NOTE: In production environment, to make a Kubernetes Cluster Highly Available, its recommended that we should always have an odd (like 3 or 5 or 7) number of Master Nodes. But here we are not using this cluster to run any production grade application, we have decided to proceed with 2 Master Nodes and 4 worker nodes.

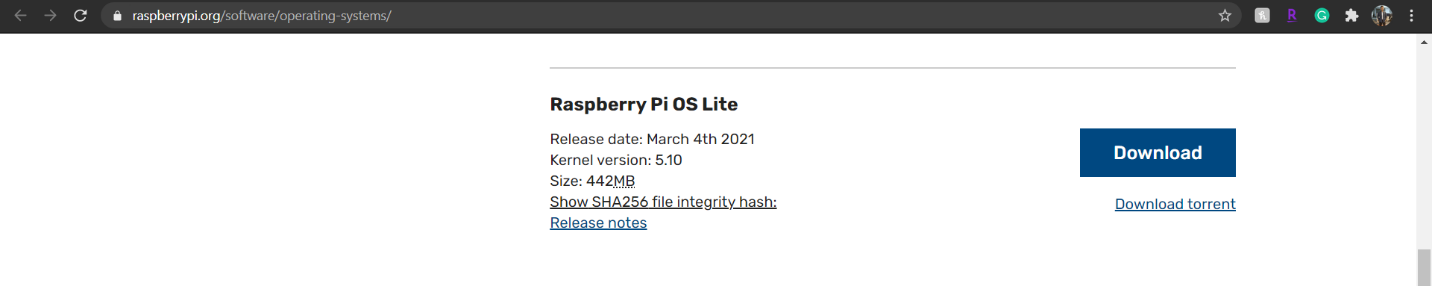
**Pre-Requisites:**

1. 6 Raspberry Pi 4 boards with 8 GB Memory
2. SD Card – 32 GB
3. 8 Port Ethernet Switch
4. Power Supply
5. Ethernet Cables – 7 (1 for each board and 1 for up linking)
6. HDMI to micro-USB Cables
7. Keyboard
8. Display

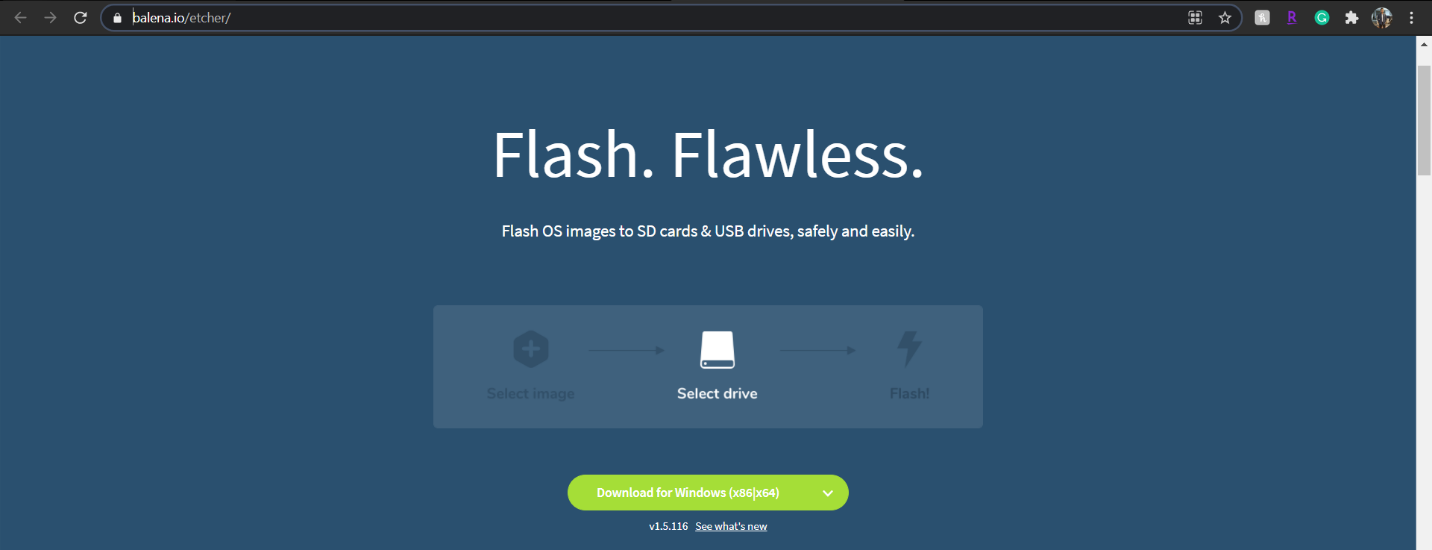
HDMI – microUSB, Keyboard, Display you can use as per your convenience and availability.

**Making SD Card Ready and Booting up Boards:**

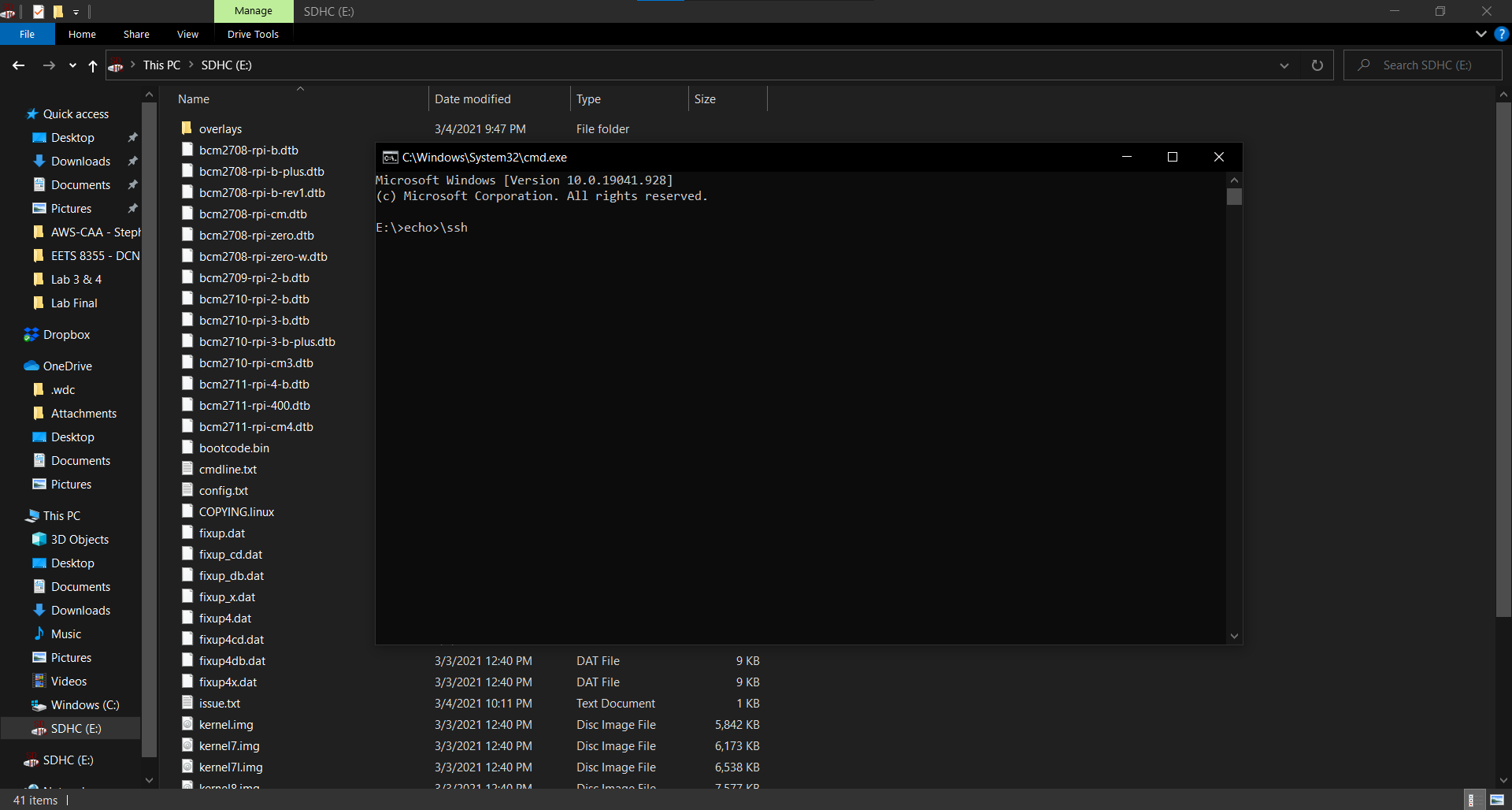
1. Download Raspbian Lite using this [link](https://www.raspberrypi.org/software/operating-systems/).



1. Download Balena Etcher to flash SD Card using this [link](https://www.balena.io/etcher/).



1. Once SD card is ready, then mount it to your Laptop and create a blank file named “ssh” in “/boot” partition.



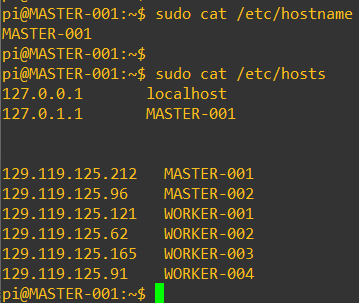
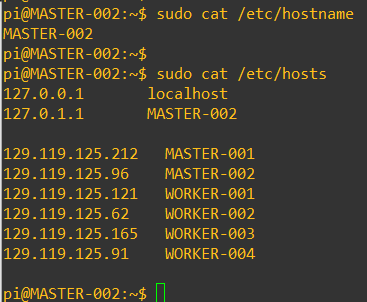
1. Once all 6 cards are ready, then insert them in all Raspberry Pi 4 boards.
2. Connect Ethernet Cables, HDMI-mircoHDMI cables (for display), Keyboards and Displays.
3. And then at last, Power On your all boards. Let the boards power on completely, usually it takes 25ms. However, depends on many other components like storage, Application etc.
4. Once all boards are up and running, login using default credentials:

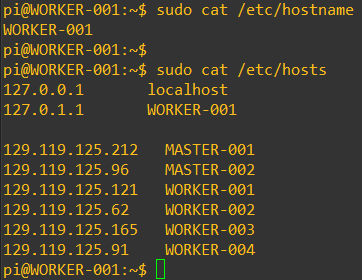
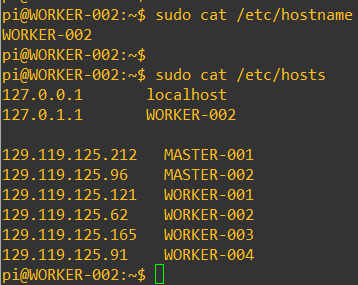
Username: pi

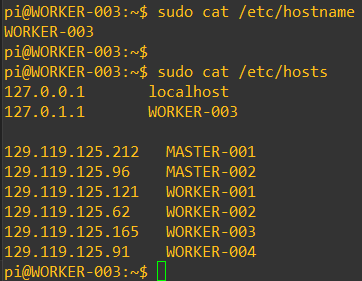
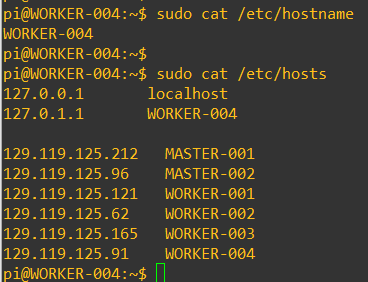
Password: raspberry

**Setting Up k3s cluster:**

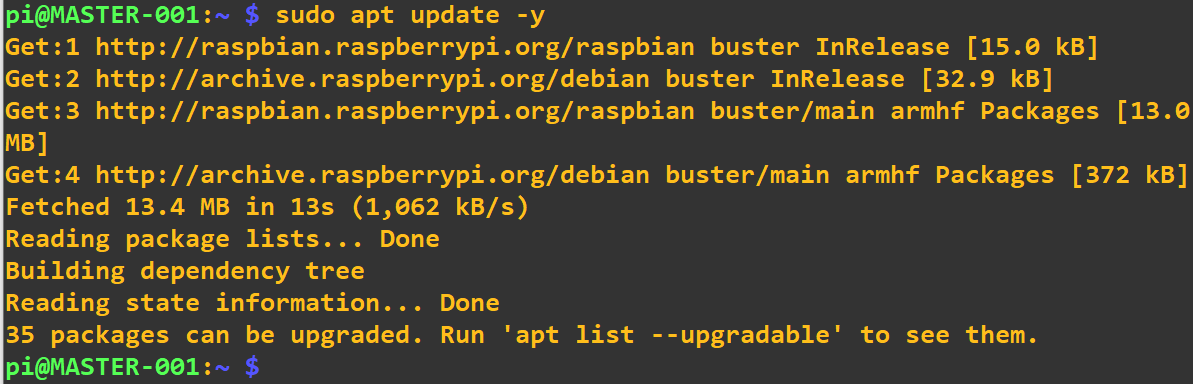
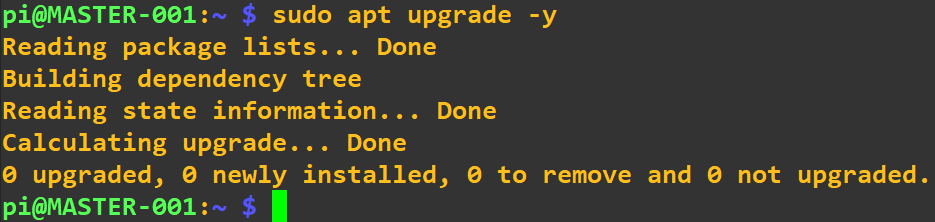
1. We are not working on Raspberry Pi 4’s Terminal directly, instead we’ve SSHed into console (Terminal) access through Putty.
2. At first change the hostnames of all the nodes you are going to use inside your cluster. As stated earlier, here we are using 2 Master and 4 Worker Nodes. So changed their hostnames accordingly.
3. After changing hostnames on all nodes, we have added entry of hostnames and IP address of every node in “/etc/hosts” file, so that they can ping one another via hostname.

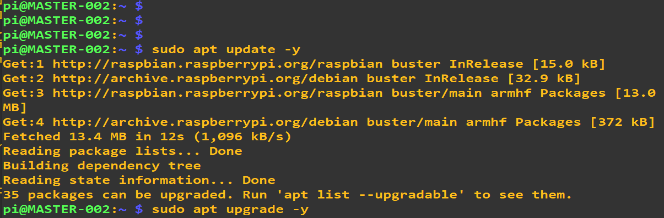
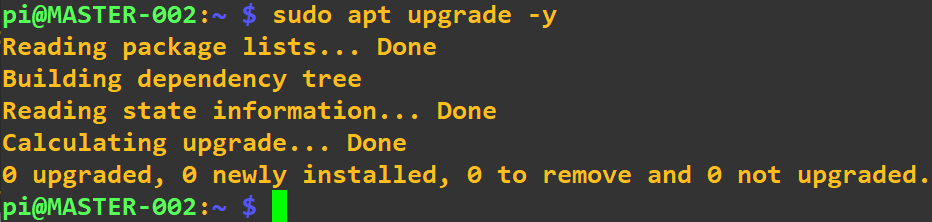
 

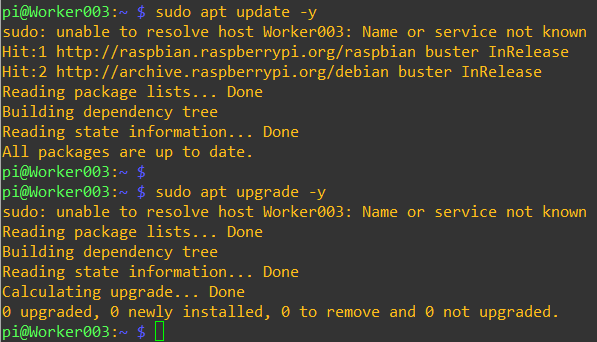
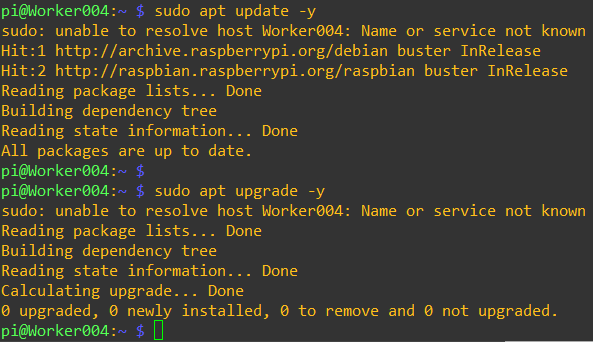
 

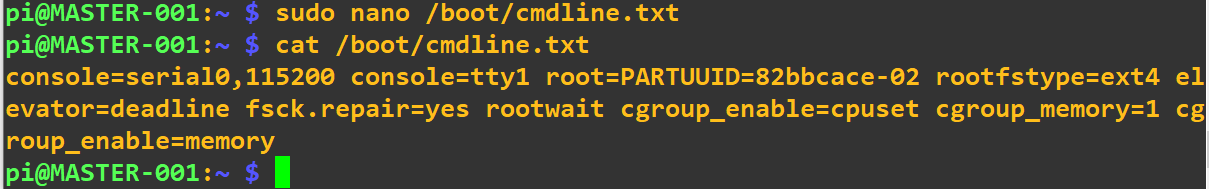
1. Once ping is successful, update DHCP assigned IP addresses into static, also add Google’s DNS address entry as well.
2. Then update and upgrade boards.

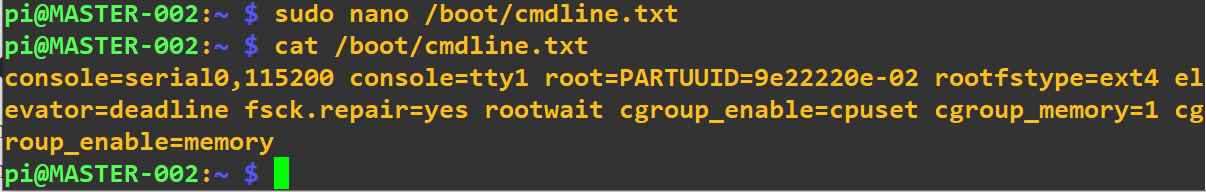
 

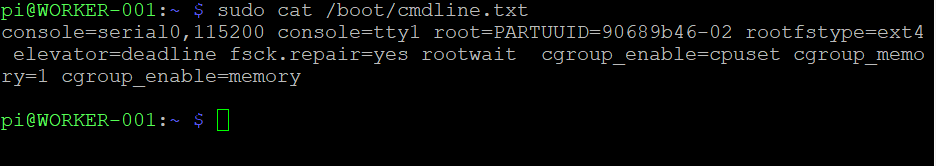
 

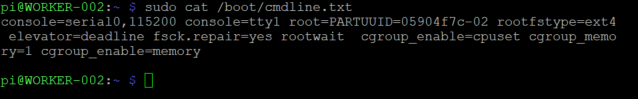
 

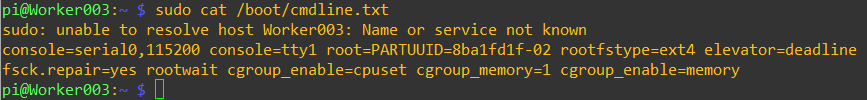
1. Then edit “/boot/cmdline.text” file in order to enable container features on Raspbian Kernel. We added “cgroup\_enable=cpuset cgroup\_memory=1 cgroup\_enable=memory” at the end of file.

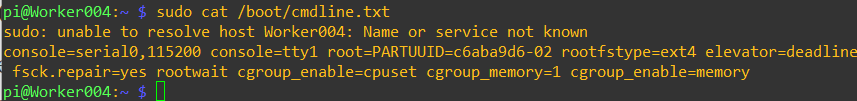










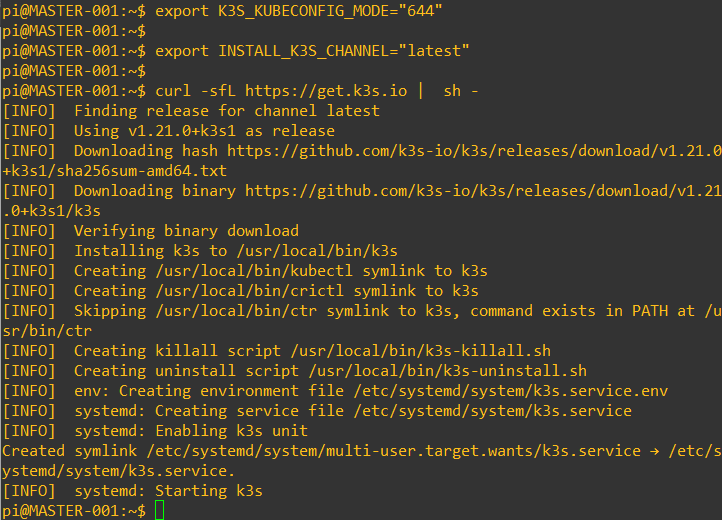


1. Once done with editing file mentioned in #6, reboot all the boards to enable new features & configurations.
2. Let’s initialize our 1st Master node. Run following commands as below:

export K3S\_KUBECONFIG\_MODE="644"

export INSTALL\_K3S\_CHANNEL="latest"

curl -sfL https://get.k3s.io | sh -

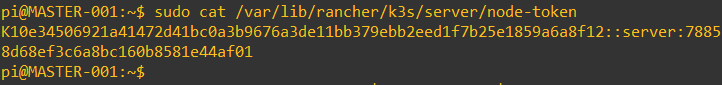


K3S\_KUBECONFIG\_MODE="644" is updating the environment variable to enable k3s cluster import into rancher.

INSTALL\_K3S\_CHANNEL="latest" explicitly mention it’s a Server configuration, if not, its agent configuration.

1. Generate new token or use the existing token from your primary Master node using which redundant Master and Worker nodes could join the cluster. To get the token run following command:

cat /var/lib/rancher/k3s/server/node-token

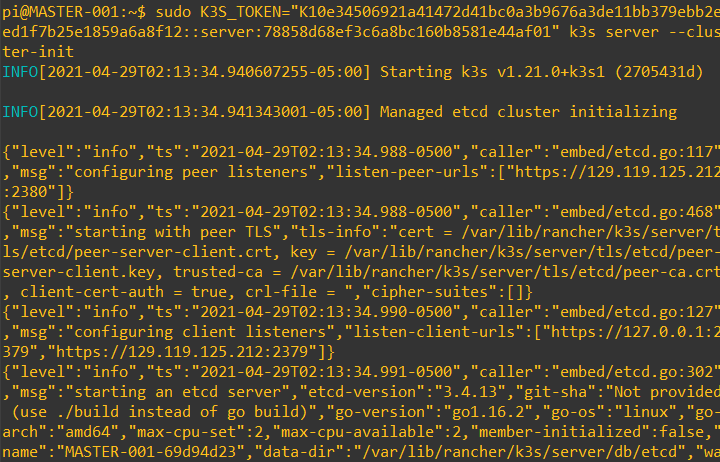


1. Kill the current k3s process running on port 6443 of your 1st Master node. To do so run following command:

sudo systemctl stop k3s

1. Run initialize cluster, run following command:

sudo K3S\_TOKEN="<node-*token*>" k3s server --cluster-init



After executing above command, you will see a bunch of processes getting initialized. Let the command run for about 2 minutes and then you can press “ctrl+c” to skip errors.

1. Once command is successfully executed, restart k3s service.

sudo systemctl restart k3s



In this way we have initialized k3s cluster and made 1st Master node ready.

1. Now it’s time to add 2nd Master to cluster. Run following commands on your 2nd Master node.

export K3S\_KUBECONFIG\_MODE="644"

export INSTALL\_K3S\_CHANNEL="latest"

curl -sfL https://get.k3s.io | sh –

Kill the k3s process running on port 6443 of your 2nd Master node by running:

sudo systemctl stop k3s

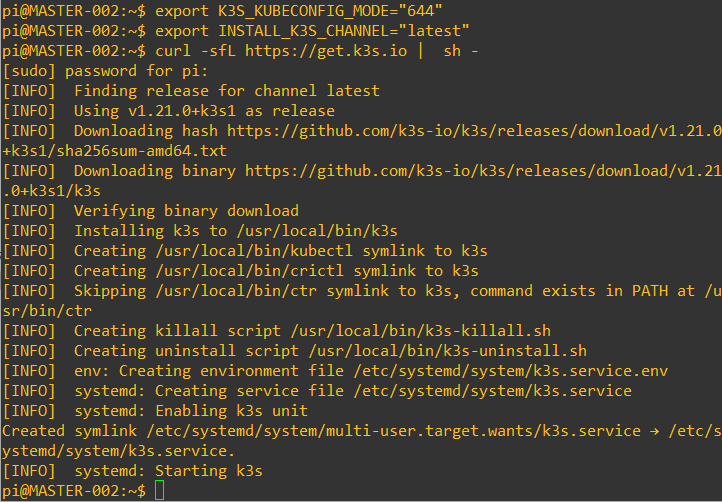
Then join your 2nd Master node to Master 1 by running:

sudo K3S\_TOKEN="<*node-token-from-master1>*" k3s server --server https://<*Master-1’s IP Address or hostname*>:6443

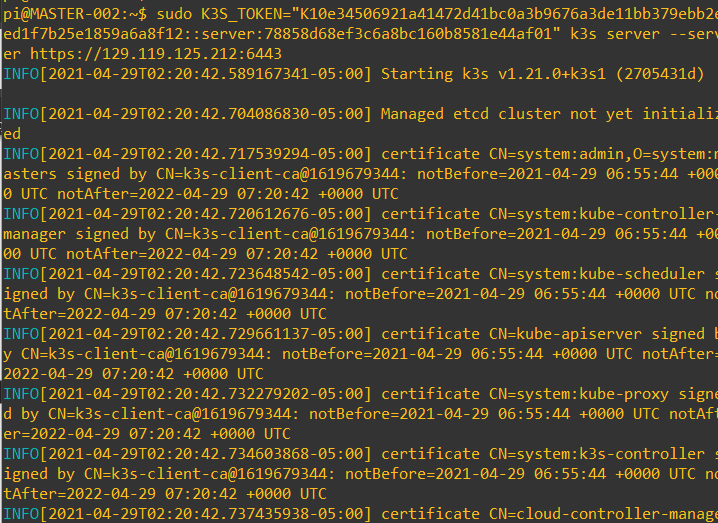
Let the command run for about 2 minutes and then you can press “ctrl+c” to skip errors.

At the end restart k3s service by running following command:

sudo systemctl restart k3s



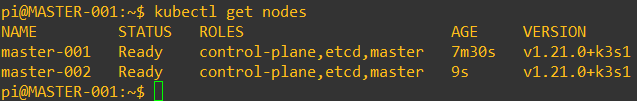






1. This adds 2nd Master node to your k3s cluster, to verify the same run:

kubectl get nodes



1. Let’s add worker nodes to cluster now. Run following commands:

export K3S\_KUBECONFIG\_MODE="644"

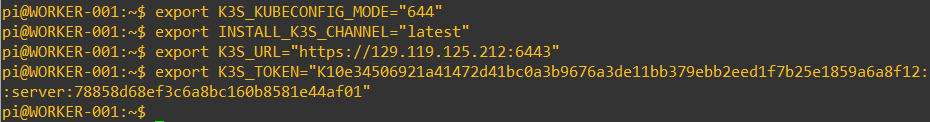
export INSTALL\_K3S\_CHANNEL="latest"

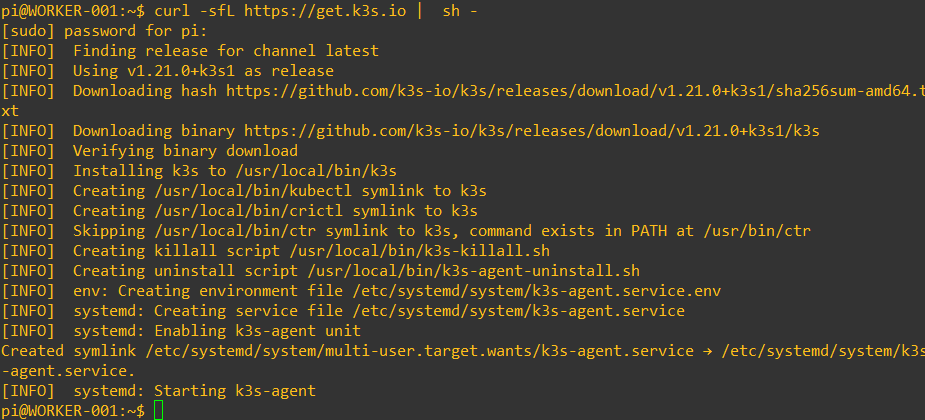
export K3S\_URL="https://<*Master-1’s IP Address or hostname*>:6443"

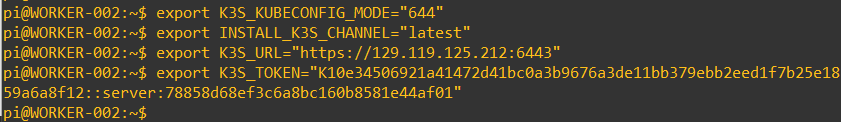
export K3S\_TOKEN="<*node-token-from-master1>*"

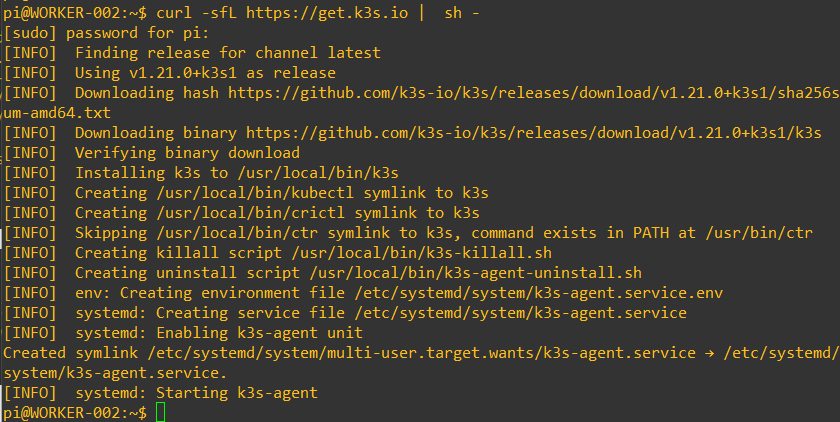
curl -sfL https://get.k3s.io | sh –

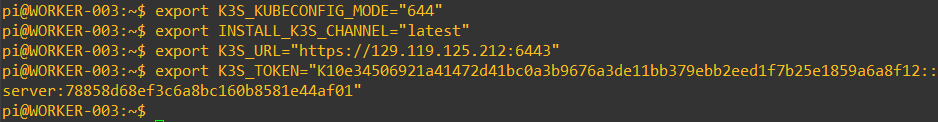
Execute same commands on all 4 worker nodes.

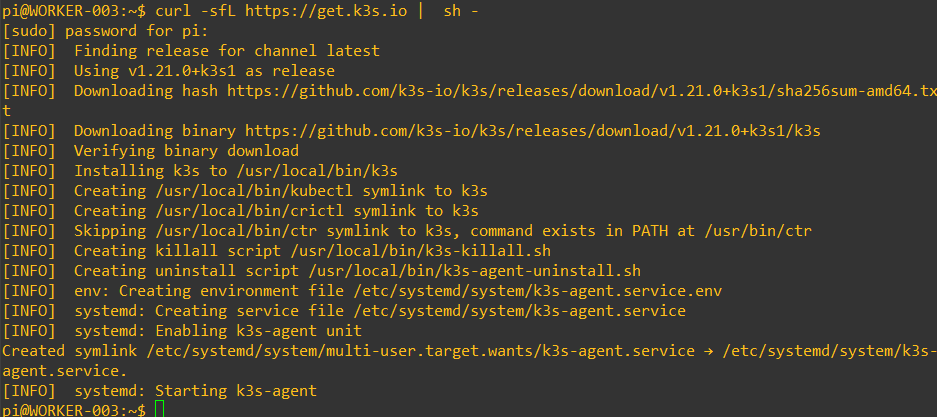


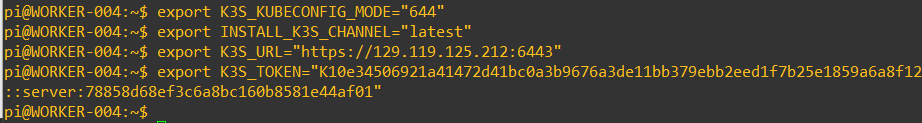


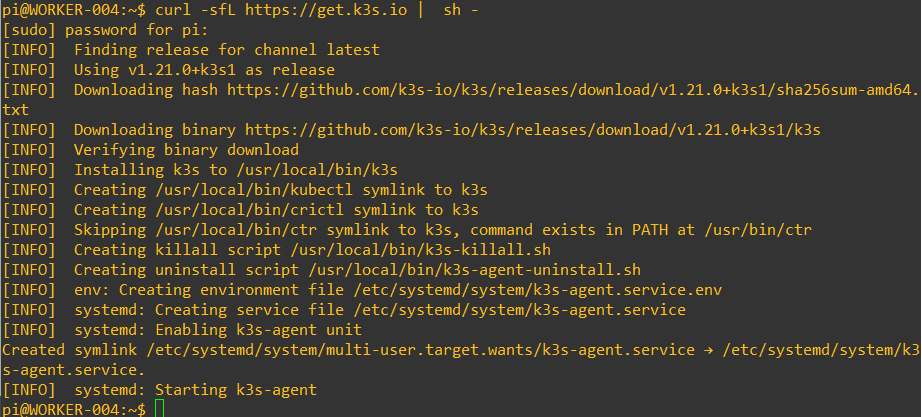




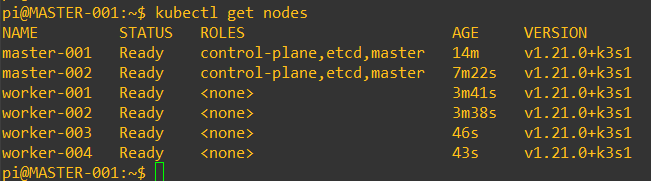




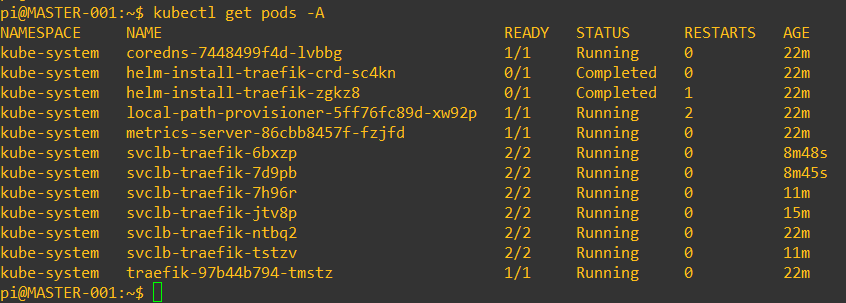




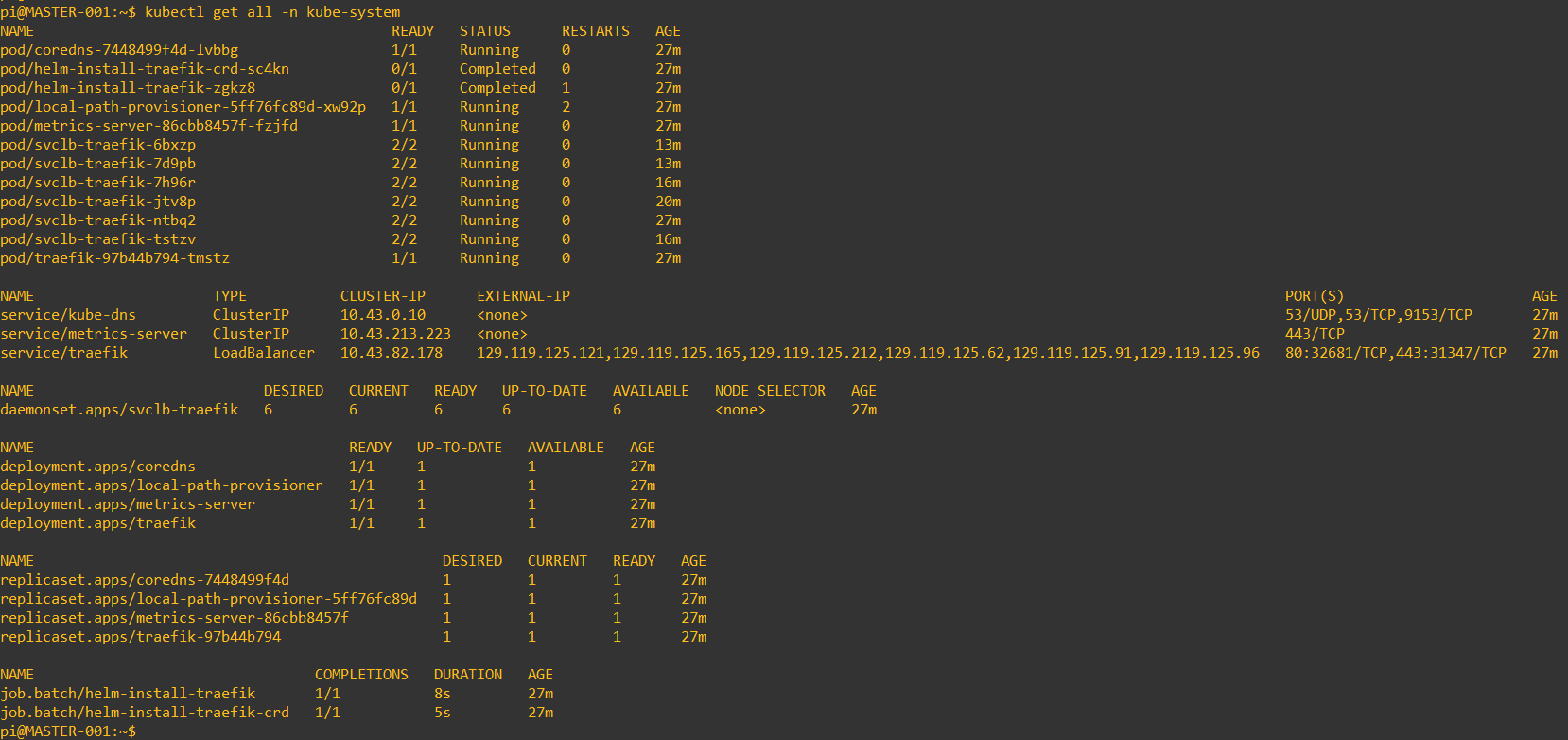
1. Once done execute “kubectl get nodes” command to verify all workers nodes are added to cluster. In the output look for “ROLES” column, it will say master nodes have control-plane, etcd, master roles added while worker nodes don’t have any role specified.



1. To check pods (from all namespaces) present inside cluster run “kubectl get pods -A” command.



1. To check all Kubernetes Control Plane components run “kubectl get all -n kube-system” command.

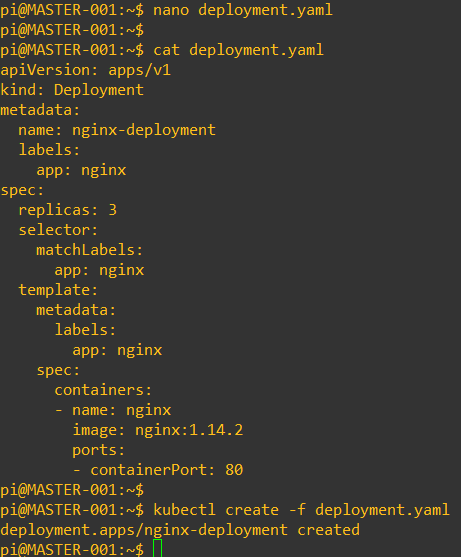


Now our Kubernetes (k3s) Cluster setup is completed.

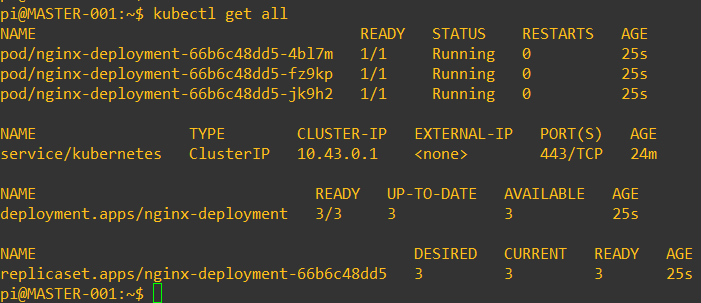
**Testing k3s cluster:**

1. Let’s run a nginx deployment to test our k3s cluster. For that we have created a Deployment manifest file (YAML file) with 3 replicas as shown below. Once file is ready, execute

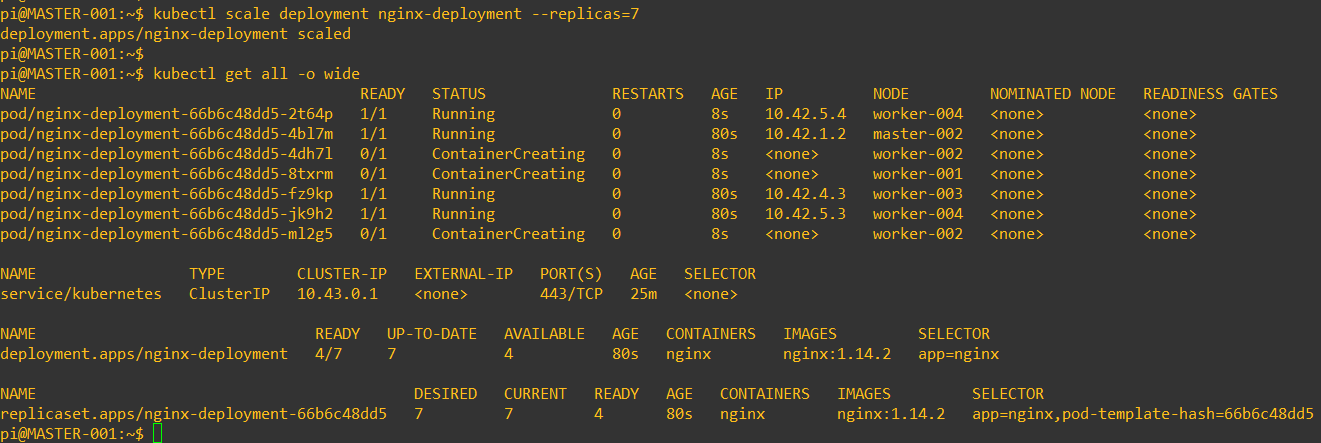
“kubectl create -f <manifest-file-name>”



1. To get all objects created by deployment, run “kubectl get all” command.

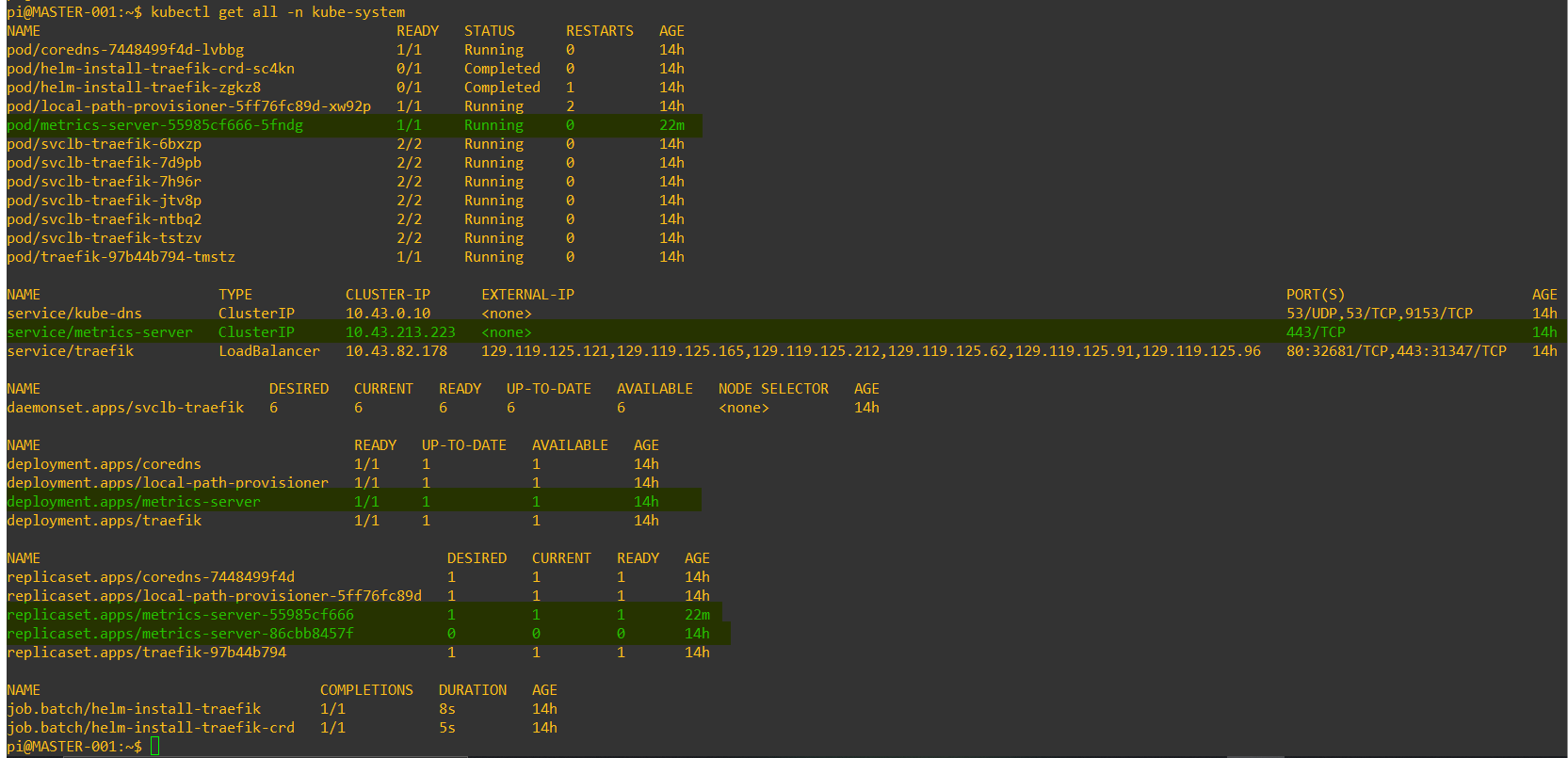


1. Let’s scale our deployment to have 7 replicas, for that run “kubectl scale deployment <deployment-name> --replicas=7”

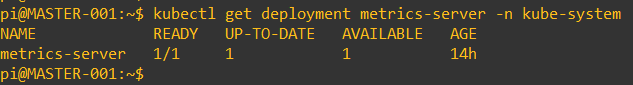


**Performance Testing:**

1. We have deployed metrics server at the time of cluster creation. But if it’s not, then one can install by running “kubectl apply -f <https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.6/components.yaml>”

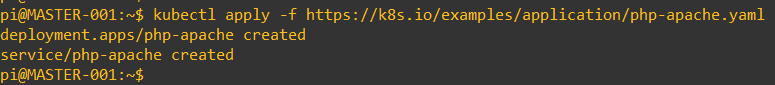


1. To verify metrics-server deployment, run: “kubectl get deployment metrics-server -n kube-system”

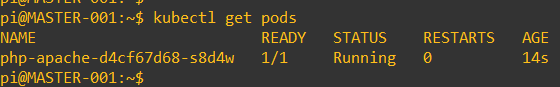


1. Now we will run a simple PHP web application by running following command.

kubectl apply -f <https://k8s.io/examples/application/php-apache.yaml>

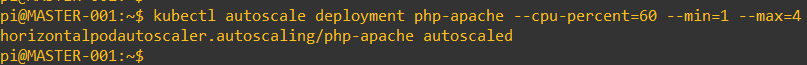


1. To verify application status, run “kubectl get pods”



1. Now setup Horizontal Pod Autoscaling (HPA) for our PHP application to automatically scale number of pods based on CPU load. We will set CPU Load percentage to 60, minimum number of application pods always running=1 and maximum replicas in case of load=4. To do so, run following command:

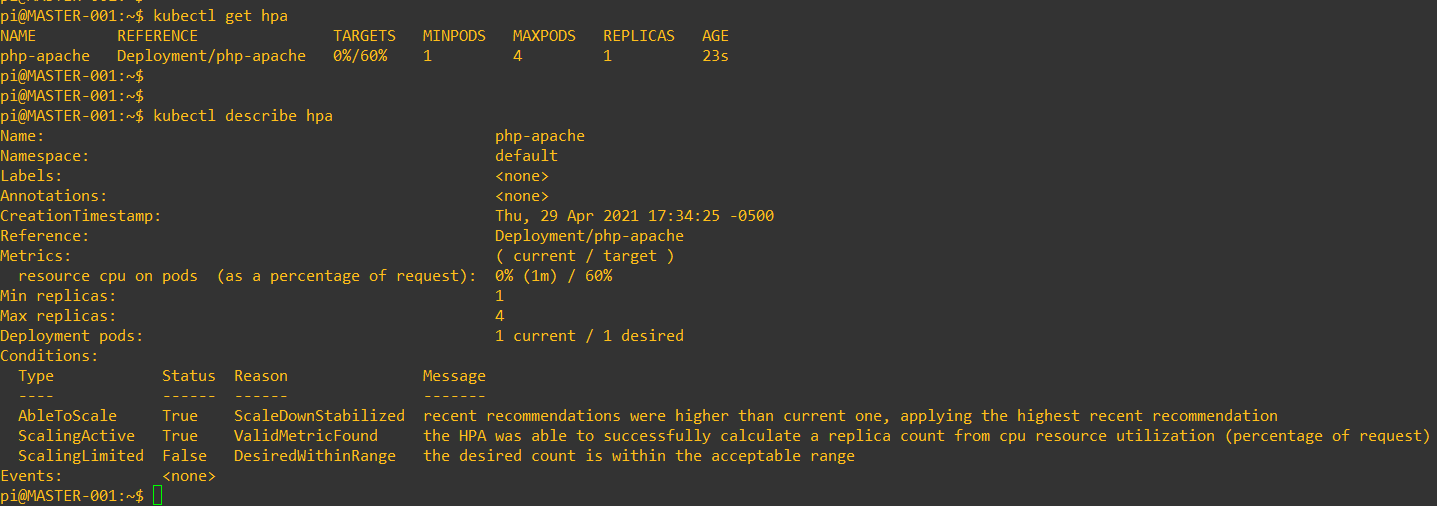
kubectl autoscale deployment php-apache --cpu-percent=60 --min=1 --max=4



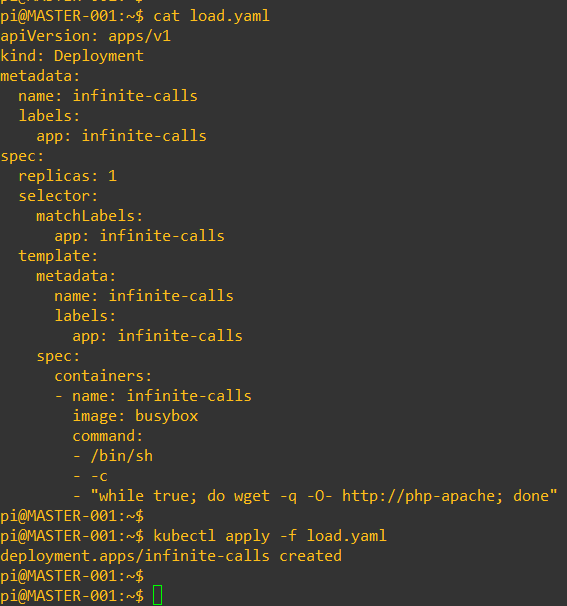
1. To check status of HPA, we can run “kubectl get hpa” command. It shows details like current CPU utilization, minimum and maximum pods defined, etc.



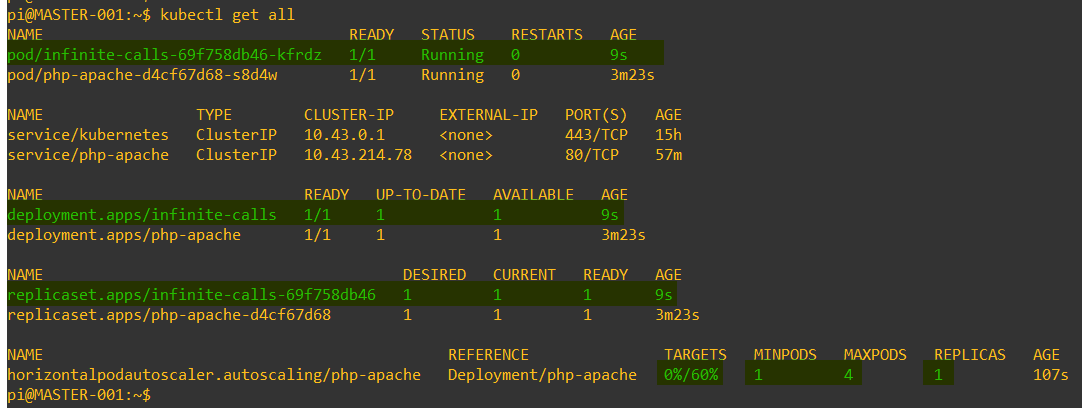
1. To get the detailed information of HPA, we can run “kubectl describe hpa” command.



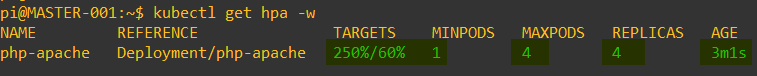
1. To test HPA, now we will put load on our PHP application. We will use “Busybox” image to send infinite requests to PHP application. So, lets create busybox deployment now. In this deployment we used wget in while loop to send recurring calls (requests) to PHP application.



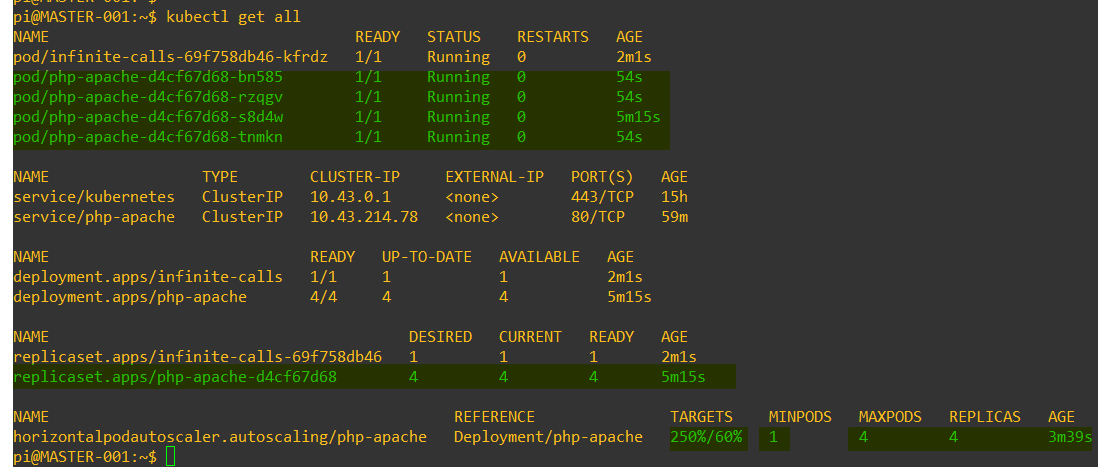
1. To verify Deployment status, run “kubectl get all”. In output you will find the replicaSet created by this deployment is running with only 1 pod and current CPU load is 0%.



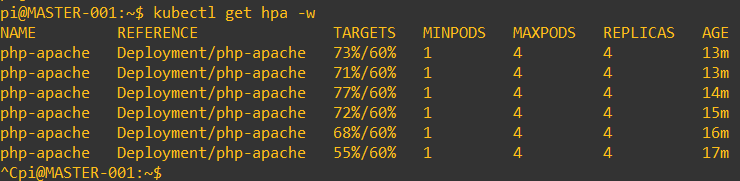
1. Then wait for few minutes and again verify HPA status to check current CPU load. Run: “Kubectl get hpa -w”



Here we will find current CPU load is beyond 60%, so HPA scaled php-apache application deployment horizontally to 4 Pods in order to manage load coming from infinite-calls deployment.



Then after monitoring HPA for several minutes, we can see CPU load reduced from 250% to 55% over the 13-14 minutes span.



Thus, we can say that we have successfully proved our highly available k3s cluster is capable of handling sudden spike in load due to high demand. HPA made sure that deployments in our k3s cluster are running without any crashes or major issues.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*