Install Production Kubernetes Cluster with Rancher RKE | ComputingForGeeks

Josphat Mutai

18-22 minutes

How can I use RKE to deploy Production ready Kubernetes Cluster?. Kubernetes has gained much traction and is now the standard orchestration layer for containerized workloads. If you want an open-source system for automating deployment of containerized applications without worrying about scaling and management, then Kubernetes is the right tool for you.

There are many standard ways of deploying a production-grade Kubernetes cluster. This include the use of tools such as kops, kubespray or manually building a cluster with **kubeadm.** We have some of the guides you can use for reference.

Deploy Production Ready Kubernetes Cluster with Ansible & Kubespray

Deploy Kubernetes Cluster on CentOS 7 / CentOS 8 With Ansible and Calico CNI

This guide walks you through the simple steps for installation a production-grade Kubernetes cluster with RKE. We'll set up a 5-node cluster with Rancher Kubernetes Engine (RKE) and install the *Rancher chart* with the Helm package manager.

What is RKE?

Rancher Kubernetes Engine (RKE), is an extremely simple, lightning fast Kubernetes distribution that runs entirely within containers. Rancher is a container management platform built for organizations that deploy containers in production. Rancher makes it easy to run Kubernetes everywhere, meet IT requirements, and empower DevOps teams.

Prepare Workstation machine

A number of CLI Tools are required on your Workstation where deployment will be done. This can also be a virtual machine that is able to access cluster nodes.

1. **kubectl:**

```
--- Linux ---
curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl
-s https://storage.googleapis.com/kubernetes-release/release/stable.txt`
/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
kubectl version --client

--- macOS ---
curl -LO "https://storage.googleapis.com/kubernetes-release/release
/$(curl -s https://storage.googleapis.com/kubernetes-release/release
/stable.txt)/bin/darwin/amd64/kubectl"
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
kubectl version --client
```

2. **rke**

```
--- Linux ---
```

```
curl -s https://api.github.com/repos/rancher/rke/releases/latest | grep
download_url | grep amd64 | cut -d '"' -f 4 | wget -qi -
chmod +x rke_linux-amd64
sudo mv rke_linux-amd64 /usr/local/bin/rke
rke --version

--- macOS ---
curl -s https://api.github.com/repos/rancher/rke/releases/latest | grep
download_url | grep darwin-amd64 | cut -d '"' -f 4 | wget -qi -
chmod +x rke_darwin-amd64
sudo mv rke_darwin-amd64 /usr/local/bin/rke
rke --version
```

3. **helm**

```
--- Helm 3 ---
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm
/master/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

Install Kubernetes with RKE

I'll be working on 5 nodes:

- **3 Master Nodes** – etcd and control plane (3 for HA)
- **2** Worker nodes – Scale to meet your Workloads demand

These are the specifications for my setup.

- Master Nodes – **8GB** of RAM and **4** vcpus
- Worker Machines – **16GB** of RAM and **8** vpcus

RKE Supported operating systems

RKE runs on almost any Linux OS with Docker installed. Rancher has been tested and is supported with:

- Red Hat Enterprise Linux
- Oracle Enterprise Linux
- CentOS Linux
- Ubuntu
- RancherOS

Step 1: Update your Linux System

The first step is to update your Linux machines which will be used to build the cluster.

```
--- CentOS ---
$ sudo yum -y update
$ sudo reboot

--- Ubuntu / Debian ---
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo reboot
```

Step 2: Create rke user

If using Red Hat Enterprise Linux, Oracle Enterprise Linux or CentOS, you cannot use the `root` user as SSH user due to Bugzilla 1527565. For this reason, we'll create a user account called **rke** for deployment purpose.

Using Ansible Playbook:

```
---
- name: Create rke user with passwordless sudo
  hosts: rke-hosts
  remote_user: root
  tasks:
    - name: Add RKE admin user
      user:
        name: rke
        shell: /bin/bash

    - name: Create sudo file
      file:
        path: /etc/sudoers.d/rke
        state: touch

    - name: Give rke user passwordless sudo
      lineinfile:
        path: /etc/sudoers.d/rke
        state: present
        line: 'rke ALL=(ALL:ALL) NOPASSWD: ALL'

    - name: Set authorized key taken from file
      authorized_key:
        user: rke
        state: present
        key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
```

Create user manually on all hosts

Login to each of your cluster nodes and create *rke* user.

```
sudo useradd rke
sudo passwd rke
```

Enable passwordless sudo for the user:

```
$ sudo vim /etc/sudoers.d/rke
rke  ALL=(ALL:ALL) NOPASSWD: ALL
```

Copy your ssh public key to the user's ~/**.ssh**/**authorized_keys** file.

```
for i in rke-master-01 rke-master-02 rke-master-03 rke-worker-01 rke-
worker-02; do
  ssh-copy-id rke@$i
done
```

Confirm you can login from your workstation:

```
$ ssh rke@rke-master-01
Warning: Permanently added 'rke-master-01,x.x.x.x' (ECDSA) to the list
of known hosts.
[rke@rke-master-01 ~]$ sudo su - # No password prompt
Last login: Mon Jan 27 21:28:53 CET 2020 from y.y.y.y on pts/0
[root@rke-master-01 ~]# exit
[rke@rke-master-01 ~]$ exit
logout
Connection to rke-master-01 closed.
```

Step 3: Enable required Kernel modules:

Using Ansible:

Create a playbook with below contents and run it against your RKE servers inventory.

```
---
```

```
- name: Load RKE kernel modules
  hosts: rke-hosts
  remote_user: root
  vars:
    kernel_modules:
      - br_netfilter
      - ip6_udp_tunnel
      - ip_set
      - ip_set_hash_ip
      - ip_set_hash_net
      - iptable_filter
      - iptable_nat
      - iptable_mangle
      - iptable_raw
      - nf_conntrack_netlink
      - nf_conntrack
      - nf_conntrack_ipv4
      - nf_defrag_ipv4
      - nf_nat
      - nf_nat_ipv4
      - nf_nat_masquerade_ipv4
      - nfnetlink
      - udp_tunnel
      - veth
      - vxlan
      - x_tables
      - xt_addrtype
      - xt_conntrack
      - xt_comment
      - xt_mark
      - xt_multiport
      - xt_nat
      - xt_recent
      - xt_set
      - xt_statistic
      - xt_tcpudp

  tasks:
    - name: Load kernel modules for RKE
      modprobe:
        name: "{{ item }}"
        state: present
      with_items: "{{ kernel_modules }}"
```

The manual way

Login to each host and enable Kernel modules required to run Kubernetes.

```
for module in br_netfilter ip6_udp_tunnel ip_set ip_set_hash_ip
ip_set_hash_net iptable_filter iptable_nat iptable_mangle iptable_raw
nf_conntrack_netlink nf_conntrack nf_conntrack_ipv4   nf_defrag_ipv4
nf_nat nf_nat_ipv4 nf_nat_masquerade_ipv4 nfnetlink udp_tunnel veth
vxlan x_tables xt_addrtype xt_conntrack xt_comment xt_mark xt_multiport
xt_nat xt_recent xt_set  xt_statistic xt_tcpudp;
     do
       if ! lsmod | grep -q $module; then
         echo "module $module is not present";
       fi;
```

Step 4: Disable swap and Modify sysctl entries

The recommendation of Kubernetes is to disable swap and add some sysctl values.

With Ansible:

```
---
- name: Disable swap and load kernel modules
  hosts: rke-hosts
  remote_user: root
  tasks:
    - name: Disable SWAP since kubernetes can't work with swap enabled
(1/2)
      shell: |
        swapoff -a

    - name: Disable SWAP in fstab since kubernetes can't work with swap
enabled (2/2)
      replace:
        path: /etc/fstab
        regexp: '^([^#].*?\sswap\s+.*)$'
        replace: '# \1'
    - name: Modify sysctl entries
      sysctl:
        name: '{{ item.key }}'
        value: '{{ item.value }}'
        sysctl_set: yes
        state: present
        reload: yes
      with_items:
        - {key: net.bridge.bridge-nf-call-ip6tables, value: 1}
        - {key: net.bridge.bridge-nf-call-iptables,  value: 1}
        - {key: net.ipv4.ip_forward,  value: 1}
```

Manually

Swap:

```
$ sudo vim /etc/fstab
# Add comment to swap line

$ sudo swapoff -a
```

Sysctl:

```
$ sudo tee -a /etc/sysctl.d/99-kubernetes.conf <<EOF
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

$ sysctl --system
```

Confirm is disabled:

```
$ free -h
              total        used        free      shared  buff/cache
available
Mem:           7.6G        180M        6.8G        8.5M        633M
7.2G
Swap:            0B          0B          0B
```

Step 5: Install Supported version of Docker

Each Kubernetes version supports different Docker versions. The Kubernetes release notes contain the current list of validated Docker versions.

As of this article, supported docker versions are:

Docker Version	Install Script
19.03	*curl https://releases.rancher.com/install-docker/19.03.sh | sh*

Docker Version	Install Script
18.09	*curl https://releases.rancher.com/install-docker/18.09.sh | sh*
18.06	*curl https://releases.rancher.com/install-docker/18.06.sh | sh*
17.03	*curl https://releases.rancher.com/install-docker/17.03.sh | sh*

You can either follow the Docker installation instructions or use one of Rancher's install scripts to install Docker. I'll install the latest supported version:

```
curl https://releases.rancher.com/install-docker/19.03.sh | sudo bash -
```

Start and enable docker service:

```
sudo systemctl enable --now docker
```

Confirm that a Kubernetes supported version of Docker is installed on your machine:

```
$ sudo docker version --format '{{.Server.Version}}'
18.09.2
```

Add **rke** user to docker group.

```
$ sudo usermod -aG docker rke
$ id rke
uid=1000(rke) gid=1000(rke) groups=1000(rke),994(docker)
```

Step 6: Open Ports on firewall

- **For a single-node installation,** you only need to open the ports required to enable Rancher to communicate with downstream user clusters.
- **For a high-availability installation,** the same ports need to be opened, as well as additional ports required to set up the Kubernetes cluster that Rancher is installed on.

Check all ports used in the requirements page

Firewalld TCP ports:

```
for i in 22 80 443 179 5473 6443 8472 2376 8472 2379-2380 9099 10250
10251 10252 10254 30000-32767; do
    sudo firewall-cmd --add-port=${i}/tcp --permanent
done
sudo firewall-cmd --reload
```

Firewalld UDP ports:

```
for i in 8285 8472 4789 30000-32767; do
   sudo firewall-cmd --add-port=${i}/udp --permanent
done
```

Step 6: Allow SSH TCP Forwarding

You need to enable your SSH server system-wide TCP forwarding.

Open ssh configuration file located at /**etc/ssh/sshd_config**:

```
$ sudo vi /etc/ssh/sshd_config
AllowTcpForwarding yes
```

Restart ssh service after making the change.

```
--- CentOS ---
$ sudo systemctl restart sshd

--- Ubuntu ---
$ sudo systemctl restart ssh
```

Step 7: Generate RKE cluster configuration file.

RKE uses a cluster configuration file, referred to as `cluster.yml` to determine what nodes will be in the cluster and how to deploy Kubernetes.

There are many configuration options that can be set in the `cluster.yml`. This file can be created from minimal example templates or generated with the **rke config** command.

Run *rke config* command to create a new *cluster.yml* in your current directory.

```
rke config --name cluster.yml
```

This command will prompt you for all the information needed to build a cluster.

If you want to create an empty template `cluster.yml` file instead, specify the `--empty` flag.

```
rke config --empty --name cluster.yml
```

This is how my cluster configuration file looks like – **Don't copy paste, just use it as reference** to create your own configuration.

```
# https://rancher.com/docs/rke/latest/en/config-options/
nodes:
- address: 10.10.1.10
  internal_address:
  hostname_override: rke-master-01
  role: [controlplane, etcd]
  user: rke
- address: 10.10.1.11
  internal_address:
  hostname_override: rke-master-02
  role: [controlplane, etcd]
  user: rke
- address: 10.10.1.12
  internal_address:
  hostname_override: rke-master-03
  role: [controlplane, etcd]
  user: rke
- address: 10.10.1.13
  internal_address:
  hostname_override: rke-worker-01
  role: [worker]
  user: rke
- address: 10.10.1.114
  internal_address:
  hostname_override: rke-worker-02
  role: [worker]
  user: rke

# using a local ssh agent
# Using SSH private key with a passphrase - eval `ssh-agent -s` && ssh-
add
ssh_agent_auth: true

#  SSH key that access all hosts in your cluster
ssh_key_path: ~/.ssh/id_rsa

# By default, the name of your cluster will be local
# Set different Cluster name
cluster_name: rke

# Fail for Docker version not supported by Kubernetes
ignore_docker_version: false

# prefix_path: /opt/custom_path

# Set kubernetes version to install: https://rancher.com/docs/rke/latest
/en/upgrades/#listing-supported-kubernetes-versions
# Check with -> rke config --list-version --all
```

```
kubernetes_version:
# Etcd snapshots
services:
  etcd:
    backup_config:
      interval_hours: 12
      retention: 6
    snapshot: true
    creation: 6h
    retention: 24h

kube-api:
  # IP range for any services created on Kubernetes
  #  This must match the service_cluster_ip_range in kube-controller
  service_cluster_ip_range: 10.43.0.0/16
  # Expose a different port range for NodePort services
  service_node_port_range: 30000-32767
  pod_security_policy: false


kube-controller:
  # CIDR pool used to assign IP addresses to pods in the cluster
  cluster_cidr: 10.42.0.0/16
  # IP range for any services created on Kubernetes
  # # This must match the service_cluster_ip_range in kube-api
  service_cluster_ip_range: 10.43.0.0/16

kubelet:
  # Base domain for the cluster
  cluster_domain: cluster.local
  # IP address for the DNS service endpoint
  cluster_dns_server: 10.43.0.10
  # Fail if swap is on
  fail_swap_on: false
  # Set max pods to 150 instead of default 110
  extra_args:
    max-pods: 150

# Configure  network plug-ins
# KE provides the following network plug-ins that are deployed as add-
ons: flannel, calico, weave, and canal
# After you launch the cluster, you cannot change your network provider.
# Setting the network plug-in
network:
    plugin: canal
    options:
      canal_flannel_backend_type: vxlan

# Specify DNS provider (coredns or kube-dns)
dns:
  provider: coredns

# Currently, only authentication strategy supported is x509.
# You can optionally create additional SANs (hostnames or IPs) to
# add to the API server PKI certificate.
# This is useful if you want to use a load balancer for the
# control plane servers.
authentication:
  strategy: x509
  sans:
    - "k8s.computingforgeeks.com"

# Set Authorization mechanism
authorization:
    # Use `mode: none` to disable authorization
```

```
    mode: rbac

# Currently only nginx ingress provider is supported.
# To disable ingress controller, set `provider: none`
# `node_selector` controls ingress placement and is optional
ingress:
  provider: nginx
  options:
     use-forwarded-headers: "true"
```

In my configuration, the master nodes only have **etcd** and **controlplane** roles. But they can be used to schedule pods by adding **worker** role.

```
role: [controlplane, etcd, worker]
```

Step 7: Deploy Kubernetes Cluster with RKE

Once you've created the **cluster.yml** file, you can deploy your cluster with a simple command.

```
rke up
```

This command assumes the *cluster.yml* file is in the same directory as where you are running the command. If using a different filename, specify it like below.

```
$ rke up --config ./rancher_cluster.yml
```

Using SSH private key with a passphrase – eval `ssh-agent -s` && ssh-add

Ensure the setup doesn't show any failure in its output:

```
......
INFO[0181] [sync] Syncing nodes Labels and Taints
INFO[0182] [sync] Successfully synced nodes Labels and Taints
INFO[0182] [network] Setting up network plugin: canal
INFO[0182] [addons] Saving ConfigMap for addon rke-network-plugin to
Kubernetes
INFO[0183] [addons] Successfully saved ConfigMap for addon rke-network-
plugin to Kubernetes
INFO[0183] [addons] Executing deploy job rke-network-plugin
INFO[0189] [addons] Setting up coredns
INFO[0189] [addons] Saving ConfigMap for addon rke-coredns-addon to
Kubernetes
INFO[0189] [addons] Successfully saved ConfigMap for addon rke-coredns-
addon to Kubernetes
INFO[0189] [addons] Executing deploy job rke-coredns-addon
INFO[0195] [addons] CoreDNS deployed successfully..
INFO[0195] [dns] DNS provider coredns deployed successfully
INFO[0195] [addons] Setting up Metrics Server
INFO[0195] [addons] Saving ConfigMap for addon rke-metrics-addon to
Kubernetes
INFO[0196] [addons] Successfully saved ConfigMap for addon rke-metrics-
addon to Kubernetes
INFO[0196] [addons] Executing deploy job rke-metrics-addon
INFO[0202] [addons] Metrics Server deployed successfully
INFO[0202] [ingress] Setting up nginx ingress controller
INFO[0202] [addons] Saving ConfigMap for addon rke-ingress-controller to
Kubernetes
INFO[0202] [addons] Successfully saved ConfigMap for addon rke-ingress-
controller to Kubernetes
INFO[0202] [addons] Executing deploy job rke-ingress-controller
INFO[0208] [ingress] ingress controller nginx deployed successfully
INFO[0208] [addons] Setting up user addons
INFO[0208] [addons] no user addons defined
INFO[0208] Finished building Kubernetes cluster successfully
```

Step 8: Accessing your Kubernetes cluster

As part of the Kubernetes creation process, a `kubeconfig` file has been created and written at `kube_config_cluster.yml`.

Set *KUBECONFIG* variable to the file generated.

```
export KUBECONFIG=./kube_config_cluster.yml
```

Check list of nodes in the cluster.

```
$ kubectl get nodes
NAME             STATUS   ROLES               AGE     VERSION
rke-master-01    Ready    controlplane,etcd   16m     v1.17.0
rke-master-02    Ready    controlplane,etcd   16m     v1.17.0
rke-master-03    Ready    controlplane,etcd   16m     v1.17.0
rke-worker-01    Ready    worker              6m33s   v1.17.0
rke-worker-02    Ready    worker              16m     v1.17.0
```

You can copy this file to **$HOME/.kube/config** if you don't have any other kubernetes cluster.

Step 9: Install Kubernetes Dashboard

If you'll prefer deploying containerized applications on Kubernetes from a dashboard, use our guide below.

How To Install Kubernetes Dashboard with NodePort

Learning Courses:

In our next guide, we'll cover installation of Rancher – An **open-source multi-cluster orchestration platform** that lets you easily manage and secure your enterprise Kubernetes.

We really appreciate you supporting our efforts by buying us Coffee:

Coming up with fresh, high quality content takes time. Sometimes working late at night building labs and then doing the writing. We appreciate if you consider supporting our efforts with a cup of coffee to keep us awake and always deliver.

No contribution is small. We are greatful for any amount you support us with. Thank you!