminology, our management host is *Ansible control node*). It is strongly recommended to install and run Ansible in Python virtual environment. A good guide covering this option is available here:

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

- Install `kubectl` command line tool on the management node using, e.g., this method:

  - Binary download of the recommended version for amd64 (see also the note *Important* below):
    `curl -LO https://dl.k8s.io/release/v1.32.1/bin/linux/amd64/kubectl`

  Important: new releases of Kubernetes appear frequently. This guide has been tested using k3s version v1.32.1+k3s1 being the newest one at the time of starting the tests (2025.02.14). We thus recommend installing the **tested** k3s v1.32.1+k3s1 control plane on the cluster, and v1.32.1 client (i.e., `kubectl` tool) on the management node. Notice that **`kubectl` client has to match the version of Kubernetes control plane** according to the rule specified under the first link above (`kubectl` version should be within one minor version difference of your cluster) AND Kubernetes version has to comply with the **compatibility matrix[5] of kube-prometheus stack** for monitoring (we will use kube-prometheus in our third lab). In our case, k3s version is specified as Ansible group variable `k3s_version` defined in file `inventory/group_vars/all.yaml`[6]. Check it for consistency before running our Ansible playbook.

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

3. Preparing Raspberry Pi hosts and local DHCP

3.1. Preparing RPi hosts

Our cluster contains four[7] Raspberry Pi 4B boards, and a local switch that serves as Top-Of-the-Rack switch and a PoE (Power over Ethernet) power source for the RPi-s. There are two RPi boards with 4GB RAM (slots 1, 2 – seen from left to right) and two RPi-s with 8GB RAM[8] (slots 3, 4). As we will see later, RPi #1 will serve as the master (control) node in our clusters (4GB RAM is sufficient for that) while the nodes with 8GB RAM will host workloads.

- Install Ubuntu 24.04 LTS ARM on your RPi-s
  - we recommend using *Raspberry Pi Imager* application, but other tools will work well, too

[4] 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` utility serves in our script to detect Raspberry Pi hosts in the local network. As it will be also commented later, using `nmap` is the reason for having to run the installation script with both the cluster and the management host physically connected to the same L2 network segment (the use of VPN is not possible in this case).

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

[6] According to Ansible convention, the name `all.yaml` of the file means that the variables contained in the file relate to *all* hosts in the playbook. More on inventories and group variables in Ansible can be found here.

[7] Note that clusters based on RPi5 contain three RPi boards each.

[8] All RPi5 boards have 8GB RAM.

- in Raspberry Pi Imager, select: *Other general-purpose OS -> Ubuntu -> **Ubuntu server 24.04.x LTS (64bit)***

  Note: if for some reason you prefer to install Ubuntu 22.04 LTS server / 64bit ARM on your RPi-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 24.04 LTS on our RPis**.

- 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, …). Hostnames HAVE to be consistent with the convention used in bash script `install.sh`. Check it in the directory pi-cluster-install.
    - 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 *RPi Imager*, otherwise it will be *ubuntu* and you may be prompted to change it on the first logging)
  - *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 nodes.

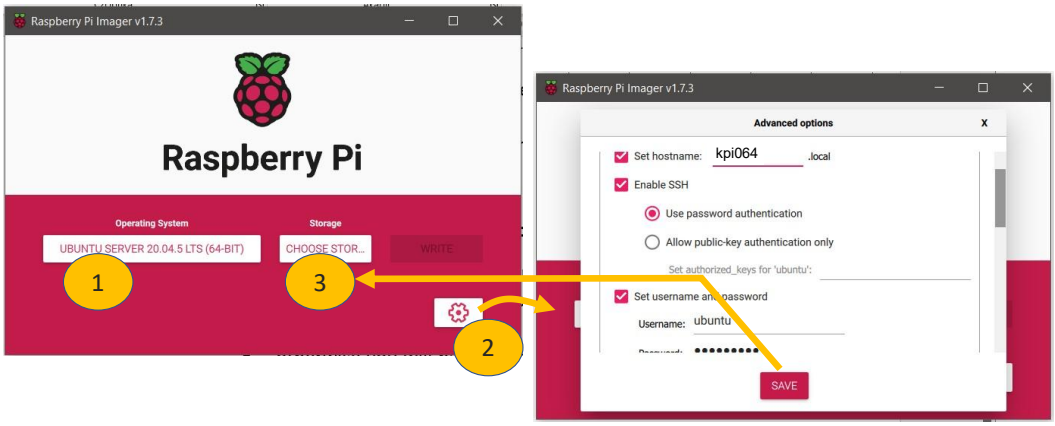

Figure 3. Raspberry Pi Imager settings (notice setting the hostname in our case). Caution: the newest version of Raspberry Pi Imager presents a separate panel SERVICES for SSH settings – check Enable SSH therein.

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
```

(Alternatively, you can manually delete `known_hosts` file altogether.)

During every run, script `install.sh` adds hosts to the `~.ssh/config` file. Before this file is filled in it will be deleted by the script so it's contents is always up-to-date. This file will be used later (also by Ansible) to ssh to the nodes by their name (Linux *hostname* of each RPi node will be configured in our Ansible inventory as `inventory_hostname`).

3.2. Configuring local DHCP

The instructions provided below basically apply to WRT54GL routers with Fresh Tomato firmware. Mapping them onto DD-WRT is quite straightforward. Setting static DHCP in TOTO Link is briefly described in Note 3 at the end of this section.

Actually, two configuration settings should be made:

- Reserving fixed IP addresses for the RPi hosts to be used in the cluster.
- Reserving a suitable address range for dynamic use in the subnet by 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 RPi boards).

Note2: 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) its subnet IP address space (panel Basic/Network/LAN) is DIFFERENT from your home subnetwork address space (for routing consistency).

***Note3:** Cluster configuration script `install.sh` we are going to use will order cluster nodes according to the value of their IP address – the higher the address (value) the higher the index the node will receive (it is so because the `nmap` utility used in `install.sh` discovers host ordered by their increasing IP address). Script `install.sh` also generates Ansible inventory file for our cluster (file `hosts.ini`) that contains `inventory_hostname` of each cluster node. The convention used in `install.sh` is that the master node of the cluster (say, node1) has the lowest IP address in the cluster (say, 192.168.1.81) and that `inventory_hostname` of each node is the same[9] as its Linux `hostname` (so, the hostname of the master would also be node1). Successive nodes are expected to be addressed in increasing order (say, node 2 -> 192.168.1.82) although the addresses do not have to be consecutive. **Therefore, script `install.sh` strictly defines Linux `hostname` and `inventory_hostame` for each node and imposes the order of their IP addresses.** You can set the name of each RPi while burning its SD card and following above naming convention. To modify these rules[10], script `install.sh` has to be changed.*

The nodes of the cluster have to be assigned fixed IP addresses. We assume one can access the DHCP server in the local network to reserve static IP addresses for each RPi. An example procedure is as follows.

1. Make sure the cluster switch is unplugged from the power source (all RPi-s are switched-off) and all RPi-s are correctly Ether-connected to the cluster switch.
2. 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**.
3. Switch on the cluster (power on the cluster switch). The management host should be connected as in Figure 1.

[9] This is our convention. In general, however, `inventory_hostname` can be different than the Linux `hostname` of the node.

[10] In fact, whole strategy could be simplified. For example, Linux hostnames could be scrapped by `nmap` in script install.sh in addition to their IP addresses and then used in the rest of the procedure without the need to explicitly define them in `install.sh`. Alternatively, hostnames defined in `install.sh` and stored as Ansible *inventory_hostnames* (as today) could be set for the hosts using Ansible module *ansible.builtin.hostname*, without taking care about host names while burning SD cards. Some other minor adaptations could be required in both cases, but the overall effect would be less manual configuration work for the user.

4. Make sure no other Raspberry Pi devices except your cluster's RPi-s are active in the network segment of the cluster. Otherwise such unneeded devices will be configured as cluster members. Run the following command from the management host terminal to verify the IP and MAC addresses of all RPi-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 /{name=$(NF-1) ;
ipaddress=$NF}/28:CD:C1|B8:27:EB|D8:3A:DD|DC:A6:32|E4:5F:01|2C:CF:67/{mac=$3 ; print name,
ipaddress, mac}' | tr -d "()"
```

5. Execute the following steps **for each** of the RPi-s listed (use the credentials you set while burning the images):
   - `ssh <your_user>@<pi_ip_address>`
   - annotate the name of the host (the RPi 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 RPi (insert its MAC and the desired IP address). A good practice is to adopt some rules for assigning RPi-s' IP addresses and some dependency between the address and the name of a given RPi[11]; save the setting (bottom of the panel)
   - shut down current RPi from the terminal: `$ sudo poweroff`.
6. Once step 6 has been completed for all RPi-s reboot the cluster switch TP-Link: login to its management panel, then *System->System Tools->System Reboot->Reboot*. 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 configuration procedures in the next section.
7. 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.

Note 1: When you ssh to a RPi 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: On Linksys routers, 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 node you are interested in, set a static IP address. For example, for a device with the reported hostname kpi091, set the IP address as 192.168.96.91. Then, on every boot, the Raspberry Pi will receive the assigned address from the router.

Note 3: Setting static IP address in TOTO Link router: **Advanced Setup** -> **Network** -> **LAN Settings** -> make note of **DHCP Client Range** that is set (you can change it, too) -> **Set Static DHCP** -> check the option **Enable Static DHCP** -> define needed settings for your nodes using free addresses from DHCP Client Range -> **Apply** -> reboot your RPis. Now, clients (RPis in our case) will get their static IP address on each boot.

3.3. Enabling CPU temperature control on RPi (optional)

As an option, one can install software that enables RPi 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 running cluster to experiment 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

[11] For example, my rule is to name RPis 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.

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 fan noise).

NOTE: it is important to note that the current version of the install.sh script mandates that the installation of k3s is executed in the target L2 network segment and NOT over VPN – the management node and the cluster have to run in the same L2 network segment. The reason for this being that nmap tool requires access to the target L2 segment to learn the MAC address of each node. Obviously L3 VPN as Zero-Tier prevents this possibility. VPN access could be used during the installation provided we either had L2 VPN or nmap was not used and a file named config (currently generated by install.sh) was filled in manually with correct IP addresses of cluster nodes before running the script.

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 RPi-s. To this end ssh to each of them and run:

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

Bash script settings

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

  This script sets the names of RPi hosts. These names will be assigned to the hosts as their Linux *hostname*. Moreover, authorization keys are uploaded to RPi hosts and local ssh files (known_hosts, config) on the management host are updated with the information on the RPi-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 RPi-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 RPi host with the next lowest IP address will become a node with index 2, and so on.

  > **Reminder on the ordering of hosts.** The rule is the following: `nmap` utility that is run in the script `install.sh` retrieves hosts in the order of their increasing IP addresses. This ordering is then applied to processing the hosts, in particular to generating their inventory_hostname in Ansible inventory (this name will be used in Ansible playbook to configure file `/etc/hosts` on each RPi – so by our convention it is equal to

Linux `hostname` and `ansible_hostname` variable). As a result, the RPi with the lowest IP address will be assigned suffix "1", the next RPi will obtain suffix "2", and so on. Thus, if you want to have a strict control over how your RPis are named (suffixed) you should carefully assign their IP addresses.

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="$1"                                cluster CIDR/mask, script parameter
USER_NAME="ubuntu"                          ← your user name on all RPi-s
PASSWORD="raspberry"                        ← your password on all RPi-s
HOST_FILE="./cluster"                       auxiliary file for IPs addresses of hosts
INVENTORY_FILE="inventory/hosts.ini"        Ansible inventory file
CONFIG_FILE="$HOME/.ssh/config"             ssh config file (to ssh to the RPi-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
```

NOTE: The names of master and worker nodes as set here be the script will be assigned by Ansible palybook to your Raspberries as their Linux `hostname`. If you want to preserve hostname as set ditinf SD card flashing Set Ubuntu hostname for nodes (equal to the inventory_hostname, i.e. the name form hosts.yaml file)

Last part of the file: we copy kubectl config file form the cluster to the management host. There are two options, (1) and (2), and only one of them can be enabled at a time (commands of the other option should be commented out). For example, with option (2) (and option (1) commented out) we download the `config` file from the master node of the cluster and store on the management node it with a unique name. This is achieved by setting the suffix in the name of the copied file to the value of variable 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. For more details on the use of options (1) and (2) please refer to the file `instali.sh`.

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

Ansible templates settings

- **File:** …**`/pi-cluster-install/inventory/group_vars/all.yaml`**
  Settings
  `k3s_version: v1.31.6+k3s1` ← recommended (install kubectl client v.31 on the management host)
  `ansible_user: ubuntu` ← adjust to your RPi user name (authorized user name on your RPi 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 that the value of variables `ansible_user` and `ansible_become_pass` in inventory are consistent with your user on the RPi-s.

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
- *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

- **First, run on the management host:** `$ sudo usermod -aG sudo [name-of-your-local-user]`**.** This will allow sourcing the script to run it in current shell as normal user but with elevated permissions (just for any case).
- **On the management node, execute the script**: **`$ source ./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. Observe the progress notifications that should be similar to the ones shown in the frame below. Successful installation by Ansible is reported by the **`failed 0`** status for each cluster node visible in the REACP PLAY at the end of the Ansible printout. Note: detailed guides for troubleshooting in case of errors is out of the scope of this document. A simple solution is to reinstall your cluster from the beginning sticking to the instructions. Depending on the cause of the failure, one can also try to only uninstall the cluster with dedicated playbook and reinstall k3s without burning microSD cards.

```
#start of 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 '192.168.1.38 (192.168.1.38)' can'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 password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh -p '22' 'ubuntu@192.168.1.38'"
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 suggests you may have not disabled unattended upgrades (ref. the beginning of this section). This is not a well-solved issue on Ubuntu with Ansible. A simple workaround is to wait some time (e.g., 15-20 minutes) and run script install.sh once again. (https://github.com/ansible/ansible/issues/51663).

```
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": []}
...
```

Last but not least, you can always check connectivity from your management host to the cluster nodes running the following Ansible ad-hoc command in the installation directory:

```
$ ansible -i inventory/hosts.ini -m ping all
```

4.2. Configuring kubeconfig file and checking the liveness of Kubernetes

If you run only one cluster, its default kubeconfig file is ready for use and nothing needs to be changed inside it. It is sufficient **to change the file name to `config`**. If you plan to run multiple clusters then the file can be prepared to assign unique names to identify them in kubectl commands. An example of prepared kubeconfig file is given below.

As an example, the content of a simple `config` file currently 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. The file contains unique identifiers (name) of various variables (name, cluster, user, current-context, etc.) that will differ from respective name of other clusters that might be registered in the 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[12] 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
```

[12] `$ ansible-palybook playbook_uninstall_k3s.yaml -i inventory/hosts.ini`
`$ ansible-palybook playbook_install_k3s.yaml -i inventory/hosts.ini`

```
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
# 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$
```

4.3. Uninstalling k3s

If something goes wrong you can uninstall k3s. There's a dedicated Ansible playbook in our repo that can be used. But one can also uninstall k3s manually using the scripts shown below on your control and worker nodes, respectively.

- uninstall k3s server: run on each server (control) node

```
$ /usr/local/bin/k3s-uninstall.sh
```

- uninstall k3s agents: run on each agent (worker) node

```
$ /usr/local/bin/k3s-agent-uninstall.sh
```

After uninstalling, you can attempt to reinstall k3s without the need to flash the OS onto your microSD cards again.

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 have successfully installed k3s to the cluster, how would you suggest to improve the lab in the future? What aspects/capabilities/skills related to automation are missing in your opinion or could be addressed more thoroughly, and which ones could be tackled to a lesser extent or skipped? For example, all the work currently executed in script install.sh can be implemented in Ansible using standard Ansible modules and creating own module for scraping hostnames and IP addresses of cluster nodes. Would it be worthwhile to introduce such an extension and in which form: extended playbook ready for use by the students as today or additional homework (project) assigned to students?

6. Conclusions

(your stuff goes here, but must be **insightful** – not just enumeration of what's been done)

7. Additional reading

- Ansible roles – nice tutorial: https://spacelift.io/blog/ansible-roles
- Ansible ad-hoc commands: https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html