How to Install, Configure, and Deploy NGINX on a Kubernetes Cluster

What is Kubernetes?

[Kubernetes](https://kubernetes.io/) is an open-source container management system that is based on [Google Borg](https://research.google.com/pubs/pub43438.html). It can be configured to provide highly available, horizontally autoscaling, automated deployments. This guide shows you how to manually set up a Kubernetes cluster on a Linode and manage the lifecycle of an NGINX service.

**Note**

You can now create a Kubernetes cluster with one command using the Linode CLI. To provision Kubernetes on Linodes, this tool uses the [Linode Kubernetes Terraform module](https://github.com/linode/terraform-linode-k8s), the [Linode Cloud Controller Manager (CCM)](https://github.com/linode/linode-cloud-controller-manager), and the [Container Storage Interface (CSI) Driver](https://github.com/linode/linode-blockstorage-csi-driver) for Linode Block Storage. See the [Kubernetes Tools](https://developers.linode.com/kubernetes/) page for installation steps. For an in-depth dive into the the Linode Kubernetes Terraform module, see its related [Community Site post](https://www.linode.com/community/questions/17611/the-linode-kubernetes-module-for-terraform).

Before You Begin

You will need:

* Two or more Linodes with [Private IPs](https://www.linode.com/docs/networking/remote-access#adding-private-ip-addresses)
* Each Linode should have a 64-bit distribution of either:
  + Ubuntu 16.04+
  + Debian 9
  + CentOS 7
  + RHEL 7
  + Fedora 26
* At least 2GB RAM per Linode
* Root or sudo privileges to install and configure Kubernetes. Any user can interact with the cluster once it’s configured.

Prepare the Host Linode for Kubernetes

The steps in this guide create a two-node cluster. Evaluate your own resource requirements and launch an appropriately-sized cluster for your needs.

1. Create two Linodes with at least 2GB memory within the same data center.
2. For each node, go into the Remote Access tab of your Linode Manager and add a [private IP](https://www.linode.com/docs/networking/remote-access#adding-private-ip-addresses). It is possible to build a Kubernetes cluster using public IPs between data centers, but performance and security may suffer.
3. Configure a firewall with [UFW](https://www.linode.com/docs/security/firewalls/configure-firewall-with-ufw/) or [iptables](https://www.linode.com/docs/security/firewalls/control-network-traffic-with-iptables/) to ensure only the two nodes can communicate with each other.

When configuring your firewall, a good place to start is to create rules for the ports Kubernetes requires to function. This includes any inbound traffic on Master nodes and their required ports. If you have changed any custom ports, you should ensure those ports are also open. Master Nodes will have a public IP address or 192.168.0.0/16. See the chart below for more details.

On Worker nodes, you should allow inbound Kubelet traffic. For NodePort traffic you should allow a large range from the world or if you are using the [Linode NodeBalancers service](https://github.com/linode/linode-cloud-controller-manager)exclusively for ingress, 192.168.255.0/24. See the chart below for more details.

The table below provides a list of the required ports for Master nodes and Worker nodes. You should also include port 22.

**Master nodes**

| Protocol | Direction | Port Range | Purpose | Used By |
| --- | --- | --- | --- | --- |
| TCP | Inbound | 6443\* | Kubernetes API server | All |
| TCP | Inbound | 2379-2380 | etcd server client API | kube-apiserver, etcd |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10251 | kube-scheduler | Self |
| TCP | Inbound | 10253 | kube-controller-manager | Self |

**Worker nodes**

| Protocol | Direction | Port Range | Purpose | Used By |
| --- | --- | --- | --- | --- |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 30000-32767 | NodePort Services\*\* | All |

**Note**

By design, kube-proxy will always place its iptables chains first. It inserts 2 rules, KUBE-EXTERNAL-SERVICES and KUBE-FIREWALL at the top of the INPUT chain. See the [Kubernetes discussion forum](https://discuss.kubernetes.io/t/custom-iptables-rules-for-input-chain/3509) for more details.

1. You should consider using the Linode NodeBalancer service with the [Linode Cloud Controller Manager (CCM)](https://github.com/linode/linode-cloud-controller-manager).
   * When using Linode NodeBalancers ensure you add iptables rules to allow the NodeBalancer traffic: 192.168.255.0/24.
2. To obtain persistent storage capabilities, you can use the [Container Storage Interface (CSI) Driver](https://github.com/linode/linode-blockstorage-csi-driver) for Linode Block Storage.

Disable Swap Memory

Linodes come with swap memory enabled by default. [kubelets](https://kubernetes.io/docs/admin/kubelet/) do not support swap memory and will not work if swap is active or even present in your /etc/fstab file.

The /etc/fstab should look something like this:

**/etc/fstab**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # /etc/fstab: static file system information.  #  # use 'blkid' to print the universally unique identifier for a  # device; this may be used with uuid= as a more robust way to name devices  # that works even if disks are added and removed. see fstab(5).  #  # &lt;file system&gt; &lt;mount point&gt; &lt;type&gt; &lt;options&gt; &lt;dump&gt; &lt;pass&gt;  # / was on /dev/sda1 during installation  /dev/sda / ext4 noatime,errors=remount-ro 0 1  /dev/sdb none swap sw 0 0 |

1. Delete the line describing the swap partition. In this example, Line 10 with /dev/sdb.
2. Disable swap memory usage:
3. swapoff -a

Set Hostnames for Kubernetes Nodes

To make the commands in this guide easier to understand, set up your hostname and hosts files on each of your machines.

1. Choose a node to designate as your Kubernetes master node and SSH into it.
2. Edit /etc/hostname, and add:

**/etc/hostname**

|  |  |
| --- | --- |
| 1 | kube-master |

1. Add the following lines to /etc/hosts:

**/etc/hosts**

|  |  |
| --- | --- |
| 1  2 | <kube-master-private-ip> kube-master  <kube-worker-private-ip> kube-worker-1 |

If you have more than two nodes, add their private IPs to /etc/hosts as well.

To make it easier to understand output and debug issues later, consider naming each hostname according to its role (kube-worker-1, kube-worker-2, etc.).

1. Perform Steps 2 and 3 on each worker node, changing the values accordingly.
2. For the changes to take effect, restart your Linodes.

Confirm Hostnames[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/#confirm-hostnames)

Once your nodes have rebooted, log into each to confirm your changes.

Check that:

* $ hostname in the terminal outputs the expected hostname.
* You can ping all of the nodes in your cluster by their hostnames.
* Swap is correctly disabled on all nodes using cat /proc/swaps.

If you are unable to ping any of your hosts by their hostnames or private IPs:

1. SSH into the host that isn’t responding.
2. Enter ifconfig. You should see an entry for eth0:1 that lists your private IP. If eth0:1 isn’t listed, it’s possible that you deployed your Linode image before adding a private IP to the underlying host. Recreate the image and return to the beginning of the guide.

Install Docker and Kubernetes on Linode[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/" \l "install-docker-and-kubernetes-on-linode" \o "Permalink)

**Debian/Ubuntu:**

apt install ebtables ethtool

**CentOS/RHEL:**

yum install ebtables ethtool

Install Docker[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/#install-docker)

These steps install Docker Community Edition (CE) using the official Ubuntu repositories. To install on another distribution, see the official [installation page](https://docs.docker.com/install/).

1. Remove any older installations of Docker that may be on your system:
2. sudo apt remove docker docker-engine docker.io
3. Make sure you have the necessary packages to allow the use of Docker’s repository:
4. sudo apt install apt-transport-https ca-certificates curl software-properties-common
5. Add Docker’s GPG key:
6. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
7. Verify the fingerprint of the GPG key:
8. sudo apt-key fingerprint 0EBFCD88

You should see output similar to the following:

pub 4096R/0EBFCD88 2017-02-22

Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

uid Docker Release (CE deb)

sub 4096R/F273FCD8 2017-02-22

1. Add the stable Docker repository:
2. sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"
3. Update your package index and install Docker CE:
4. sudo apt update
5. sudo apt install docker-ce
6. Add your limited Linux user account to the docker group:
7. sudo usermod -aG docker $USER

**Note**

After entering the usermod command, you will need to close your SSH session and open a new one for this change to take effect.

1. Check that the installation was successful by running the built-in “Hello World” program:
2. docker run hello-world

Install kubeadm, kubectl, and kubelet[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/" \l "install-kubeadm-kubectl-and-kubelet" \o "Permalink)

**Debian/Ubuntu:**

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

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

sudo apt update

sudo apt install -y kubelet kubeadm kubectl

**CentOS/RHEL:**

cat <<eof > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg

https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

eof

setenforce 0

yum install -y kubelet kubeadm kubectl

systemctl enable kubelet && systemctl start kubelet

Kubernetes Master and Slave[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/#kubernetes-master-and-slave)

Configure the Kubernetes Master Node[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/#configure-the-kubernetes-master-node)

1. On the master node initialize your cluster using its private IP:
2. kubeadm init --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=<private IP>

If you encounter a warning stating that swap is enabled, return to the Disable Swap Memorysection.

If successful, your output will resemble:

To start using your cluster, you need to run (as a regular user):

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

http://kubernetes.io/docs/admin/addons/

You can now join any number of machines by running the following on each node

as root:

kubeadm join --token 921e92.d4582205da623812 :6443 --discovery-token-ca-cert-hash sha256:bd85666b6a97072709b210ddf677245b4d79dab88d61b4a521fc00b0fbcc710c

1. On the master node, configure the kubectl tool:
2. mkdir -p $HOME/.kube
3. sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
4. sudo chown $(id -u):$(id -g) $HOME/.kube/config
5. Check on the status of the nodes with kubectl get nodes. Output will resemble:
6. root@kube-master:~# kubectl get nodes
7. name status roles age version
8. kube-master NotReady master 1m v1.8.1

The master node is listed as NotReady because the cluster does not have a Container Networking Interface ([CNI](https://github.com/containernetworking/cni/blob/master/spec.md)). CNI is a spec for a of container based network interface. In this guide, we will be using Calico. Alternatively, you can use [Flannel](https://raw.githubusercontent.com/coreos/flannel/v0.9.0/Documentation/kube-flannel.yml) or another CNI for similar results.

The --pod-network-cidr argument used in the [Configure the Kubernetes Master Node](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/#configure-the-kubernetes-master-node) section defines the network range for the CNI.

1. While still on the master node run the following command to deploy the CNI to your cluster:
2. kubectl apply -f https://docs.projectcalico.org/v2.6/getting-started/kubernetes/installation/hosted/kubeadm/1.6/calico.yaml
3. To ensure Calico was set up correctly, use kubectl get pods --all-namespaces to view the pods created in the kube-system namespace:
4. root@kube-master:~# kubectl get pods --all-namespaces
5. NAMESPACE NAME READY STATUS RESTARTS AGE
6. kube-system calico-etcd-nmx26 1/1 Running 0 48s
7. kube-system calico-kube-controllers-6ff88bf6d4-p25cw 1/1 Running 0 47s
8. kube-system calico-node-bldzb 1/2 CrashLoopBackOff 2 48s
9. kube-system calico-node-k5c9m 2/2 Running 0 48s
10. kube-system etcd-master 1/1 Running 0 3m
11. kube-system kube-apiserver-master 1/1 Running 0 3m
12. kube-system kube-controller-manager-master 1/1 Running 0 3m
13. kube-system kube-dns-545bc4bfd4-g8xtm 3/3 Running 0 4m
14. kube-system kube-proxy-sw562 1/1 Running 0 4m
15. kube-system kube-proxy-x6psn 1/1 Running 0 1m
16. kube-system kube-scheduler-master 1/1 Running 0 3m

This command uses the -n flag. The -n flag is a global kubectl flag that selects a non-default namespace. We can see our existing name spaces by running kubectl get namespaces:

root@kube-master:~# kubectl get namespaces

NAME STATUS AGE

default Active 4h

kube-public Active 4h

kube-system Active 4h

1. Run kubectl get nodes again to see that the master node is now running properly:
2. root@kube-master:~# kubectl get nodes
3. name status roles age version
4. kube-master Ready master 12m v1.8.1

Add Nodes to the Kubernetes Cluster[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/#add-nodes-to-the-kubernetes-cluster)

1. Run kubeadm join with the kube-master hostname to add the first worker:
2. kubeadm join --token <some-token> kube-master:6443 --discovery-token-ca-cert-hash sha256:<some-sha256-hash>
3. On the master node, use kubectl to see that the slave node is now ready:
5. root@kube-master:~# kubectl get nodes
6. name status roles age version
7. kube-master ready master 37m v1.8.1
8. kube-worker-1 ready 2m v1.8.1

Deploy NGINX on the Kubernetes Cluster[:](https://www.linode.com/docs/applications/containers/how-to-deploy-nginx-on-a-kubernetes-cluster/#deploy-nginx-on-the-kubernetes-cluster)

A *deployment* is a logical reference to a pod or pods and their configurations.

1. From your master node kubectl create an nginx deployment:
2. kubectl create deployment nginx --image=nginx
3. This creates a deployment called nginx. kubectl get deployments lists all available deployments:
4. kubectl get deployments
5. Use kubectl describe deployment nginx to view more information:
7. Name: nginx
8. Namespace: default
9. CreationTimestamp: Sun, 15 Oct 2017 06:10:50 +0000
10. Labels: app=nginx
11. Annotations: deployment.kubernetes.io/revision=1
12. Selector: app=nginx
13. Replicas: 1 desired | 1 updated | 1 total | 1 available | 0 unavailable
14. StrategyType: RollingUpdate
15. MinReadySeconds: 0
16. RollingUpdateStrategy: 1 max unavailable, 1 max surge
17. Pod Template:
18. Labels: app=nginx
19. Containers:
20. nginx:
21. Image: nginx
22. Port:
23. Environment:
24. Mounts:
25. Volumes:
26. Conditions:
27. Type Status Reason
28. ---- ------ ------
29. Available True MinimumReplicasAvailable
30. OldReplicaSets:
31. NewReplicaSet: nginx-68fcbc9696 (1/1 replicas created)
32. Events:
33. Type Reason Age From Message
34. ---- ------ ---- ---- -------
35. Normal ScalingReplicaSet 1m deployment-controller Scaled up replica set nginx-68fcbc9696 to 1

The describe command allows you to interrogate different kubernetes resources such as pods, deployments, and services at a deeper level. The output above indicates that there is a deployment called nginx within the default namespace. This deployment has a single replicate, and is running the docker image nginx. The ports, mounts, volumes and environmental variable are all unset.

1. Make the NGINX container accessible via the internet:
2. kubectl create service nodeport nginx --tcp=80:80

This creates a public facing service on the host for the NGINX deployment. Because this is a nodeport deployment, kubernetes will assign this service a port on the host machine in the 32000+ range.

Try to get the current services:

root@kube-master:~# kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 5h

nginx NodePort 10.98.24.29 <none> 80:32555/TCP 52s

1. Verify that the NGINX deployment is successful by using curl on the slave node:
2. root@kube-master:~# curl kube-worker-1:32555

The output will show the unrendered “Welcome to nginx!” page HTML.

1. To remove the deployment, use kubectl delete deployment:
2. root@kube-master:~# kubectl delete deployment nginx
3. deployment "nginx" deleted
4. root@kube-master:~# kubectl get deployments
5. No resources found.