

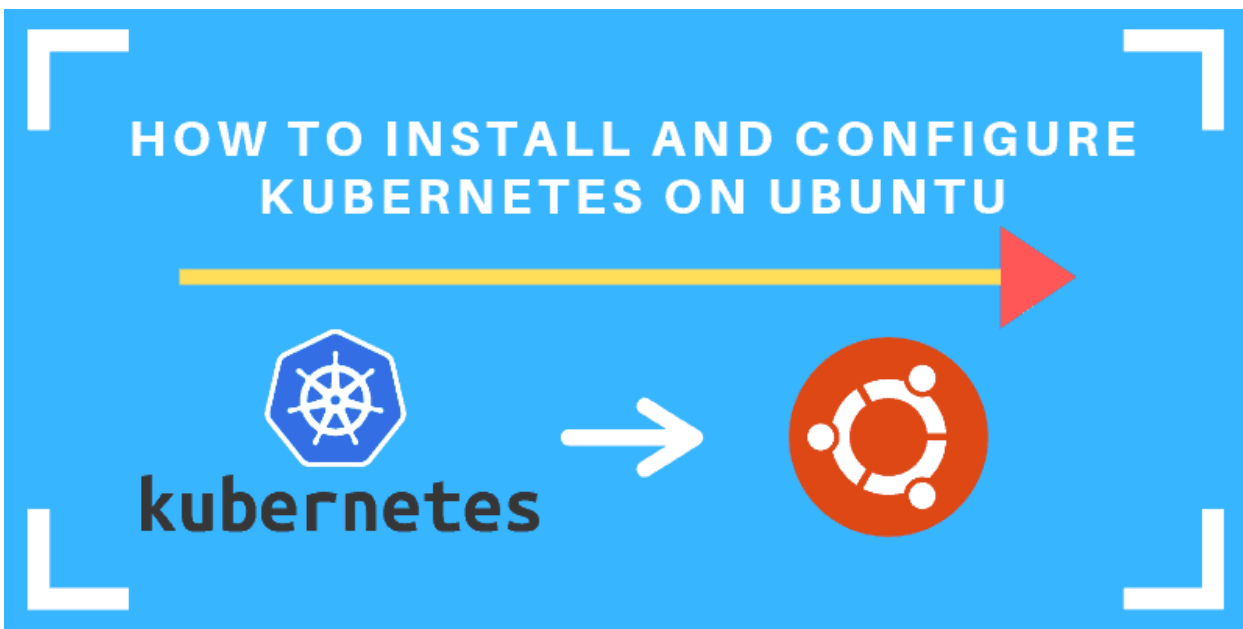
Kubernetes Installation on Ubuntu

Introduction

Kubernetes is an open source platform for managing container technologies such as Docker.

Docker lets you create containers for a pre-configured image and application. Kubernetes provides the next step, allowing you to balance loads between containers and run multiple containers across multiple systems.

This guide will walk you through how to install Kubernetes on Ubuntu 18.04.



Prerequisites

- **2 or more Linux servers running Ubuntu 18.04 /20.04 on Virtual box or you can use EC2 free tier instances choose the ubuntu 20.04 AMI free tier**
- Access to a user account on each system with sudo or root privileges
- The apt package manager, included by default
- Command-line/terminal window (Ctrl-Alt-T)

Steps to Install Kubernetes on Ubuntu

Set up Docker

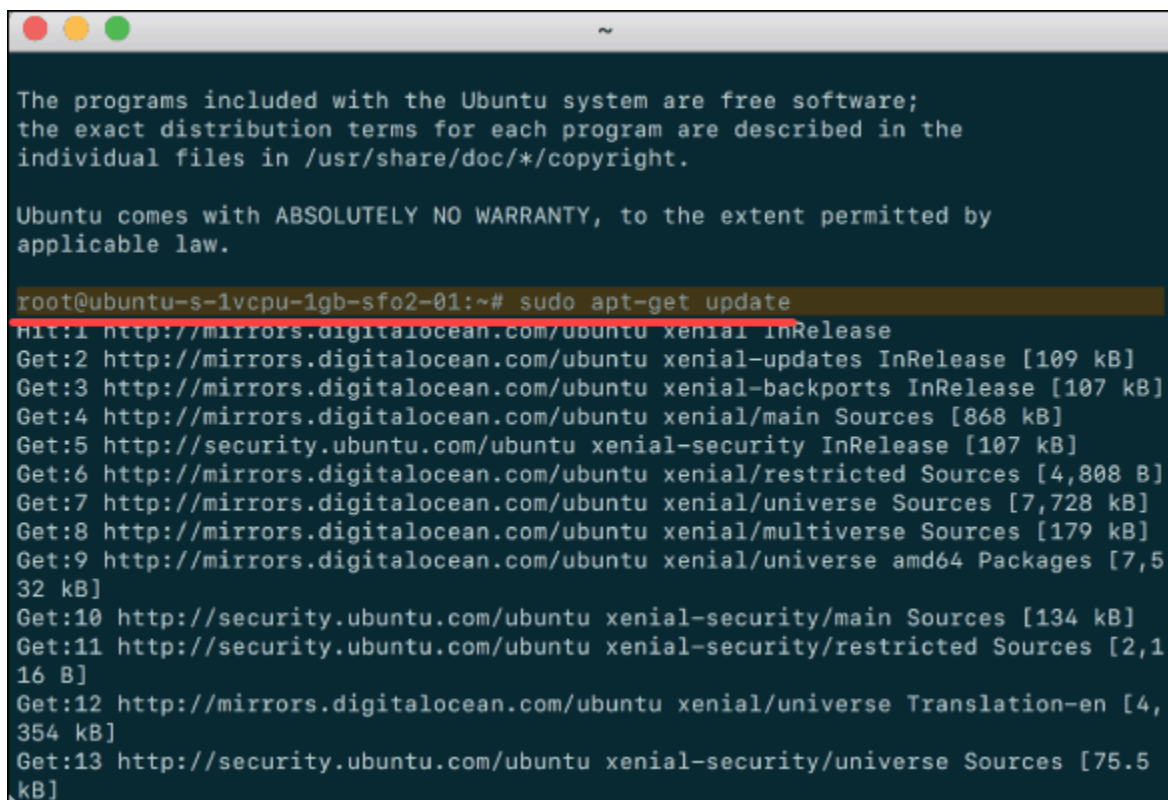
Step 1: Install Docker

Kubernetes requires an existing Docker installation. If you already have Docker installed, skip ahead to Step 2.

If you do not have Kubernetes, install it by following these steps:

1. Update the package list with the command:

```
on-master&slave$sudo apt-get update
```

A terminal window with a dark blue background and white text. The window title bar shows three colored circles (red, yellow, green) on the left. The terminal output shows the Ubuntu copyright notice, a warranty disclaimer, and the command 'root@ubuntu-s-1vcpu-1gb-sfo2-01:~# sudo apt-get update'. The command output lists 13 sources being updated, including mirrors.digitalocean.com and security.ubuntu.com, with their respective package lists and sizes. The last line of the output is 'Get:13 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [75.5 kB]'.

```
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
  
root@ubuntu-s-1vcpu-1gb-sfo2-01:~# sudo apt-get update  
Hit:1 http://mirrors.digitalocean.com/ubuntu xenial InRelease  
Get:2 http://mirrors.digitalocean.com/ubuntu xenial-updates InRelease [109 kB]  
Get:3 http://mirrors.digitalocean.com/ubuntu xenial-backports InRelease [107 kB]  
Get:4 http://mirrors.digitalocean.com/ubuntu xenial/main Sources [868 kB]  
Get:5 http://security.ubuntu.com/ubuntu xenial-security InRelease [107 kB]  
Get:6 http://mirrors.digitalocean.com/ubuntu xenial/restricted Sources [4,808 B]  
Get:7 http://mirrors.digitalocean.com/ubuntu xenial/universe Sources [7,728 kB]  
Get:8 http://mirrors.digitalocean.com/ubuntu xenial/multiverse Sources [179 kB]  
Get:9 http://mirrors.digitalocean.com/ubuntu xenial/universe amd64 Packages [7,532 kB]  
Get:10 http://security.ubuntu.com/ubuntu xenial-security/main Sources [134 kB]  
Get:11 http://security.ubuntu.com/ubuntu xenial-security/restricted Sources [2,116 B]  
Get:12 http://mirrors.digitalocean.com/ubuntu xenial/universe Translation-en [4,354 kB]  
Get:13 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [75.5 kB]
```

2. Next, install Docker with the command:

```
on-master&slave$sudo apt-get install docker.io
```

3. Repeat the process on each server that will act as a node.

4. Check the installation (and version) by entering the following:

```
on-master&slave$docker --version
```

Step 2: Start and Enable Docker

1. Set Docker to launch at boot by entering the following:

```
on-master&slave$sudo systemctl enable docker
```

2. Verify Docker is running:

```
on-master&slave$sudo systemctl status docker
```

To start Docker if it's not running:

```
on-master&slave$sudo systemctl start docker
```

3. Repeat on all the other nodes.

Install Kubernetes

Step 3: Add Kubernetes Signing Key

Since you are downloading Kubernetes from a non-standard repository, it is essential to ensure that the software is authentic. This is done by adding a signing key.

1. Enter the following to add a signing key:

```
on-master&slave$curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key  
add
```

If you get an error that curl is not installed, install it with:

```
on-master&slave$sudo apt-get install curl
```

2. Then repeat the previous command to install the signing keys. Repeat for each server node.

Step 4: Add Software Repositories

Kubernetes is not included in the default repositories. To add them, enter the following:

```
on-master&slave$sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
```

Repeat on each server node.

Step 5: Kubernetes Installation Tools

Kubeadm (Kubernetes Admin) is a tool that helps initialize a cluster. It fast-tracks setup by using community-sourced best practices. Kubelet is the work package, which runs on every node and starts containers. The tool gives you command-line access to clusters.

1. Install Kubernetes tools with the command:

```
on-master&slave$sudo apt-get install kubeadm kubelet kubectl -y  
on-master&slave$sudo apt-mark hold kubeadm kubelet kubectl
```

Allow the process to complete.

2. Verify the installation with:

```
on-master&slave$kubeadm version
```

3. Repeat for each server node.

Note: Make sure you install the same version of each package on each machine. Different versions can create instability. Also, this process prevents apt from automatically updating Kubernetes. For update instructions, please see the [developers' instructions](#).

Kubernetes Deployment

Step 6: Begin Kubernetes Deployment

Start by disabling the swap memory on each server:

```
on-master&slave$sudo swapoff --a
```

Step 7: Assign Unique Hostname for Each Server Node

Decide which server to set as the master node. Then enter the command:

```
on-master$sudo hostnamectl set-hostname master-node
```

Next, set a worker node hostname by entering the following on the worker server:

```
on-slave$sudo hostnamectl set-hostname worker01
```

If you have additional worker nodes, use this process to set a unique hostnsame on each.

Step 8: Initialize Kubernetes on Master Node

Switch to the master server node, and enter the following:

```
on-master$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

If you are trying to run this on EC2 you'll get an error message saying less cpu and memory to override the error run the above command with `--ignore-preflight-errors=all`

For eg: `on-master$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all`

Once this command finishes, it will display a kubeadm join message at the end. Make a note of the whole entry. This will be used to join the worker nodes to the cluster.

Next, enter the following to create a directory for the cluster:

```
kubernetes-master:~$ mkdir -p $HOME/.kube
```

```
kubernetes-master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
kubernetes-master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Step 9: Deploy Pod Network to Cluster

A Pod Network is a way to allow communication between different nodes in the cluster. This tutorial uses the flannel virtual network.

Enter the following:

```
kubernetes-master:~$ sudo kubectl apply -f
```

```
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

Allow the process to complete.

Verify that everything is running and communicating:

```
kubernetes-master:~$ kubectl get pods --all-namespaces
```

Step 10: Join Worker Node to Cluster

As indicated in Step 7, you can enter the kubeadm join command on each worker node to connect it to the cluster.

Switch to the worker01 system and enter the command you noted from Step 7:

```
kubernetes-slave:~$ kubeadm join --discovery-token abcdef.1234567890abcdef --discovery-token-  
ca-cert-hash sha256:1234..cdef 1.2.3.4:6443
```

ON EC2 make sure you open the port in security group ADVERTISED HERE:

Replace the alphanumeric codes with those from your master server. Repeat for each worker node on the cluster. Wait a few minutes; then you can check the status of the nodes.

Switch to the master server, and enter:

```
kubernetes-master:~$ kubectl get nodes
```

The system should display the worker nodes that you joined to the cluster.

Output

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	1d	v1.14.0
worker1	Ready	<none>	1d	v1.14.0

If all of your nodes have the value Ready for STATUS, it means that they're part of the cluster and ready to run workloads.

If, however, a few of the nodes have NotReady as the STATUS, it could mean that the worker nodes haven't finished their setup yet. Wait for around five to ten minutes before re-running kubectl get node and inspecting the new output. If a few nodes still have NotReady as the status, you might have to verify and re-run the commands in the previous steps.

Now that your cluster is verified successfully, let's schedule an example Nginx application on the cluster.

Running An Application on the Cluster

You can now deploy any containerized application to your cluster. To keep things familiar, let's deploy Nginx using *Deployments* and *Services* to see how this application can be deployed to the cluster. You can use the commands below for other containerized applications as well, provided you change the Docker image name and any relevant flags (such as ports and volumes).

Still within the master node, execute the following command to create a deployment named nginx:

```
kubernetes-master:~$kubectl create deployment nginx --image=nginx
```

A deployment is a type of Kubernetes object that ensures there's always a specified number of pods running based on a defined template, even if the pod crashes during the cluster's lifetime. The above deployment will create a pod with one container from the Docker registry's [Nginx Docker Image](#).

Next, run the following command to create a service named nginx that will expose the app publicly. It will do so through a *NodePort*, a scheme that will make the pod accessible through an arbitrary port opened on each node of the cluster:

```
kubernetes-master:~$kubectl expose deploy nginx --port 80 --target-port 80 --type NodePort
```

Services are another type of Kubernetes object that expose cluster internal services to clients, both internal and external. They are also capable of load balancing requests to multiple pods, and are an integral component in Kubernetes, frequently interacting with other components.

Run the following command:

```
kubernetes-master:~$kubectl get services
```

This will output text similar to the following:

Output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	1d
nginx	NodePort	10.109.228.209	<none>	80:nginx_port/TCP	40m

From the third line of the above output, you can retrieve the port that Nginx is running on. Kubernetes will assign a random port that is greater than 30000 automatically, while ensuring that the port is not already bound by another service.

Note: if you're running your setup on ec2 ensure the nginx_port is open under the inbound rules in the security groups.

To test that everything is working, visit

`http://worker_1_ip:nginx_port`

or

`http://worker_2_ip:nginx_port`

through a browser on your local machine. You will see Nginx's familiar welcome page.

To see the deployed container on worker node switch to worker01

`on-slave#docker ps`

Output: you will see the container for nginx image running.

If you want to scale up the replicas for a deployment (nginx in our case) then use the following command:

`kubernetes-master:~$kubectl scale --current-replicas=1 --replicas=2 deployment/nginx`

`kubernetes-master:~$kubectl get pods`

Output: you will see 2/2 as output in nginx deployment.

`kubernetes-master:~$kubectl describe deployment/nginx`

Output: give details about the service deployed

If you would like to remove the Nginx application, first delete the nginx service from the master node:


```
kubernetes-master:~$kubectl delete service nginx
```

Run the following to ensure that the service has been deleted:

```
kubernetes-master:~$kubectl get services
```

You will see the following output:

Output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	1d

Then delete the deployment:

```
kubernetes-master:~$kubectl delete deployment nginx
```

Run the following to confirm that this worked:

```
kubernetes-master:~$kubectl get deployments
```

Output

No resources found.

How to gracefully remove a node from Kubernetes?

On Master Node

Find the node

```
kubernetes-master:~$kubectl get nodes
```

Drain it

```
kubernetes-master:~$kubectl drain nodetoberemoved
```

Delete it

```
kubernetes-master:~$kubectl delete node nodetoberemoved
```

On Worker Node (nodetoberemoved). Remove join/init setting from node

```
kubernetes-slave:~$kubeadm reset
```

Press y to proceed

```
kubernetes-slave:~$docker ps
```

Output: all the containers and service related to the kubernetes cluster are deleted.

```
kubernetes-master:~$kubectl get nodes
```

Output:

No worker node present.....

Conclusion

After following the steps mentioned in this article carefully, you should now have Kubernetes installed on Ubuntu.

This network uses multiple servers to communicate back and forth. Kubernetes allows you to launch and manage Docker containers across multiple servers in the pod.

References:

<https://phoenixnap.com/kb/install-kubernetes-on-ubuntu>

<https://stackoverflow.com/questions/35757620/how-to-gracefully-remove-a-node-from-kubernetes>

<https://www.digitalocean.com/community/tutorials/how-to-create-a-kubernetes-cluster-using-kubeadm-on-ubuntu-16-04>

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

Extra :

Try this link for deployment of application:

<https://kubernetes.io/docs/tutorials/stateless-application/guestbook/>

Or

<https://rancher.com/learning-paths/how-to-deploy-your-application-to-kubernetes/>

<https://www.youtube.com/watch?v=qmDzcu5uY1I>

<https://gitlab.com/nanuchi/youtube-tutorial-series/-/tree/master/kubernetes-configuration-file-explained>

<https://www.bmc.com/blogs/kubernetes-port-targetport-nodeport/>