Requirements and System Configuration for Installing Kubernetes on Ubuntu The Kubernetes cluster consists of master nodes and worker nodes. Hardware configuration depends on your needs and the applications that you are planning to run in Docker containers. The minimum hardware requirements for installing Kubernetes on Ubuntu are:

• At least a 2-core x86/x64 CPU

• 2 GB of RAM or more (preferably 4 GB or more)

Ports that must be opened for installing Kubernetes on Ubuntu:

• TCP 443 - Kubernetes API Server

• TCP 10250 - Worker node Kubelet health check port

• TCP 30000-32767 - Default port range for providing external services

• UDP 8285 - UDP backend of Flannel overlay network

• UDP 8472 - VXLAN backend of Flannel overlay network

• TCP 2379-2380 - etcd server client API

We have installed Kubernetes on Ubuntu 18.04 LTS that is running on ESXi. The first explained Kubernetes deployment type is with a master node, and two worker nodes are used for the Kubernetes cluster. Please check the table below to view conditional numbers of nodes, node roles, hostnames and IP addresses of the machines used in the considered example.

Node role IP Address Host name Master 192.168.100.21 docker-atlassoft21 Worker 192.168.100.31 docker-atlassoft31 Worker 192.168.100.32 docker-atlassoft32

The IP address of the host machine: 14.99.27.198 The IP address of the virtual gateway for the NAT network (VMNet8): 192.168.100.1 The same Linux user exists on all Ubuntu machines: kubernetes-user VM configuration: 2CPU, 4GB RAM, 20-GB virtual disk

Deploying the Ubuntu VM Create a new VM named is docker-atlassoft21. Install Ubuntu 64-bit on the first machine, and set the host name and user name. VM name: docker-atlassoft21 Username: kubernetes-user

Now install VMware Tools from the ISO image provided with the VMware hypervisor or from Linux repositories (explained below).

$ sudo apt-get install open-vm-tools

Reboot the VM.

Configure the Ubuntu Machine Before Installing Kubernetes Some preparations must be made before installing Kubernetes on Ubuntu machines. First of all, you must configure the static IP address and the host name for any usual server.

Set the static IP address: Install Linux networking tools before setting the IP address.

$ sudo apt-get install net-tools

Type ifconfig to check the current IP address of the Ubuntu VM. Next install vim as the text editor.

$ sudo apt-get install vim

In the latest Ubuntu versions, network configuration is set in the yaml file. Open the network configuration yaml file in vim.

$ sudo vim /etc/netplan/01-network-manager-all.yaml

The default view of the configuration file is:

Edit this network configuration file as shown below:

network: version: 2

renderer: networkd ethernets:

ens33:

dhcp4: no addresses: [192.168.100.21/24] gateway4: 192.168.100.1 nameservers:

addresses: [192.168.100.1,8.8.8.8]

Save changes and exit.

$ sudo netplan try

Press ENTER to accept the new configuration.

Check whether your network configuration has been changed and try to ping, for example, google.com.

$ ifconfig

$ ping google.com

Disable a swap file: Using a swap file (swap partition) is not supported by Kubernetes and disabling swapiness is necessary to install Kubernetes on Ubuntu successfully.

Disable a swap file to prevent kubelet high CPU usage.

$ sudo swapoff -a

Edit /etc/fstab and comment the string by using the # character.

$ sudo nano /etc/fstab

#/swapfile none swap sw 0 0

Disable swap in sysctl.conf

$ sudo echo "vm.swappiness=0" | sudo tee --append /etc/sysctl.conf

Where 0 is the percent of swapiness. In this case swap can be used only if you are out of RAM (by default swap is used when more that 60% of RAM is full).

Apply configuration changes without reboot.

$ sudo sysctl -p

If the swap partition is not disabled, the kswapd0 process of Ubuntu Linux running Kubernetes can consume a large amount of CPU resources on the machine, causing applications to become unresponsive, and the system to hang. This happens when the operating system runs out of memory, and the old memory pages are moved to swap by a Linux kernel system process. For strange reasons, sometimes things go wrong, and a never-ending loop that consumes all CPU resources occurs. On the screenshot below, high CPU consumption by the kswapd0 process when Kubernetes is installed on Ubuntu. The load average value is excessively high.

If kswapd0 still overloads your CPU, run the following command to invalidate all memory cashes and stop kswapd0 (execute as root).

# echo 1 > /proc/sys/vm/drop\_caches

Shutdown the VM.

This partially configured virtual machine (docker-atlassoft21) is about to be used as a master node. Create two machines to be used as worker nodes. If you use physical machines, repeat the previous steps manually (or use automation tools such as Ansible for configuring multiple Linux machines simultaneously via SSH). As VMs are used in the current example, they can be cloned to save time during preparing environment for installing Kubernetes on Ubuntu machines.

Clone the VM Clone the first VM. And change the hostname to docker-atlassoft31

Check your current hostname.

$ hostnamectl

Change the hostname

$ sudo hostnamectl set-hostname docker-atlassoft31

Check that a new hostname is applied.

$ less /etc/hostname

Edit the hosts file.

$ sudo nano /etc/hosts

The contents of the host’s file must look like this:

127.0.0.1 localhost

127.0.1.1 docker-atlassoft31

Restart the machine.

Do the same thing to create the other host as docker-atlassoft32

Configuring IP Addresses and Host Names on VMs Change the IP address and hostname on docker-atlassoft31 and docker-atlassoft32 VMs (as shown above). Repeat the steps explained in the above sections to configure the static IP addresses and hostnames.

The IP address must be: 192.168.100.31 and 192.168.100.32; the hostnames must be docker- atlassoft31 and docker- atlassoft32 accordingly on worker nodes.

All machines must be configured for resolving hostnames of nodes to IP addresses. You can configure a DNS server or manually edit the hosts file on each machine. Let’s edit hosts.

Add the following strings to the hosts file on each machine (docker-atlassoft21, docker- atlassoft31, docker- atlassoft32).

$ sudo nano /etc/hosts

Add these lines to the hosts file:

192.168.100.21 docker-atlassoft21 192.168.100.31 docker- atlassoft31 192.168.100.32 docker- atlassoft32

Ping other hosts from each host to ensure that the hostnames are resolved:

$ ping docker-atlassoft21

$ ping docker- atlassoft31

$ ping docker- atlassoft32

Configure SSH Access on All Hosts (VMs) Configure SSH access on all hosts. Install the OpenSSH server by executing the commands on each machine.

$ sudo apt-get install openssh-server

Go to the home directory of kubernetes-user and generate the SSH key pair (a set of cryptographic keys which consists of a private key and public key). SSH key pairs can be used to access the remote Linux console via SSH without using passwords. The public key can be copied to a machine from which you need to connect remotely, while the private key is highly secret and must be stored on the machine to which you need to connect.

$ ssh-keygen

It is not necessary to enter the password for key generating (the password is optional). Copy the keys to other Ubuntu machines:

$ ssh-copy-id kubernetes-user@192.168.100.31

$ ssh-copy-id kubernetes-user@192.168.100.32

Enter the user password to confirm copying the keys.

Try to connect to the second machine (docker-atlassoft31) as kubernetes-user (that is a regular user).

$ ssh 'kubernetes-user@192.168.100.31'

Then test connection to the third machine (docker-atlassoft32).

$ ssh 'kubernetes-user@192.168.100.32'

You will see the name of the remote machine in the command prompt of your console after successful connection.

Press Ctrl + D to exit from the remote console.

Copy the key for connecting via SSH as a root user Root privileges are needed in Kubernetes, let’s create keys for configuring SSH access for root. Execute the following commands on all machines (docker-atlassoft21, docker- atlassoft31 and docker- atlassoft32) that need to be accessed via SSH as a root user.

$ sudo -i

Edit the SSH server configuration file.

# nano /etc/ssh/sshd\_config

Add/edit the following string to this file.

PermitRootLogin yes

Restart the SSH server daemon.

# /etc/init.d/ssh stop # /etc/init.d/ssh start

Set the root password (password for the root user).

# passcode

$ cd /home/kubernetes-user/

$ sudo ssh-keygen -t rsa

Copy the public key to be able to login remotely via SSH as root (the key is stored in the home directory of the regular user since the previous command was run from that directory).

$ sudo ssh-copy-id -i /home/kubernetes-user/.ssh/id\_rsa.pub 127.0.0.1

If key is saved into the home directory of the root user, copy the key with this command:

# ssh-copy-id -i /root/.ssh/id\_rsa.pub 127.0.0.1

Confirm this operation and enter your password.

Repeat the action, copying the key from each machine to other machines. For instance, on the docker- atlassoft21 machine execute:

# ssh-copy-id -i /root/.ssh/id\_rsa.pub 192.168.101.31 # ssh-copy-id -i /root/.ssh/id\_rsa.pub 192.168.101.32

Make the public key authorized.

# cat /root/.ssh/id\_rsa.pub >> /root/.ssh/authorized\_keys

Verify whether you can log in as root via SSH on the local machine.

$ sudo ssh root@127.0.0.1

Try to connect from/to the remote machine without entering a password.

$ sudo ssh root@192.168.101.21

$ sudo ssh root@192.168.101.31

$ sudo ssh root@192.168.101.32

Docker Installation Install Docker on all machines. Execute commands shown below on docker-atlassoft21, docker- atlassoft31, docker-atlassoft32. Simply install Docker by using the usual command:

$ sudo apt-get install -y docker.io

Still, in this case, the version of Docker used may not be the latest. Let’s remedy that by installing the latest version of Docker. First, install the required packages.

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

Add the GPG key for the official Docker repository to the Ubuntu system:

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

The console output must be OK.

Add the official Docker repository to the apt package manager:

$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

Update the database of the package manager after manual changes have been made by the previous command.

$ sudo apt-get update

Check the version of the Docker package available in the official repository.

$ apt-cache policy docker-ce

Install Docker.

$ sudo apt-get install docker-ce

You can check the version of Docker after installation.

$ docker --version

Start Docker and make its daemon loadable automatically on system startup.

$ sudo systemctl start docker $ sudo systemctl enable docker

Once installed Docker on all machines, you can go directly to the step of installing Kubernetes on Ubuntu.

Installing Kubernetes on Ubuntu and Cluster Initialization Run the commands as root on all machines to be included into the Kubernetes cluster.

$ sudo -i

Add the GPG key for the official Docker repository to your Ubuntu system:

# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

Add the official Kubernetes repository to the database of available package repositories for your apt package manager.

# cat <<EOF >/etc/apt/sources.list.d/kubernetes.list > deb http://apt.kubernetes.io/ kubernetes-xenial main > EOF

As an alternative, you can add the repository with this command:

# echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' | sudo tee /etc/apt/sources.list.d/kubernetes.list

Where tee is a tool that reads the standard input and writes the input data to standard output and defined files. Update the package list of available repositories on your Ubuntu system.

# apt-get update

Installing kubectl, kubeadm and kubectl is crucial to install Kubernetes on Ubuntu.

# apt-get install -y kubelet kubeadm kubectl

Install keepalived.

# apt-get install keepalived # systemctl enable keepalived && systemctl start keepalived

Verify whether the value is 1 for correct functioning of Kubernetes installed on Ubuntu.

# sysctl net.bridge.bridge-nf-call-iptables

In order to set this value to 1 run the command (if in case value is not 1):

sysctl net.bridge.bridge-nf-call-iptables=1

Edit the kubeadm configuration file.

# nano /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

Add the string after the existing Environment string:

Environment=”cgroup-driver=systemd/cgroup-driver=cgroupfs”

(Please check yellow string, which has been added)

On the master node (docker-atlassoft21) run the command to initialize the Kubernetes cluster on Ubuntu.

# kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=192.168.100.21

Where: --pod-network-cidr is required by the Flannel driver. CIDR (Classless Inter-Domain Routing) defines the address of your overlay network (such as Flannel) that will be configured later. The network mask also defines how many pods can run per node. The CIDR network address and the network address used for Flannel must be the same. --apiserver-advertise-address=192.168.100.21 defines the IP address that will be advertised by Kubernetes as its API server.

Read the output and save commands displayed at the end of the text. This is an important point. The generated token is required for adding worker nodes to the Kubernetes cluster. Run the following commands as a user that has run kubeadm init. In this case, the commands are executed as root.

# mkdir -p $HOME/.kube

# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config # sudo chown $(id -u):$(id -g) $HOME/.kube/config

If you don’t run these commands, Kubernetes shall return the error: The connection to the server localhost:8080 was refused - did you specify the right host or port? Kubernetes doesn’t copy this config file to the user directory automatically. You should perform this operation manually.

# kubectl get nodes

You can see one master node that has the NotReady status in the Kubernetes cluster which is being installed on Ubuntu. This is due to the fact that the overlay network has not been configured. Configure Flannel in order to fix the NotReady status of the Kubernetes master node. Create the directory to store yaml files for Docker and Kunernetes, for example /home/kubernetes- user/kubernetes/ YAML offers you greater convenience when creating pods and deployments in Kubernetes. You can define all parameters of containers which must be deployed in the YAML configuration file instead of running each command manually in Linux console. YAML files are also referred to as manifest files in the context of Kubernetes. Create the yaml configuration file (kube-flannel.yml) with the following content:

--- apiVersion: policy/v1beta1 kind: PodSecurityPolicy metadata:

name: psp.flannel.unprivileged annotations:

seccomp.security.alpha.kubernetes.io/allowedProfileNames: docker/default

seccomp.security.alpha.kubernetes.io/defaultProfileName: docker/default apparmor.security.beta.kubernetes.io/allowedProfileNames: runtime/default

apparmor.security.beta.kubernetes.io/defaultProfileName: runtime/default spec:

privileged: false volumes:

- configMap

- secret - emptyDir - hostPath allowedHostPaths:

- pathPrefix: "/etc/cni/net.d" - pathPrefix: "/etc/kube-flannel" - pathPrefix: "/run/flannel" readOnlyRootFilesystem: false # Users and groups runAsUser:

rule: RunAsAny supplementalGroups:

rule: RunAsAny fsGroup:

rule: RunAsAny # Privilege Escalation allowPrivilegeEscalation: false defaultAllowPrivilegeEscalation: false # Capabilities allowedCapabilities: ['NET\_ADMIN'] defaultAddCapabilities: [] requiredDropCapabilities: [] # Host namespaces hostPID: false hostIPC: false hostNetwork: true hostPorts: - min: 0

max: 65535 # SELinux seLinux:

# SELinux is unused in CaaSP rule: 'RunAsAny' --- kind: ClusterRole apiVersion: rbac.authorization.k8s.io/v1beta1 metadata:

name: flannel rules:

- apiGroups: ['extensions']

resources: ['podsecuritypolicies'] verbs: ['use'] resourceNames: ['psp.flannel.unprivileged'] - apiGroups: - "" resources: - pods verbs:

- get - apiGroups: - "" resources:

- nodes verbs:

- list - watch - apiGroups:

- "" resources:

- nodes/status verbs:

- patch --- kind: ClusterRoleBinding apiVersion: rbac.authorization.k8s.io/v1beta1 metadata:

name: flannel roleRef:

apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: flannel subjects: - kind: ServiceAccount

name: flannel namespace: kube-system --- apiVersion: v1 kind: ServiceAccount metadata:

name: flannel namespace: kube-system --- kind: ConfigMap apiVersion: v1 metadata:

name: kube-flannel-cfg namespace: kube-system labels:

tier: node app: flannel data:

cni-conf.json: |

{

"name": "cbr0", "cniVersion": "0.3.1", "plugins": [

{

"type": "flannel", "delegate": {

"hairpinMode": true, "isDefaultGateway": true } }, {

"type": "portmap", "capabilities": {

"portMappings": true } } ] } net-conf.json: |

{

"Network": "10.244.0.0/16",

"Backend": {

"Type": "vxlan" } } --- apiVersion: apps/v1 kind: DaemonSet metadata:

name: kube-flannel-ds-amd64 namespace: kube-system labels:

tier: node app: flannel spec:

selector:

matchLabels:

app: flannel template:

metadata: labels:

tier: node app: flannel spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: beta.kubernetes.io/os

operator: In values:

- linux - key: beta.kubernetes.io/arch

operator: In values:

- amd64 hostNetwork: true tolerations: - operator: Exists

effect: NoSchedule serviceAccountName: flannel initContainers: - name: install-cni

image: quay.io/coreos/flannel:v0.11.0-amd64 command: - cp args: - -f - /etc/kube-flannel/cni-conf.json - /etc/cni/net.d/10-flannel.conflist volumeMounts: - name: cni

mountPath: /etc/cni/net.d - name: flannel-cfg

mountPath: /etc/kube-flannel/ containers: - name: kube-flannel

image: quay.io/coreos/flannel:v0.11.0-amd64 command: - /opt/bin/flanneld args: - --ip-masq - --kube-subnet-mgr resources:

requests:

cpu: "100m" memory: "50Mi" limits:

cpu: "100m" memory: "50Mi" securityContext:

privileged: false capabilities:

add: ["NET\_ADMIN"] env: - name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name - name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace volumeMounts: - name: run

mountPath: /run/flannel - name: flannel-cfg

mountPath: /etc/kube-flannel/ volumes:

- name: run hostPath:

path: /run/flannel - name: cni hostPath:

path: /etc/cni/net.d - name: flannel-cfg

configMap:

name: kube-flannel-cfg --- apiVersion: apps/v1 kind: DaemonSet metadata:

name: kube-flannel-ds-arm64 namespace: kube-system labels:

tier: node app: flannel spec:

selector:

matchLabels:

app: flannel template:

metadata: labels:

tier: node app: flannel spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: beta.kubernetes.io/os

operator: In values:

- linux - key: beta.kubernetes.io/arch

operator: In values:

- arm64 hostNetwork: true tolerations: - operator: Exists

effect: NoSchedule serviceAccountName: flannel initContainers: - name: install-cni

image: quay.io/coreos/flannel:v0.11.0-arm64 command: - cp args: - -f - /etc/kube-flannel/cni-conf.json - /etc/cni/net.d/10-flannel.conflist volumeMounts: - name: cni

mountPath: /etc/cni/net.d - name: flannel-cfg

mountPath: /etc/kube-flannel/ containers: - name: kube-flannel

image: quay.io/coreos/flannel:v0.11.0-arm64 command: - /opt/bin/flanneld args: - --ip-masq - --kube-subnet-mgr resources:

requests:

cpu: "100m" memory: "50Mi" limits:

cpu: "100m" memory: "50Mi" securityContext:

privileged: false capabilities:

add: ["NET\_ADMIN"] env: - name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name - name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace volumeMounts: - name: run

mountPath: /run/flannel - name: flannel-cfg

mountPath: /etc/kube-flannel/ volumes:

- name: run hostPath:

path: /run/flannel - name: cni hostPath:

path: /etc/cni/net.d - name: flannel-cfg

configMap:

name: kube-flannel-cfg --- apiVersion: apps/v1 kind: DaemonSet metadata:

name: kube-flannel-ds-arm namespace: kube-system labels:

tier: node app: flannel spec:

selector:

matchLabels:

app: flannel template:

metadata: labels:

tier: node app: flannel spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: beta.kubernetes.io/os

operator: In values:

- linux - key: beta.kubernetes.io/arch

operator: In values: - arm hostNetwork: true tolerations: - operator: Exists

effect: NoSchedule

serviceAccountName: flannel initContainers: - name: install-cni

image: quay.io/coreos/flannel:v0.11.0-arm command: - cp args: - -f - /etc/kube-flannel/cni-conf.json - /etc/cni/net.d/10-flannel.conflist volumeMounts: - name: cni

mountPath: /etc/cni/net.d - name: flannel-cfg

mountPath: /etc/kube-flannel/ containers: - name: kube-flannel

image: quay.io/coreos/flannel:v0.11.0-arm command: - /opt/bin/flanneld args: - --ip-masq - --kube-subnet-mgr resources:

requests:

cpu: "100m" memory: "50Mi" limits:

cpu: "100m" memory: "50Mi" securityContext:

privileged: false capabilities:

add: ["NET\_ADMIN"] env: - name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name - name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace volumeMounts: - name: run

mountPath: /run/flannel - name: flannel-cfg

mountPath: /etc/kube-flannel/ volumes:

- name: run hostPath:

path: /run/flannel - name: cni hostPath:

path: /etc/cni/net.d - name: flannel-cfg

configMap:

name: kube-flannel-cfg --- apiVersion: apps/v1 kind: DaemonSet metadata:

name: kube-flannel-ds-ppc64le namespace: kube-system labels:

tier: node app: flannel spec:

selector:

matchLabels:

app: flannel template:

metadata: labels:

tier: node app: flannel spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: beta.kubernetes.io/os

operator: In values:

- linux - key: beta.kubernetes.io/arch

operator: In values:

- ppc64le hostNetwork: true tolerations: - operator: Exists

effect: NoSchedule serviceAccountName: flannel initContainers: - name: install-cni

image: quay.io/coreos/flannel:v0.11.0-ppc64le command: - cp args: - -f - /etc/kube-flannel/cni-conf.json - /etc/cni/net.d/10-flannel.conflist volumeMounts: - name: cni

mountPath: /etc/cni/net.d - name: flannel-cfg

mountPath: /etc/kube-flannel/ containers: - name: kube-flannel

image: quay.io/coreos/flannel:v0.11.0-ppc64le command: - /opt/bin/flanneld

args: - --ip-masq - --kube-subnet-mgr resources:

requests:

cpu: "100m" memory: "50Mi" limits:

cpu: "100m" memory: "50Mi" securityContext:

privileged: false capabilities:

add: ["NET\_ADMIN"] env: - name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name - name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace volumeMounts: - name: run

mountPath: /run/flannel - name: flannel-cfg

mountPath: /etc/kube-flannel/ volumes:

- name: run hostPath:

path: /run/flannel - name: cni hostPath:

path: /etc/cni/net.d - name: flannel-cfg

configMap:

name: kube-flannel-cfg --- apiVersion: apps/v1 kind: DaemonSet metadata:

name: kube-flannel-ds-s390x namespace: kube-system labels:

tier: node app: flannel spec:

selector:

matchLabels:

app: flannel template:

metadata: labels:

tier: node app: flannel spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: beta.kubernetes.io/os

operator: In values:

- linux - key: beta.kubernetes.io/arch

operator: In values:

- s390x hostNetwork: true tolerations: - operator: Exists

effect: NoSchedule serviceAccountName: flannel initContainers: - name: install-cni

image: quay.io/coreos/flannel:v0.11.0-s390x command: - cp args: - -f - /etc/kube-flannel/cni-conf.json - /etc/cni/net.d/10-flannel.conflist volumeMounts: - name: cni

mountPath: /etc/cni/net.d - name: flannel-cfg

mountPath: /etc/kube-flannel/ containers: - name: kube-flannel

image: quay.io/coreos/flannel:v0.11.0-s390x command: - /opt/bin/flanneld args: - --ip-masq - --kube-subnet-mgr resources:

requests:

cpu: "100m" memory: "50Mi" limits:

cpu: "100m" memory: "50Mi" securityContext:

privileged: false capabilities:

add: ["NET\_ADMIN"] env: - name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name - name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace volumeMounts: - name: run

mountPath: /run/flannel - name: flannel-cfg

mountPath: /etc/kube-flannel/ volumes:

- name: run hostPath:

path: /run/flannel - name: cni hostPath:

path: /etc/cni/net.d - name: flannel-cfg

configMap:

name: kube-flannel-cfg

Run the command

# kubectl apply -f /var/tmp/kube-flannel.yml

As an alternative, you can find prepared free examples of YAML deployment configurations for Kubernetes on GitHub.

# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Please note: Support for extensions/v1beta1 was removed in v1.16 onwards. So, for this implementation we have used the particular kube-funnel.yml

Check the nodes added to the Kubernetes cluster you are deploying on Ubuntu:

# kubectl get nodes

The status of the master node now is Ready.

Ensure that Flannel has been set up correctly:

# kubectl get pods --all-namespaces

You can see that the Flannel pod is running. This pod consists of two containers – the Flannel daemon, and initContainer used to deploy the CNI configuration to a location readable for Kubernetes. Sometimes when you install Kubernetes on Ubuntu, the following error may occur: Unable to connect to the server: net/http: TLS handshake timeout.

How can you fix this issue? Wait for a few seconds and try again -- this is often enough. Namespaces are logical entities in the Kubernetes cluster that represent cluster resources and can be considered virtual clusters. One physical cluster can be logically divided to multiple virtual clusters. The default Kubernetes namespaces are Default, Kube-public, and Kube-system. You can get the list of namespaces:

# kubectl get namespaces

As you recall, the basic deployment unit in Kubernetes is a pod which is a collection of containers that share network and mount namespace. All containers of the pod are scheduled on the same Kubernetes node. Check the available pods:

# kubectl -n kube-system get pods

If you wish to reset/stop the cluster, run:

# kubeadm reset

Everything is OK on the master node. This means that now you can continue to install Kubernetes on Ubuntu and switch to adding worker nodes to the cluster. On the worker nodes (docker-atlassoft31, docker-atlassoft32) run the command:

# kubeadm join 192.168.100.21:6443 --token yjmjtq.oo7ed3fc7cs2kbpc \ --discovery-token-ca-cert- hash sha256:40743549e205152aed3e1220e3fa670e7b7bc086a52dd18c51fcce65b37fb455

The token and hash were noted after cluster initialization with the kubeadm init command, as you may recall.

On the master node, check the cluster status again.

# kubectl get nodes

Now you can see one master and two worker nodes in the Kubernetes cluster running on Ubuntu machines. You can check Kubernetes configuration:

# kubectl cluster-info

Deploying a Pod in Kubernetes Now you can deploy a pod with containers in Kubernetes cluster. As you remember, containers are included in pods in Kubernetes. If you use yaml files, create a directory to store that files for more convenience. Go to that directory and run the commands like kubectl apply -f test.yaml Such a directory has been already created when configuring Flannel - /home/kubernetes- user/kubernetes/ It’s time to deploy a new pod. First you need to create a deployment. Deployment is a controller concept used for providing declarative updates to pods and replica sets. You can create deployment with a single command or by using yaml files.

Example 1 – deploying MySQL: Let’s create a yaml file in this example. The name of the file is mysql-deployment.yaml See the file configuration below:

apiVersion: v1 kind: Service metadata:

name: mysql spec:

ports: - port: 3306 selector:

app: mysql clusterIP: None --- apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2 kind: Deployment metadata:

name: mysql spec:

selector:

matchLabels: app: mysql strategy:

type: Recreate template:

metadata: labels:

app: mysql spec:

containers:

- image: mysql:5.6

name: mysql env: # Use secret in real usage - name: MYSQL\_ROOT\_PASSWORD

value: password ports: - containerPort: 3306

name: mysql volumeMounts: - name: mysql-persistent-storage

mountPath: /var/lib/mysql volumes: - name: mysql-persistent-storage

persistentVolumeClaim:

claimName: mysql-pv-claim

# nano /var/tmp/mysql-deployment.yaml

There are two popular approaches for managing resources with kubectl. What is the difference between kubectl create and kubectl apply? When using kubectl create, you tell Kubernetes what you want to create, replace or delete; this command overwrites all changes. Alternatively, kubectl apply makes incremental changes and this command can be used to save changes applied to a live object.

Create a deployment:

# kubectl apply -f /var/tmp/mysql-deployment.yaml

Kubernetes can display information about your deployment.

# kubectl describe deployment mysql

Check pods:

# kubectl get po

or # kubectl get pods

or # kubectl get pods -l app=mysql

If you see the pending status for the pod, it means there are not enough computing resources. Try to add some CPU and memory capacity to fix the pending status of the pod in Kubernetes.

Example 2 - Deploying nginx: Let’s deploy nginx by using another method without yaml files.

Create a deployment.

# kubectl create deployment nginx --image=nginx

Check that the deployment has been created.

# kubectl get deployments

Create a service.

# kubectl create service nodeport nginx --tcp=80:80

A service can be created by using the following service types – ClusterIP, NodePort, LoadBalance, and ExternalName. If the NodePort type is used, then a random port from the 30000-32767 range is allocated for accessing the provided services. Traffic that is sent to this port is forwarded to the necessary service.

Check that the service is created and is listening on the defined port.

# kubectl get svc

Remember the number of port (31344 in this case).

Check whether the service is deployed and available (the command is run on the master node in this example). Use the hostname of the node and port you have remembered from the previous step.

# curl docker-atlassoft31:31344

You can also check that the service is accessible in the browser of any node. In the address bar of the web browser try to visit the pages:

http:// 10.104.159.153

or http://docker-atlassoft31:31344

or http://docker-atlassoft32:31344

If everything is OK, you will see the nginx welcome page.

It is also possible to visit the nginx test page from any machine that has access to the network to which Kubernetes nodes are connected (192.168.100.0/24) in this case. For example, you can visit the web pages with your browser:

http://192.168.100.21:31344/

http://192.168.100.31:31344/

http://192.168.100.32:31344/

Set Up the Web Interface for Monitoring Kubernetes Installing Kubernetes on Ubuntu is almost complete, but you can also install Kubernetes dashboard for more convenience. Kubernetes dashboard is a web interface for Kubernetes management and monitoring. In order to install the dashboard, create the kubernetes-dashboard.yaml file, much like you have done previously before executing the commands.

# Copyright 2017 The Kubernetes Authors. # # Licensed under the Apache License, Version 2.0 (the "License"); # you may not use this file except in compliance with the License. # You may obtain a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS, # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. # See the License for the specific language governing permissions and # limitations under the License.

apiVersion: v1 kind: Namespace metadata:

name: kubernetes-dashboard

---

apiVersion: v1 kind: ServiceAccount metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard namespace: kubernetes-dashboard

---

kind: Service apiVersion: v1 metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard namespace: kubernetes-dashboard spec:

ports:

- port: 443

targetPort: 8443 selector:

k8s-app: kubernetes-dashboard

---

apiVersion: v1 kind: Secret metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard-certs namespace: kubernetes-dashboard type: Opaque

---

apiVersion: v1 kind: Secret metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard-csrf namespace: kubernetes-dashboard type: Opaque data:

csrf: ""

---

apiVersion: v1 kind: Secret metadata:

labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard-key-holder namespace: kubernetes-dashboard type: Opaque

---

kind: ConfigMap apiVersion: v1 metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard-settings namespace: kubernetes-dashboard

---

kind: Role apiVersion: rbac.authorization.k8s.io/v1 metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard namespace: kubernetes-dashboard rules:

# Allow Dashboard to get, update and delete Dashboard exclusive secrets. - apiGroups: [""]

resources: ["secrets"] resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes- dashboard-certs", "kubernetes-dashboard-csrf"]

verbs: ["get", "update", "delete"] # Allow Dashboard to get and update 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"] resourceNames: ["kubernetes-dashboard-settings"] verbs: ["get", "update"] # Allow Dashboard to get metrics. - apiGroups: [""]

resources: ["services"] resourceNames: ["heapster", "dashboard-metrics-scraper"] verbs: ["proxy"] - apiGroups: [""]

resources: ["services/proxy"] resourceNames: ["heapster", "http:heapster:", "https:heapster:", "dashboard-metrics-scraper", "http:dashboard-metrics-scraper"]

verbs: ["get"]

---

kind: ClusterRole apiVersion: rbac.authorization.k8s.io/v1 metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard

rules:

# Allow Metrics Scraper to get metrics from the Metrics server - apiGroups: ["metrics.k8s.io"]

resources: ["pods", "nodes"] verbs: ["get", "list", "watch"]

---

apiVersion: rbac.authorization.k8s.io/v1 kind: RoleBinding metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard namespace: kubernetes-dashboard roleRef:

apiGroup: rbac.authorization.k8s.io kind: Role name: kubernetes-dashboard subjects:

- kind: ServiceAccount

name: kubernetes-dashboard namespace: kubernetes-dashboard

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata:

name: kubernetes-dashboard roleRef:

apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: kubernetes-dashboard subjects:

- kind: ServiceAccount

name: kubernetes-dashboard namespace: kubernetes-dashboard

---

kind: Deployment apiVersion: apps/v1 metadata: labels:

k8s-app: kubernetes-dashboard name: kubernetes-dashboard namespace: kubernetes-dashboard spec:

replicas: 1 revisionHistoryLimit: 10 selector:

matchLabels:

k8s-app: kubernetes-dashboard template:

metadata: labels:

k8s-app: kubernetes-dashboard spec:

containers:

- name: kubernetes-dashboard

image: kubernetesui/dashboard:v2.0.0-beta8 imagePullPolicy: Always ports:

- containerPort: 8443

protocol: TCP args:

- --auto-generate-certificates - --namespace=kubernetes-dashboard # Uncomment the following line to manually specify Kubernetes API server Host

# If not specified, Dashboard will attempt to auto discover the API server and connect

# to it. Uncomment only if the default does not work. # - --apiserver-host=http://my-address:port volumeMounts:

- name: kubernetes-dashboard-certs

mountPath: /certs # Create on-disk volume to store exec logs - mountPath: /tmp

name: tmp-volume livenessProbe:

httpGet:

scheme: HTTPS path: / port: 8443 initialDelaySeconds: 30 timeoutSeconds: 30 securityContext:

allowPrivilegeEscalation: false readOnlyRootFilesystem: true runAsUser: 1001 runAsGroup: 2001 volumes:

- name: kubernetes-dashboard-certs

secret:

secretName: kubernetes-dashboard-certs - name: tmp-volume

emptyDir: {} serviceAccountName: kubernetes-dashboard nodeSelector:

"beta.kubernetes.io/os": linux # Comment the following tolerations if Dashboard must not be deployed on master

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule

---

kind: Service apiVersion: v1 metadata: labels:

k8s-app: dashboard-metrics-scraper name: dashboard-metrics-scraper namespace: kubernetes-dashboard spec:

ports:

- port: 8000

targetPort: 8000 selector:

k8s-app: dashboard-metrics-scraper

---

kind: Deployment apiVersion: apps/v1 metadata: labels:

k8s-app: dashboard-metrics-scraper name: dashboard-metrics-scraper namespace: kubernetes-dashboard spec:

replicas: 1 revisionHistoryLimit: 10 selector:

matchLabels:

k8s-app: dashboard-metrics-scraper template:

metadata: labels:

k8s-app: dashboard-metrics-scraper annotations:

seccomp.security.alpha.kubernetes.io/pod: 'runtime/default' spec:

containers:

- name: dashboard-metrics-scraper

image: kubernetesui/metrics-scraper:v1.0.1 ports:

- containerPort: 8000

protocol: TCP livenessProbe:

httpGet:

scheme: HTTP path: / port: 8000 initialDelaySeconds: 30 timeoutSeconds: 30 volumeMounts: - mountPath: /tmp

name: tmp-volume securityContext:

allowPrivilegeEscalation: false readOnlyRootFilesystem: true runAsUser: 1001 runAsGroup: 2001 serviceAccountName: kubernetes-dashboard nodeSelector:

"beta.kubernetes.io/os": linux

# Comment the following tolerations if Dashboard must not be deployed on master

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule volumes:

- name: tmp-volume

emptyDir: {}

For this we have used the file hosted on: https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0- beta8/aio/deploy/recommended.yaml

# kubectl create -f ./kubernetes-dashboard.yaml

# kubectl apply -f ./kubernetes-dashboard.yaml

Check pods.

# kubectl get pods -o wide --all-namespaces

Start the proxy to the Kubernetes API server.

# kubectl proxy

(Please check yellow marked highlighted commands)

To enter the next commands in the console, please open another console window. Otherwise, the process would be terminated.

In your web browser on the master node, go to the page:

http://localhost:8001

You can see the test page.

Enter the full address in the address bar of the web browser.

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes- dashboard:/proxy/#/login

Please note: Unlike version 1.10.x in v2.0.0-xxx the dashboard url has been changed.

Create a dashboard with a service account by executing the commands in the new console window.

# kubectl create serviceaccount dashboard -n default

# kubectl create clusterrolebinding dashboard-admin -n default \

--clusterrole=cluster-admin \

--serviceaccount=default:dashboard

# kubectl get secret $(kubectl get serviceaccount dashboard -o jsonpath="{.secrets[0].name}") -o jsonpath="{.data.token}" | base64 --decode

The generated token is:

Copy the generated token and paste it in the token section of the web interface to log into the dashboard.

On the screenshot below, you can see the web interface of Kubernetes dashboard. You can see the status of nodes, deployments, and pods, as well as check roles, storage classes, and other components.