***Note: These steps were performed using online guides and videos – on how to setup Kubernetes on Ubuntu VMs. The sources and links are mentioned at the end of the document.***

## 1. Setup Kubernetes

1. Install Docker

$ sudo apt-get update \

&& sudo apt-get install -qy docker.io

Don't upgrade the Docker version on this host. You can still build images in your CI

2. Install Kubernetes apt repo

$ sudo apt-get update \

&& sudo apt-get install -y apt-transport-https \

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

OK

$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" \

| sudo tee -a /etc/apt/sources.list.d/kubernetes.list \

&& sudo apt-get update

3. Now update your packages list with apt-get update.

1. Install kubelet, kubeadm and kubernetes-cni

* The kubelet is responsible for running containers on your hosts.
* kubeadm is a convenience utility to configure the various components that make up a working cluster
* kubernetes-cni represents the networking components.
  + [*CNI*](https://github.com/containernetworking/cni/blob/master/SPEC.md)*stands for Container Networking Interface which is a spec that defines how network drivers should interact with Kubernetes*

$ sudo apt-get update \

&& sudo apt-get install -y \

kubelet \

kubeadm \

kubernetes-cni

1. Initialize your cluster with kubeadm

* *kubeadm aims to create a secure cluster out of the box via mechanisms such as RBAC.*
* Docker Swarm provides an overlay networking driver by default - but with kubeadm this decision is left to us. The team are still working on updating their instructions - so I'll show you how to use the most similar driver to Docker's overlay driver (flannel by CoreOS).
* *Update: if you want a quick script to run in all the changes up to this point in one shot run the following:*

$ curl -sL https://gist.githubusercontent.com/alexellis/7315e75635623667c32199368aa11e95/raw/b025dfb91b43ea9309ce6ed67e24790ba65d7b67/kube.sh | sudo sh

1. **Prepare the host - notes for Kubernetes 1.8/1.9**

If you are using Kubernetes 1.7+ then the following applies:

* Swap must be disabled

You can check if you have swap enabled by typing in cat /proc/swaps. If you have a swap file or partition enabled then turn it off with swapoff. You can make this permanent by commenting out the swap file in /etc/fstab.

1. **Flannel**

* [Flannel](https://github.com/coreos/flannel) provides a software defined network (SDN) using the Linux kernel's overlay and ipvlan modules.
* *Another popular SDN offering is Weave Net by WeaveWorks.*
* Packet provides two networks for its machines - the first is a datacenter link which goes between your hosts in a specific region and project and the second faces the public Internet. There is no default firewall - if you want to lock things down you'll have to configure iptables or ufw rules manually.
* You can find your private/datacenter IP address through ifconfig:

root@kubeadm:~# ifconfig bond0:0

bond0:0 Link encap:Ethernet HWaddr 0c:c4:7a:e5:48:d4

inet addr:10.80.75.9 Bcast:255.255.255.255 Mask:255.255.255.254

UP BROADCAST RUNNING MASTER MULTICAST MTU:1500 Metric:1

1. We'll now use the internal IP address to broadcast the Kubernetes API - rather than the Internet-facing address.

*You must replace --apiserver-advertise-address with the IP of your host.*

$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=10.80.75.9 --kubernetes-version stable-1.8

* --apiserver-advertise-address determines which IP address Kubernetes should advertise its API server on.
* --pod-network-cidr is needed for the flannel driver and specifies an address space for containers
* --skip-preflight-checks allows kubeadm to check the host kernel for required features. If you run into issues where a host has the kernel meta-data removed you may need to run with this flag.
* --kubernetes-version stable-1.8 this pins the version of the cluster to 1.8, but if you want to use Kubernetes 1.7 for example - then just alter the version. Removing this flag will use whatever counts as "latest".

Here's the output we got:

[kubeadm] WARNING: kubeadm is in beta, please do not use it for production clusters.

[init] Using Kubernetes version: v1.8.1

[init] Using Authorization modes: [Node RBAC]

[preflight] Running pre-flight checks

[kubeadm] WARNING: starting in 1.8, tokens expire after 24 hours by default (if you require a non-expiring token use --token-ttl 0)

[certificates] Generated ca certificate and key.

[certificates] Generated apiserver certificate and key.

[certificates] apiserver serving cert is signed for DNS names [kubehost1 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 10.100.195.129]

[certificates] Generated apiserver-kubelet-client certificate and key.

[certificates] Generated sa key and public key.

[certificates] Generated front-proxy-ca certificate and key.

[certificates] Generated front-proxy-client certificate and key.

[certificates] Valid certificates and keys now exist in "/etc/kubernetes/pki"

[kubeconfig] Wrote KubeConfig file to disk: "admin.conf"

[kubeconfig] Wrote KubeConfig file to disk: "kubelet.conf"

[kubeconfig] Wrote KubeConfig file to disk: "controller-manager.conf"

[kubeconfig] Wrote KubeConfig file to disk: "scheduler.conf"

[controlplane] Wrote Static Pod manifest for component kube-apiserver to "/etc/kubernetes/manifests/kube-apiserver.yaml"

[controlplane] Wrote Static Pod manifest for component kube-controller-manager to "/etc/kubernetes/manifests/kube-controller-manager.yaml"

[controlplane] Wrote Static Pod manifest for component kube-scheduler to "/etc/kubernetes/manifests/kube-scheduler.yaml"

[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/manifests/etcd.yaml"

[init] Waiting for the kubelet to boot up the control plane as Static Pods from directory "/etc/kubernetes/manifests"

[init] This often takes around a minute; or longer if the control plane images have to be pulled.

[apiclient] All control plane components are healthy after 55.504048 seconds

[uploadconfig] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace

[markmaster] Will mark node kubehost1 as master by adding a label and a taint

[markmaster] Master kubehost1 tainted and labelled with key/value: node-role.kubernetes.io/master=""

[bootstraptoken] Using token: f2292a.77a85956eb6acbd6

[bootstraptoken] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials

[bootstraptoken] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token

[bootstraptoken] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster

[bootstraptoken] Creating the "cluster-info" ConfigMap in the "kube-public" namespace

[addons] Applied essential addon: kube-dns

[addons] Applied essential addon: kube-proxy

**Your Kubernetes master has initialized successfully!**

**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 f2292a.77a85956eb6acbd6 10.100.195.129:6443 --discovery-token-ca-cert-hash sha256:0c4890b8d174078072545ef17f295a9badc5e2041dc68c419880cca93d084098

1. Configure an unprivileged user-account

* Packet's Ubuntu installation ships without an unprivileged user-account, so let's add one.

$ sudo useradd packet -G sudo -m -s /bin/bash

$ sudo passwd packet

1. Configure environmental variables as the new user

* You can now configure your environment with the instructions at the end of the init message above.
* Switch into the new user account with: sudo su packet.

$ cd $HOME

$ sudo whoami

$ sudo cp /etc/kubernetes/admin.conf $HOME/

$ sudo chown $(id -u):$(id -g) $HOME/admin.conf

$ export KUBECONFIG=$HOME/admin.conf

$ echo "export KUBECONFIG=$HOME/admin.conf" | tee -a ~/.bashrc

1. Apply your pod network (flannel)

* We will now apply configuration to the cluster using kubectl and two entries from the flannel docs:

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

serviceaccount "flannel" created

configmap "kube-flannel-cfg" created

daemonset "kube-flannel-ds" created

$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifests/kube-flannel-rbac.yml

clusterrole "flannel" created

clusterrolebinding "flannel" created

* We've now configured networking for pods.

1. Allow a single-host cluster

* Kubernetes is about multi-host clustering - so by default containers cannot run on master nodes in the cluster. Since we only have one node - we'll taint it so that it can run containers for us.

$ kubectl taint nodes --all node-role.kubernetes.io/master-

*An alternative at this point would be to provision a second machine and use the join token from the output of kubeadm.*

1. Check it's working

* Many of the Kubernetes components run as containers on your cluster in a hidden namespace called kube-system. You can see whether they are working like this:

$ kubectl get all --namespace=kube-system

NAME READY STATUS RESTARTS AGE

po/etcd-kubeadm 1/1 Running 0 12m

po/kube-apiserver-kubeadm 1/1 Running 0 12m

po/kube-controller-manager-kubeadm 1/1 Running 0 13m

po/kube-dns-692378583-kqvdd 3/3 Running 0 13m

po/kube-flannel-ds-w9xvp 2/2 Running 0 1m

po/kube-proxy-4vgwp 1/1 Running 0 13m

po/kube-scheduler-kubeadm 1/1 Running 0 13m

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

svc/kube-dns 10.96.0.10 <none> 53/UDP,53/TCP 14m

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

deploy/kube-dns 1 1 1 1 14m

NAME DESIRED CURRENT READY AGE

rs/kube-dns-692378583 1 1 1 13m

As you can see all of the services are in a state of Running which indicates a healthy cluster. If these components are still being downloaded from the Internet, they may appear as not started.

## 2. Run a container

You can now run a container on your cluster. Kubernetes organizes containers into [Pods](https://kubernetes.io/docs/concepts/workloads/pods/pod/) which share a common IP address, are always scheduled on the same node (host) and can share storage volumes.

* 1. First check you have no pods (containers) running with:

$ kubectl get pods

* Now use kubectl run to deploy a container. Here supply the docker image you want tp deploy the container. In our case the image is singhalb/container-repo:part1\
* Name the deployment, here it is named: pts-app

$ kubectl run pts-app --image= singhalb/container-repo:part1--port <port number>

deployment "pts-app" created

* You'll now be able to see the Name assigned to the new Pod and when it gets started up:

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

pts-app-7d9bc5cb67-wzzx6 1/1 Running 0 4m

* Use the Name to check on the pod:

$ kubectl describe pod pts-app-7d9bc5cb67-wzzx6

...

Pulling pulling image " singhalb/container-repo:part1"

...

* A very useful feature for debugging containers is the ability to attach to the console via a shell to execute ad-hoc commands in the container:

$ **kubectl exec -it** pts-app-7d9bc5cb67-wzzx6 **--container pts-app -- /bin/bash**

## 3. Deploy any Docker container on Kubernetes

Taken from [**https://codeburst.io/getting-started-with-kubernetes-deploy-a-docker-container-with-kubernetes-in-5-minutes-eb4be0e96370**](https://codeburst.io/getting-started-with-kubernetes-deploy-a-docker-container-with-kubernetes-in-5-minutes-eb4be0e96370)

**Step 1**: Pull the image from the Repository and create a Container on the Cluster

**$ kubectl** **run** my-app **--image=**gcr.io/some-repo/my-app:v1 **--port=**3000

deployment "my-app" created

* **Note:** This application is hosted on **port 3000** so I’m opening port 3000 when I run the container, if your application doesn’t require a port just remove the port parameter.
* If you want to see this container running in your cluster, simply call for the pods. (Kubernetes groups containers together in ‘Pods’. Since our container is by itself, it’ll be in a pod of it’s own.)

**$ kubectl get pods**

NAME READY STATUS RESTARTS AGE  
my-app 1/1 Running 0 10m

* **Note:** If you check immediately after deployment, you might see the STATUS as ContainerCreating. This means K8s is still creating a Container from your image; just wait a few seconds (or minutes depending on the size of your container) and run this command again.

**Step 2:** Expose the Kubernetes Deployment through a Load Balancer

**$ kubectl expose deployment** my-app **--type=**LoadBalancer **--port=**8080 **--target-port=**3000

service "my-app" exposed

* Now we need to expose our Container with a Kubernetes Load Balancer to the world
* I’m forwarding my port exposed on the Container (which is port 3000) to port 8080 (a random pick — no particular reason to use this port. ). Just remember that the target-port and port arguments are synonymous to port forwarding 8080:3000 (port:target-port) as you’d do with docker run -p 8080:3000 my-app (external:internal)

**Step 3:** Find the external IP of your Container

* First let’s get the deployment service details from our cluster.

**$ kubectl get svc**

NAME TYPE CLUSTER-IP EXTERNAL-IP   
my-app LoadBalancer 10.11.452.237 **56.170.30.123**

* And now you know the **external IP address** of your container.
* my-app is now exposed on [http://56.170.30.123:8080](http://56.170.30.123:8080/) through a Kubernetes Load Balancer!

**(Extra) Step 4:** Use Kubernetes Rolling Updates

* Say you’ve updated this application. Do we have to go through this again to update it on the Cluster? Nope.
* I’m going to make a few changes and push a **new image** with a new **v2** tag to gcr.io. gcr.io/some-repo/my-app:v2
* We can now get K8s to update our application with just one command -

**$ kubectl set image deployment/**my-app **my-app=**gcr.io/some-repo/my-app:v2

* And that’s it — Kubernetes will pull your new image and update your current deployment through a **Rolling Update.**

**(Extra) Step 5**: Clean Up

**$ kubectl delete deployment** my-app  
**$ kubectl delete svc** my-app

## 4. Creating a deployment using yaml file

*Note: This section assumes the cluster and kubectl has been setup*

A deployment to tell Kubernetes to manage a set of replicas.

Step 1: Create the following yaml file called deployment.yaml the node where you have kubectl installed. This file describes the deployment and number of replicas and the docker image to deploy and at which port.

---

apiVersion: v1

kind: Deployment

metadata:

 name: pts-app

spec:

 replicas: 2

template:

   metadata:

     labels:

       app: web

   spec:

     containers:

       - name: pts-container

         image: singhalb/container-repo:part1

         ports:

           - containerPort: 8080

Step 2: Create the deployment from the same folder, using the following command

kubectl create -f deployment.yaml

Step 3: we can check on the deployments list:

kubectl get deployments

Step 4: check the event log by describing the Deployment, as before:

kubectl describe deployment pts-app

Step 5: Scale deployments and after that check pods if they are scaled

kubectl scale deployments/pts-app --replicas=3

deployment.extensions/pts-app scaled

kubectl get pods

NAME READY STATUS RESTARTS AGE

pts-app-7d9bc5cb67-q4lvl 0/1 ContainerCreating 0 32s

pts-app-7d9bc5cb67-wzzx6 1/1 Running 0 7m

Step 6: Enter one of the pods:

kubectl exec -it pts-app-7d9bc5cb67-q4lvl --container pts-app -- /bin/bash

Step 7: Use the phoronix test suite normally.

## 5. References

* <https://blog.alexellis.io/kubernetes-in-10-minutes/>
* <https://www.youtube.com/watch?v=6xJwQgDnMFE>
* <https://codeburst.io/getting-started-with-kubernetes-deploy-a-docker-container-with-kubernetes-in-5-minutes-eb4be0e96370>
* <https://www.mirantis.com/blog/introduction-to-yaml-creating-a-kubernetes-deployment/>
* Fixing error: <https://stackoverflow.com/questions/52119985/kubeadm-init-shows-kubelet-isnt-running-or-healthy>