Arman Shafiei
Posted on Apr 4 • Updated on Aug 24

Install RKE2 with Cilium and Metallb

#rancher #metallb #cilium #kubernetes

Introduction

In today's rapidly evolving technological landscape, container orchestration has become a critical component for managing complex applications. Kubernetes is an open-source platform that revolutionizes how we deploy, scale, and manage containerized workloads. Its advantages are manifold: it provides seamless scalability, high availability, and declarative configuration. With a thriving ecosystem and a robust community, Kubernetes empowers developers to focus on innovation rather than infrastructure intricacies.

But what about simplicity and security? That's where Rancher RKE2 steps in. This lightweight Kubernetes distribution offers a streamlined experience without compromising on safety. With a single binary installation, RKE2 simplifies setup and maintenance. Its built-in high availability, enhanced security

the following specifications:

- **Control-Plane-1** => IP: 192.168.100.11 , OS: Debian 12 , Hostname: kuber-master-1
- **Control-Plane-2** => IP: 192.168.100.12 , OS: Debian 12 , Hostname: kuber-master-2
- **Control-Plane-3** => IP: 192.168.100.13 , OS: Debian 12 , Hostname: kuber-master-3
- **Worker-1** => IP: 192.168.100.14 , OS: Debian 12 , Hostname: kuber-worker-1
- **Worker-2** => IP: 192.168.100.15 , OS: Debian 12 , Hostname: kuber-worker-2
- **Worker-3** => IP: 192.168.100.16 , OS: Debian 12 , Hostname: kuber-worker-3
- **L4 Load Balancer** => ip: 192.168.100.100

Arman Shafiei
Follow
A DevOps engineer, Linux and open source enthusiastic.
LOCATION
Tehran, Iran
EDUCATION
Bachelor's degree from Qazvin Islamic Azad University
WORK
DevOps Engineer at AioCom
JOINED
Apr 1, 2024

More from Arman Shafiei
Deploy Nginx Load Balancer for Rancher
#rancher #loadbalancer #nginx
Write Helm chart from scratch
#kubernetes #helm

NOTE: We've assumed that there is an external load balancer, which could be a cloud load balancer or an on-premise node. The load balancer is also accessible with the following domains:

- rancher.arman-projects.com
- kubernetes.arman-projects.com
- rke2.arman-projects.com
- k8s.arman-projects.com

NOTE: In this article, we use the terms server node which refers to the control-plane node, and agent node which refers to the worker node. We use these terms interchangeably.

Operating System Pre-requisites

The very basic thing to do is to update the servers.

```
apt-get update && apt-get upgrade -y
```

next reboot the server.

```
reboot
```

Networking Pre-requisites

First of all, we must enable 2 kernel modules.
Run the following:

```
modprobe br_netfilter
modprobe overlay
cat <<EOF | tee /etc/modules-load.d/k8s.conf
br_netfilter
overlay
EOF
```

Now configure sysctl to prepare OS for Kubernetes and of course for security purposes.

```
cat <<EOF | tee /etc/sysctl.conf
net.ipv4.ip_forward=1
net.ipv4.conf.all.send_redirects=0
net.ipv4.conf.default.send_redirects=0
net.ipv4.conf.default.accept_source_route=0
net.ipv4.conf.all.accept_redirects=0
net.ipv4.conf.default.accept_redirects=0
net.ipv4.conf.all.log_martians=1
net.ipv4.conf.default.log_martians=1
net.ipv4.conf.all.rp_filter=1
net.ipv4.conf.default.rp_filter=1
net.ipv6.conf.all.accept_ra=0
net.ipv6.conf.default.accept_ra=0
net.ipv6.conf.all.accept_redirects=0
net.ipv6.conf.default.accept_redirects=0
kernel.keys.root_maxbytes=25000000
kernel.keys.root_maxkeys=1000000
kernel.panic=10
kernel.panic_on_oops=1
vm.overcommit_memory=1
vm.panic_on_oom=0
net.ipv4.ip_local_reserved_ports=30000-32767
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-arptables=1
net.bridge.bridge-nf-call-ip6tables=1
EOF
```

After sysctl is configured,
For changes to take effect run sysctl command below:

```
sysctl --system
```

Firewall Pre-requisites

There are some ports used by rke2 and Kubernetes services that must be opened between nodes. The list of ports can be found at "https://docs.rke2.io/install/requirements".
For simplicity, we've allowed all traffic between cluster nodes and just allowed some ports access only through the Load balancer.
First, flush all nftables rules.

```
nft flush ruleset
```

Then add these contents to the nftables default config file.

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
  chain input {
    type filter hook input priority filter; policy accep
    tcp dport 22 accept;
    ct state established,related accept;
    iifname "lo" accept;
    ip protocol icmp accept;
    ip daddr 8.8.8.8 tcp dport 53 accept;
    ip daddr 8.8.8.8 udp dport 53 accept;
    ip daddr 8.8.4.4 tcp dport 53 accept;
    ip daddr 8.8.4.4 udp dport 53 accept;
    ip daddr 1.1.1.1 tcp dport 53 accept;
    ip daddr 1.1.1.1 udp dport 53 accept;
    ip saddr 192.168.100.11 accept;
    ip saddr 192.168.100.12 accept;
    ip saddr 192.168.100.13 accept;
    ip saddr 192.168.100.14 accept;
    ip saddr 192.168.100.15 accept;
    ip saddr 192.168.100.16 accept;
    ip saddr 192.168.100.100 tcp dport {9345,6443,443,80
    ip saddr 192.168.100.100 udp dport {9345,6443,443,80
    counter packets 0 bytes 0 drop;
  }

  chain forward {
    type filter hook forward priority filter; policy acc
  }

  chain output {
    type filter hook output priority filter; policy acce
  }
}
```

In the firewall config above, all traffic from cluster nodes is allowed to reach each other and just 4 ports are supposed to access the cluster through the Load balancer.
NOTE: The ssh port is 22 and it's benn opened from all sources.
Enable and restart the nftables service.

```
systemctl enable nftables
systemctl restart nftables
```

Install Rancher rke2

Till now, we've prepared our machines. It's time to deploy rke2.
The subsequent commands will configure and install Rancher rke2.

Control Plane 1

Set up Rancher configs

Create Rancher configuration directory.

```
mkdir -p /etc/rancher/rke2/
```

We need to change the default options and arguments of rke2. In order to do that, create a file named **config.yaml** and put the below lines in it.

```
write-kubeconfig-mode: "0644"
advertise-address: 192.168.100.100
tls-san:
  - 192.168.100.100
  - rancher.arman-projects.com
  - kubernetes.arman-projects.com
  - rke2.arman-projects.com
  - k8s.arman-projects.com
cni: none
cluster-cidr: 10.100.0.0/16
service-cidr: 10.110.0.0/16
cluster-dns: 10.110.0.10
cluster-domain: arman-projects.com
etcd-arg: "--quota-backend-bytes 2048000000"
etcd-snapshot-schedule-cron: "0 3 * * *"
etcd-snapshot-retention: 10
disable:
  - rke2-ingress-nginx
disable-kube-proxy: true
kube-apiserver-arg:
  - '--default-not-ready-toleration-seconds=30'
  - '--default-unreachable-toleration-seconds=30'
kube-controller-manager-arg:
  - '--node-monitor-period=4s'
kubelet-arg:
  - '--node-status-update-frequency=4s'
  - '--max-pods=100'
egress-selector-mode: disabled
protect-kernel-defaults: true
```

Let's explain the above options:

- **write-kubeconfig-mode**: The permission of the generated kubeconfig file.
- **advertise-address**: Kubernetes API server address that all nodes must connect to.
- **tls-san**: Valid addresses for Kubernetes client certificate. kubectl trusts these addresses and any others are not trusted.
- **cni**: The CNI plugin that must get installed. Here we put none which indicates no CNI should be installed.
- **cluster-cidr**: The CIDR used in pods.
- **service-cidr**: The CIDR used in services.
- **cluster-dns**: Coredns service address.
- **cluster-domain**: The Kubernetes cluster domain. The default value is cluster.local
- **etcd-arg**: etcd database arguments. Here we've just increased the default etcd allowed memory usage.
- **etcd-snapshot-schedule-cron**: Specifies when to perform an etcd snapshot.
- **etcd-snapshot-retention**: Specifies how many snapshots will be kept.
- **disable**: instructs rke2 to not deploy the specified add-ons. Here we've disabled the nginx ingress add-on.
- **disable-kube-proxy**: Whether or not to use kube-proxy for pod networking. Here we have disabled the kube proxy so as to use Cilium instead.
- **kube-apiserver-arg**: Specifies kube api server arguments. Here we've set default-not-ready-toleration-seconds and default-unreachable-toleration-seconds to 30 seconds. The default value is 300 seconds, so in order to reschedule pods faster and maintain service availability, the default values have been decreased.
- **kube-controller-manager-arg**: Specifies the interval of kubelet health monitoring time.

- **kubelet-arg**: Sets kubelet arguments. node-status-update-frequency specifies the time in which node status is updated and max-pods is the maximum number of pods allowed to run on that node.
- **egress-selector-mode**: Disable rke2 egress mode. By default this value is set to agent and Rancher rke2 servers establish a tunnel to communicate with nodes. This behavior is due to prevent opening several connections over and over. In some cases, enabling this mode will cause some routing issues in your cluster, so it's been disabled in our scenario.
- **protect-kernel-defaults**: compare kubelet default parameters and OS kernel. If they're different, the container is killed.

(Optional) Taint Control Plane nodes

It's preferable to taint the Control Plane nodes so the workload pods won't get scheduled on those. To do so, edit `/etc/rancher/rke2/config.yaml` and add the following line inside it.

```
node-taint:
  - "CriticalAddonsOnly=true:NoExecute"
```

(Optional) Set up offline files

In order to speed up the bootstrapping process or to install Rancher in an air-gapped environment, we may want to download image files and put them in our nodes.
NOTE: Using offline files is useful but optional. In case you've got a poor internet connection or installing in an air-gapped environment, you should use the offline files.

Go to "https://github.com/rancher/rke2/releases" and select a release. After that, you should download ***rke2-images-core.linux-amd64.tar.gz***.
Create the Rancher data directory to put the compressed offline images.

```
mkdir -p /var/lib/rancher/rke2/agent/images
```

Put the compressed files in the Rancher data directory.

```
mv  rke2-images-core.linux-amd64.tar.gz /var/lib/rancher
```

Install rke2

Download rke2 installer script.

```
curl -sfL https://get.rke2.io > install_rke2.sh
```

Execute the binary with your desired arguments.
Here we've specified two arguments, the rke2 version and installation type:

```
chmod ug+x install_rke2.sh
INSTALL_RKE2_VERSION="v1.29.4+rke2r1" INSTALL_RKE2_TYPE=
```

NOTE: In this tutorial, we've installed rke2 version 1.29.4. You can change it in your environment.

Now you must have 2 new services available on your system, rke2-server and rke2-agent.

For server nodes, we just need rke2-server, so you should

disable and mask the rke2-agent service:

```
systemctl disable rke2-agent && systemctl mask rke2-agen
```

It's time to initiate our first cluster node. Start rke2-server service. The rke2-server service will bootstrap your control-plane node.

```
systemctl enable --now rke2-server.service
```

NOTE: This will take some time depending on your network bandwidth.

If no error occurred, verify the rke2 status:

```
systemctl status rke2-server
```

In case a problem has shown up, check logs to troubleshoot it:

```
journalctl -u rke2-server -f
```

To check the pods in the cluster, you'll need to use kubectl. The Rancher installation has downloaded the necessary binaries(e.g. kubectl, ctr, containerd,...) and put them in /var/lib/rancher/rke2/bin. Add the path to the default OS path(Let's assume you're using bash).

```
echo 'PATH=$PATH:/var/lib/rancher/rke2/bin' >> ~/.bashrc
source ~/.bashrc
```

Also, Rancher has generated **KUBECONFIG** which can be used by kubectl. By default, the KUBECONFIG has been set to use 127.0.0.1 as the API server which in our case should be set to an external address that is our load balancer(i.e. 192.168.100.100). Proceed with the subsequent steps:

```
mkdir ~/.kube
cp /etc/rancher/rke2/rke2.yaml ~/.kube/config
sed -i 's/127.0.0.1/192.168.100.100/g' ~/.kube/config
```

If you cat `~/.kube/config`, you'll have something like this:

You should now be able to use kubectl to check the status of your cluster.

NOTE: If you use kubectl, you'll see that nodes are in **NotReady** state and the reason for that, is due to the absence of a CNI plugin.

Install Cilium

We're ready to install Cilium as the CNI in our cluster.

The first step is to install Cilium CLI. To do so run the following commands to download and install the CLI.

```
CILIUM_CLI_VERSION=$(curl -s https://raw.githubuserconte
CLI_ARCH=amd64
curl -L --remote-name-all https://github.com/cilium/cili
sha256sum --check cilium-linux-${CLI_ARCH}.tar.gz.sha256
tar xzvfC cilium-linux-${CLI_ARCH}.tar.gz /usr/local/bin
```

NOTE: As of this day, our Cilium stable version is 1.15.4 and the version of Cilium CLI is 0.16.6.

Add the following config to a file named **cilium.yaml**:

```
cluster:
  name: cluster-1
  id: 10
prometheus:
  enabled: true
  serviceMonitor:
    enabled: false
dashboards:
  enabled: true
hubble:
  metrics:
    enabled:
    - dns:query;ignoreAAAA
    - drop
    - tcp
    - flow
    - icmp
    - http
    dashboards:
      enabled: true
  relay:
    enabled: true
    prometheus:
      enabled: true
  ui:
    enabled: true
    baseUrl: "/"
version: 1.15.4
operator:
  prometheus:
    enabled: true
  dashboards:
    enabled: true
```

NOTE: The Version of Cilium is 1.15.4. You can change it if there is a newer version, but be cautious to see the changelog of each version.

Apply the config:

```
cilium install -f cilium.yaml
```

Verify the status of Cilium:

```
cilium status
```

Now your node should be in **ready** state:

```
kubectl get nodes
```

By default, Cilium uses its own IP CIDR for pods and not the one configured during Cluster bootstrapping. To change this behavior, edit the Cilium config map:

```
kubectl -n kube-system edit cm cilium-config
```

There is a line like this:

```
ipam: cluster
```

Change the value to "kubernetes":

```
ipam: kubernetes
```

Save your changes and exit.

The already running pods are still using the default Cilium ipam. To apply yours, either restart the server or restart Cilium resources.

```
kubectl -n kube-system rollout restart deployment cilium
kubectl -n kube-system rollout restart ds cilium
```

NOTE: check other pods too. If they're using the old IPs, restart them too.

Our first Control plane node is ready.

Next, we'll proceed to join the other nodes to our initialized cluster.

Control Plane 2 & Control Plane 3

The procedure to join other server nodes is mostly the same as bootstrapping the first server node. The only difference is when we set which server node our new node should join to. So you should be able to follow the tutorial for all of the Server nodes 2, 3, etc.

Let's go through the steps

Set up Rancher configs

We need to change the default options and arguments of rke2.

Create the Rancher configuration directory and create and edit the Rancher config file:

```
mkdir -p /etc/rancher/rke2/
vim /etc/rancher/rke2/config.yaml
```

The configuration options are the same as the first server node. The only difference is the first 2 lines: **server** and **token**:

- **server**: This is a fixed registration address. It's a layer 4 load balancer and will distribute requests on our server nodes(the nodes that are running rke2-server service).
- **token**: The registration address. It's been generated by the first server node we bootstraped the cluster. The token value can be obtained from `/var/lib/rancher/rke2/server/node-token`.

Add these lines to the config.yaml:

```
server: https://192.168.100.100:9345
token: XXXXXXXXX
.
.
.
.
```

NOTE: We've omitted the next lines as they're the same as the first node.

(Optional) Set up offline files

Go to "https://github.com/rancher/rke2/releases" and download ***rke2-images-core.linux-amd64.tar.gz***.

Create the Rancher data directory to put the compressed offline images.

```
mkdir -p /var/lib/rancher/rke2/agent/images
mv  rke2-images-core.linux-amd64.tar.gz /var/lib/rancher
```

Install rke2

Download rke2 installer script. Make it executable and install the rke2 server:

```
curl -sfL https://get.rke2.io > install_rke2.sh
chmod ug+x install_rke2.sh
INSTALL_RKE2_VERSION="v1.29.4+rke2r1" INSTALL_RKE2_TYPE=
```

Disable and mask rke2-agent service:

```
systemctl disable rke2-agent && systemctl mask rke2-agen
```

Start rke2-server service:

```
systemctl enable --now rke2-server.service
```

If no error occurred, verify the Rancher status:

```
systemctl status rke2-server
```

In case a problem has arised, check logs to troubleshoot it:

```
journalctl -u rke2-server -f
```

Worker 1

It's time to join our first agent(worker) node to the cluster. All Kubernetes workloads are handled by our worker nodes.

Set up Rancher configs

Create the Rancher configuration directory.

```
mkdir -p /etc/rancher/rke2/
```

We need to change the default options and arguments of the rke2 agent. For that purpose, create the Rancher config file named **config.yaml** and put the below lines in it.

```
server: https://192.168.100.100:9345
token: XXXXXXXXXX
node-name: kuber-worker-1
kubelet-arg:
  - '--node-status-update-frequency=4s'
  - '--max-pods=100'
```

Let's explain the above options:

- **server**: As we mentioned in previous sections, This is a fixed registration address. It's the layer 4 load balancer and will distribute requests on our server nodes(the nodes that are running the rke2-server service).

- **token**: The registration address. It's been generated by the first server node we bootstraped the cluster. The token value can be obtained from `/var/lib/rancher/rke2/server/node-token`.
- **node-name**: A unique name for this worker node. This name is used by Rancher to identify node and must be unique.
- **kubelet-arg**: Kubelet specific arguments. As you've seen, all server nodes have the same kubelet arguments as this agent node. The kubelet arguments are the same across the cluster. Here we've specified two arguments for kubelet, one for the frequency of status updates reported to the kube-api-server and the other one for the maximum allowed pods on a single node.

(Optional) Set up offline files

Like the previous parts, we can provide offline files to install in an air-gapped environment or to make bootstrapping faster.

Go to "https://github.com/rancher/rke2/releases" and select a release. After that you should download ***rke2-images-core.linux-amd64.tar.gz***.
Create Rancher data directory to put the compressed offline images.

```
mkdir -p /var/lib/rancher/rke2/agent/images
```

Put the compressed files in the Rancher data directory.

```
mv  rke2-images-core.linux-amd64.tar.gz /var/lib/rancher
```

Install rke2

Download the rke2 installer script.

```
curl -sfL https://get.rke2.io > install_rke2.sh
```

Execute the binary with your arguments. Note that as we're installing on the agent node, you must specify **agent** mode in the arguments.

```
chmod ug+x install_rke2.sh
INSTALL_RKE2_VERSION="v1.29.4+rke2r1" INSTALL_RKE2_TYPE=
```

Disable and mask the rke2-server service as there is no need for it:

```
systemctl disable rke2-server && systemctl mask rke2-ser
```

Now we're about to join the first agent node to the cluster. Start rke2-agent service.

```
systemctl enable --now rke2-agent.service
```

NOTE: This will take some time depending on your network bandwidth.

If no error occurred, verify the Rancher status:

```
systemctl status rke2-agent
```

In case an error has shown, check logs to troubleshoot it:

```
journalctl -u rke2-agent -f
```

Worker 2 & Worker 3

The instructions on how to join the second and other worker nodes are exactly the same as the first one. The only difference is the node name.

In the Rancher configuration file located in `/etc/rancher/rke2/config.yaml`, the **node-name** option must be unique and different from the other nodes(e.g. kuber-worker-2, kuber-worker-3). For other parts, proceed with **Worker 1** guideline.

Congratulations! Now we've got a working Rancher Kubernetes cluster with 3 control plane and 3 worker nodes.

Deploy MetalLB

The last part is about deploying Metallb as a load balancer and IP pool management service. If you've got a cloud load balancer, then skip the rest of the tutorial.

Install MetalLB

For this part, we'll be using Helm to deploy MetalLB. The first thing to do is to add the MetalLB helm repo:

```
helm repo add metallb https://metallb.github.io/metallb
```

Now Install MetalLB:

```
helm install --create-namespace --namespace metallb-syst
```

Verify the pods are all in running state.

```
kubectl get pods -n metallb-system
```

Configure IP Address Pool

Let's assume we have a range IP starting from 192.168.100.50 and ending with 192.168.100.55. To create a pool for this range and have MetalLB assign IPs to services, create a yaml file(e.g. main-pool.yaml) and add the below content to it:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: main-pool
  namespace: metallb-system
spec:
  avoidBuggyIPs: true
  addresses:
  - 192.168.100.50-192.168.100.55
  serviceAllocation:
    serviceSelectors:
      - matchLabels:
        ip-pool: main-pool
```

NOTE: We've provided **serviceSelectors** section to only assign IPs to services that have the matching label.

Apply the pool:

```
kubectl apply -f main-pool.yaml
```

MetalLB will assign IP to any service of type LoadBalancer that has the label "main-pool".
The only remaining part is to advertise our IP pool in the entire cluster. Use **L2Advertisement** kind provided by MetalLB to do so.

Create a yaml file(e.g. l2advertisement.yaml) and add the following manifest in it:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: main-advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
    - main-pool
```

NOTE: If there was more than one pool, we could've advertised them all by this manifest and passed the pool names to **ipAddressPools** section as a list.

Apply the manifest:

```
kubectl apply -f l2advertisement.yaml
```

Everything is ready for you to deploy your workload into this cluster.
Good luck:)

Summary

In this essay, we showed how to use Rancher rke2 to deploy a Kubernetes cluster with 6 Debian nodes with firewall enabled. We've also covered deploying Cilium as a CNI for our cluster and have it completely replace kube-proxy so as to increase speed and gain more observability via Cilium tools.
This article also showed how to deploy Metallb to manage IP pools and load balance traffic for those IP pools.
Throughout this guide, we assumed that we have an external load balancer that will distribute traffic to our workload and control plane nodes.

For further information please visit rke2 and MetalLB official documents:

- https://docs.rke2.io/
- https://metallb.universe.tf/configuration/
- https://artifacthub.io/packages/helm/metallb/metallb

You may also want to visit my other article about deploying Rancher rke2 with Cilium and Metallb if you're interested: https://dev.to/arman-shafiei/deploy-nginx-load-balancer-for-rancher-lgk

Thank you for reading this post and please leave a comment if you have any suggestions or figured out an issue with this article.

Get started

Community matters

Top comments (4)

Subscribe

Add to the discussion

Mohammad Zakery • Apr 21

After executing the command:

systemctl enable --now rke2-server.service

I encountered an error with the following message:

Job for rke2-server.service failed because the control process exited with error code.
See "systemctl status rke2-server.service" and "journalctl -xeu rke2-server.service" for details.

Reply

Arman Shafiei • Apr 21

Hello there.
The question is kinda ambiguous.
There may be lots of reasons. One could be that your sysctl is configured incorrectly, Another thing may be that you have a problem with downloading files. Check your internet connectivity and make sure your machines can download the required images.
Also if you're on Redhat based OS, check that your Selinux is disabled.

1 like Reply

Jonathon Fritz • Jun 4

thanks for writing this post!

I have one question, what is the reason for opting to install cilium on its own and not using the chart that ships with rke2?

1 like Reply

Arman Shafiei • Jun 8

Hi Jonathon.
There are 2 reasons for that:
1- To customize Cilium at bootstrap. Going through RKE2 Helm files is a bit tricky.
2- As of my testing, the Cilium in the RKE2 Helm chart doesn't work well with kube-proxy replacement.
Actually it won't replace kube-proxy completely.

If I'm wrong in any part or missing anything, please let me know.

1 like Reply

Read next

Zero-Downtime Deployment for ASP.NET Applications in Kubernetes

Tugay Ersoy - Aug 4

9 Ways to Spin Up an EKS Cluster - Way 4 - CloudFormation

Ant(on) Weiss - Aug 4

Setup a k3s Cluster at home quickly

Vignesh Muthukumaran - Jul 14

An In-Depth Look at Kube-score : Day 22 of 50 days DevOps Tools Series

Shiivam Agnihotri - Aug 3