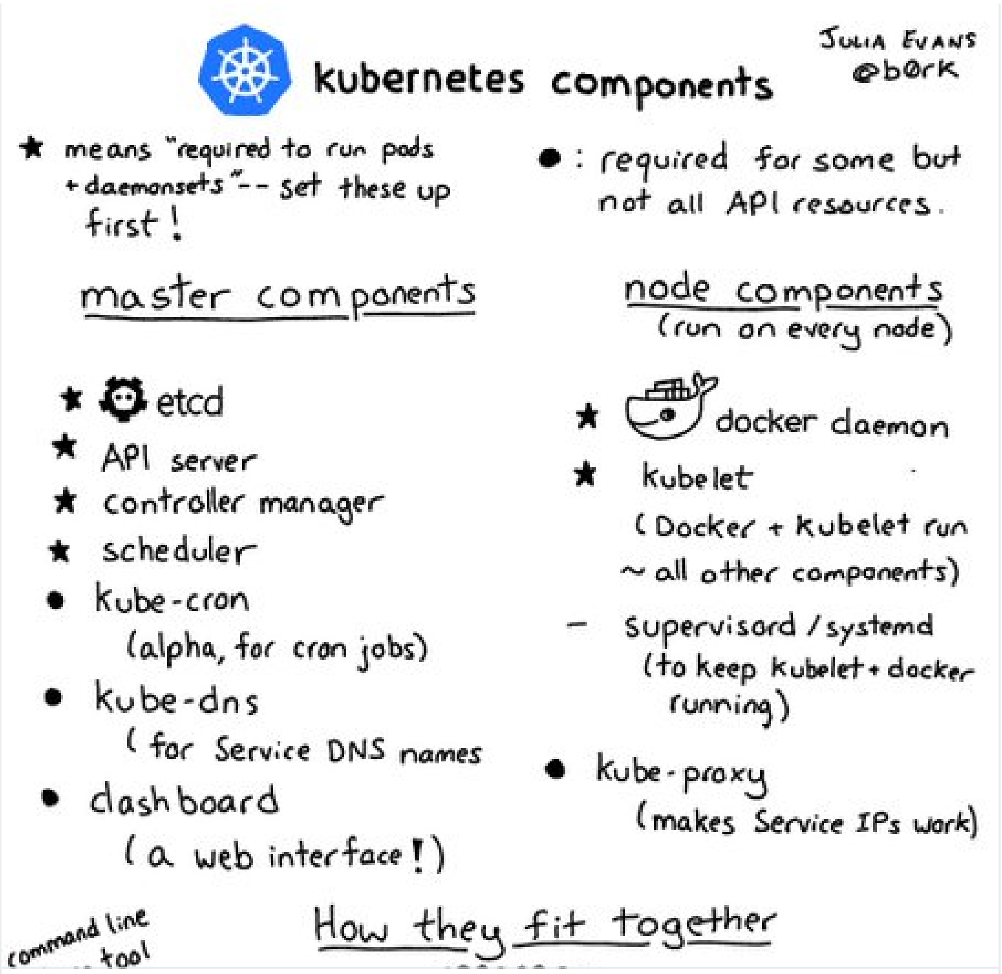
Kubernetes on bare-metal in 10 minutes

28 JUNE 2017 on docker, kubernetes, k8s, orchestration, learn-k8s

Kubernetes is an open-source container orchestration framework which was built upon the learnings of Google. It enables you to run applications using containers in a production ready-cluster. Kubernetes has many moving parts and there are countless ways to configure its pieces - from the various system components, network transport drivers, CLI utilities not to mention applications and workloads.

> *In this blog post we'll install Kubernetes 1.6 on a bare-metal machine with Ubuntu 16.04 in about 10 minutes. At the end you'll be able to start learning how to interact with Kubernetes via its CLI `kubectl`.*

Kubernetes overview:


kubernetes components

JULIA EVANS @b0rk

★ means "required to run pods + daemonsets" -- set these up first!

● : required for some but not all API resources.

master components

★ etcd
★ API server
★ controller manager
★ scheduler
● kube-cron (alpha, for cron jobs)
● kube-dns (for Service DNS names
● dashboard (a web interface!)

node components (run on every node)

★ docker daemon
★ kubelet (Docker + kubelet run ~ all other components)
– supervisord / systemd (to keep kubelet + docker running)
● kube-proxy (makes Service IPs work)

command line tool

How they fit together

🔎Julia Evans🔍
@b0rk

kubernetes components

7:57 PM - Jun 8, 2017

1,404 658 people are talking about this

Above: Kubernetes Components by Julia Evans

Pre-reqs

> *I suggest using Packet for running through the tutorial which will offer a bare-metal host - you can also run through this on a VM or your own PC if you're running Ubuntu 16.04 as your OS.*

Head over to Packet.net and create a new project. For this example we can take advantage of the Type 0 host which gives you 4x Atom cores and 8GB of RAM for $0.05/hour.

When you provision the host make sure you pick *Ubuntu 16.04* as the OS. Unlike Docker Swarm - Kubernetes is best paired with older versions of Docker. Fortunately the Ubuntu apt repository contains Docker 1.12.6.

- 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 pipe-line or on your laptop with newer versions*

Installation

- 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
```

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

- 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 and `kubernetes-cni` represents the networking components.

> *CNI 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
```

- Initialize your cluster with `kubeadm`

From the docs:

> *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/7315e75635623667
```

```
c32199368aa11e95/raw/b025dfb91b43ea9309ce6ed67e24790ba65d7b67
/kube.sh | sudo sh
```

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`.

Flannel

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. Find out more here.*

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
```

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
sha256:0c4890b8d174078072545ef17f295a9badc5e2041dc68c419880cc
a93d084098
```

- 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
```

- 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
```

- 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/Docum
entation/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/Docum
entation/k8s-manifests/kube-flannel-rbac.yml
clusterrole "flannel" created
clusterrolebinding "flannel" created
```

> *Update: the links above were changed recently by CoreOS - so I've changed them to the latest versions.*

We've now configured networking for pods.

- 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`.*

- 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.

Run a container

You can now run a container on your cluster. Kubernetes organises containers into Pods which share a common IP address, are always scheduled on the same node (host) and can share storage volumes.

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

```
$ kubectl get pods
```

Now use `kubectl run` to deploy a container. We'll deploy a Node.js and Express.js microservice that generates GUIDs over HTTP.

> *This code was originally written for a Docker Swarm tutorial and you can find the source-code there - Scale a real microservice with Docker 1.12 Swarm Mode*

```
$ kubectl run guids --image=alexellis2/guid-service:latest --
port 9000
deployment "guids" 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
guids-2617315942-lzwdh      0/1       Pending   0          11s
```

Use the `Name` to check on the pod:

```
$ kubectl describe pod guids-2617315942-lzwdh
...
Pulling                 pulling image "alexellis2/guid-
service:latest"
...
```

Once running you can get the IP address and use `curl` to generate GUIDs:

```
$ kubectl describe pod guids-2617315942-lzwdh | grep IP:
IP:             10.244.0.3

$ curl http://10.244.0.3:9000/guid ; echo
{"guid":"4659819e-cf00-4b45-
99d1a9f81bdcf6ae","container":"guids-2617315942-lzwdh"}

$ curl http://10.244.0.3:9000/guid ; echo
{"guid":"1604b4cb-88d2-49e2-bd38-
73b589da0469","container":"guids-2617315942-lzwdh"}
```

If you want to see the logs for your Pod type in:

```
$ kubectl logs guids-2617315942-lzwdh
listening on port 9000
```

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 -t -i guids-2617315942-lzwdh sh
/ # head -n3 /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.5.2
/ # exit
```

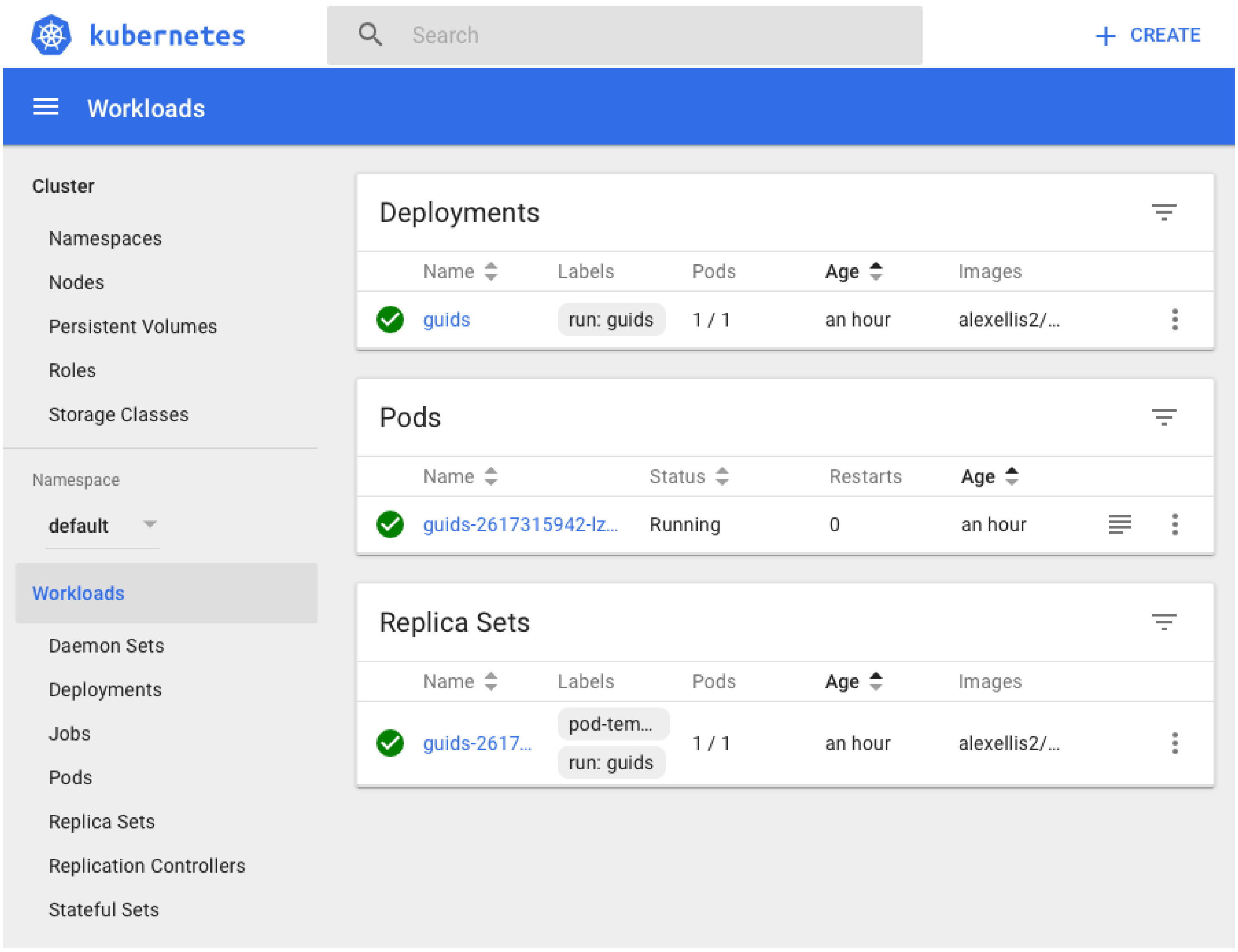
- View the Dashboard UI

The Kubernetes dashboard can be deployed as another Pod, which we can then view on our local machine. Since we did not expose Kubernetes over the Internet we'll use an SSH tunnel to view the site.

```
$ kubectl create -f https://git.io/kube-dashboard
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Now tunnel to the Packet host and navigate to http://localhost:8001/ui/ in a web-browser.

```
$ ssh -L 8001:127.0.0.1:8001 -N
```



Deployments

	Name	Labels	Pods	Age	Images
✓	guids	run: guids	1 / 1	an hour	alexellis2/...

Pods

	Name	Status	Restarts	Age
✓	guids-2617315942-lz...	Running	0	an hour

Replica Sets

	Name	Labels	Pods	Age	Images
✓	guids-2617...	pod-tem... run: guids	1 / 1	an hour	alexellis2/...

For more information on the Dashboard check it out on Github.

Wrapping up

You've now created a Kubernetes cluster and run your first micro-service. From here you can start to learn all the components that make up a cluster and explore tutorials using the `kubectl` CLI.

- Learn by example

I found Kubernetes by Example by Michael Hausenblas to be a detailed and accessible guide.

- Add more nodes

Now that you've provisioned your single-node cluster with Packet - you can go ahead and add more Type 0 nodes with the join token you got from `kubeadm`.

- Contrast to Docker Swarm

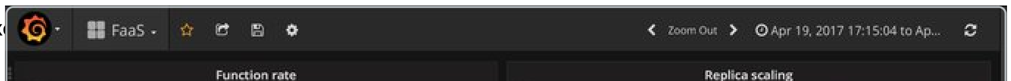
Docker Swarm is the native orchestration built into Docker's CE and EE products - you can setup a cluster in a single command. You can learn more about Swarm through my Docker Swarm tutorial series.

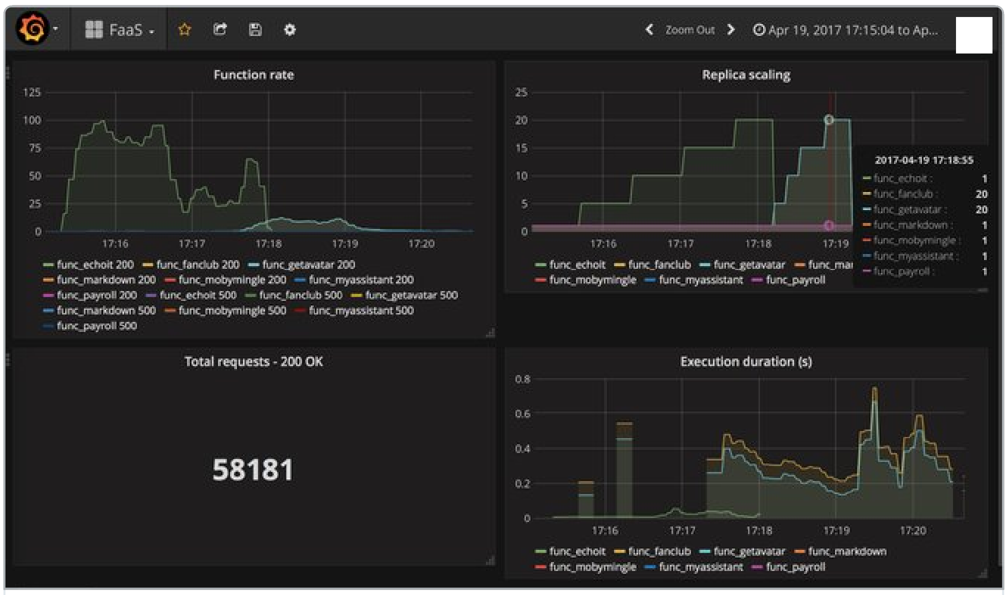
Acknowledgements:

Thanks to @mhausenblas, @__errm and @kubernetesonarm for feedback on the post and for sharing their tips on setting up a Kubernetes cluster.

You might also like

Get started with Serverless functions today





Alex Ellis
@alexellisuk

Start building #serverless functions you can run anywhere with @Docker and #FaaS - featuring the new CLI github.com/alexellis/faas

1:31 PM - Jul 10, 2017

55 36 people are talking about this

Alex Ellis

Read more posts by this author.

https://www.alexellis.io/

Share this post

Subscribe to alex ellis' blog

Get the latest posts delivered right to your inbox.

Your email address

SUBSCRIBE

or subscribe via RSS with Feedly!

30 Comments alexellis.io 1 Login

Recommend 3 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Ed Vielmetti • 8 months ago

Hey Alex, given the Kubernetes 1.7 release - http://blog.kubernetes.io/2... - are you aware off the top of your head of any changes that will need to be made to this tutorial to make it to work?

3 • Reply • Share ›

Alex Ellis Mod → Ed Vielmetti • 8 months ago

I've updated the text - it will automatically go with 1.7 now - but you can opt for 1.6. Thanks

• Reply • Share ›

Dan Todor • 3 months ago

Hey, Alex,
Thanks for the tutorial, really helpful. I looked at Packet, kinda expensive, for my tests I tend to use scaleway.com, not so many features, but I think they're less pricey, if you just want a server and no need for shared storage, by example :)

1 • Reply • Share ›

Alex Ellis Mod → Dan Todor • 3 months ago

The main point of the tutorial is to show how to setup Kubernetes with Kubeadm on Ubuntu you can run it anywhere (I think I mention this in the text). Have you looked at the Packet spotmarket? It can be cheaper than Scaleway in some instances.

• Reply • Share ›

Sudarshan Darga • 4 days ago

Hi Alex, thanks for such a nice blog. I brought up a 2 node K8s cluster following this blog.

It was working fine for me. Over the weekend I had a power failure. Post recovery I am unable to initialize my kubeadm. Can you please help me out here?

```
[kubelet-check] The HTTP call equal to 'curl -sSL
http://localhost:10255/healthz' failed with error: Get
http://localhost:10255/healthz: dial tcp 127.0.0.1:10255: getsockopt:
connection refused.
```

Unfortunately, an error has occurred:
timed out waiting for the condition

This error is likely caused by:
- The kubelet is not running
- The kubelet is unhealthy due to a misconfiguration of the node in some way (required cgroups disabled)

see more

Reply • Share ›

Alex Ellis Mod Sudarshan Darga • 4 days ago

Did you allocated a static IP as suggested in the tutorial? If not I'd suggest kubeadm reset on all nodes and bringing it back up again.

Reply • Share ›

jason welsh • 9 days ago

umm... where did

https://git.io/kube-dashboard go? It doesnt seem to be there anymore.. :(

Reply • Share ›

Alex Ellis Mod jason welsh • 9 days ago

https://github.com/kubernet...

Reply • Share ›

mpackard • 25 days ago

Good info, thanks Alex. If I have a 2nd, publicly-available ip on my server, how would I expose the Kubernetes services to that interface? This is last piece of the puzzle I have not been able to determine from the docs.

Reply • Share ›

Dirk H. Wolthuis • 2 months ago

How can I connect from my local machine with kubectl to the master? I tried using kubeadm config upload from-flags --apiserver-cert-extra-sans $PUBLIC IP and editing the ip in the files I copied from /etc/kubernetes/admin.conf. But that does not seem to work.

Reply • Share ›

Alex Ellis Mod Dirk H. Wolthuis • 2 months ago

I would think you would need to copy the certs / generated data you see from "kubeadm init" over to your local computer. This may help - https://kubernetes.io/docs/...

Reply • Share ›

Dirk H. Wolthuis Alex Ellis • 2 months ago

Those certs only work with the internal IP you specify with the kubeadm init --apiserver-advertise-address=$INTERNALIP .

Can I use a external/public IP?

Reply • Share ›

Alex Ellis Mod → Dirk H. Wolthuis • 2 months ago

You could use the external IP too yes. That will expose the API to the Internet - although it will be protected by the cert.

Reply • Share ›

Vijay Ram • 2 months ago

excellent

Reply • Share ›

Ralph • 4 months ago

Thank you Alex for writing this down. I was struggeling with 1.8 setup for such a long time. Your HOWTO was saving my life and worked out of the box :)

Reply • Share ›

Frank Squaretwo • 5 months ago

Hi, the URL you are referring to the URL https://raw.githubuserconte... which is 404. Do you know which one to use?

Reply • Share ›

Alex Ellis Mod → Frank Squaretwo • 2 months ago

Hi Frank, I think that's fixed now.

Reply • Share ›

Hai Thanh Tran Nguyen • 8 months ago

Hi Alex, thanks a lot, I have successfully install Kubernetes on my virtualbox. But just one thing cannot do is I cannot access to the Dashboard from my physical computer. Can you help me on this, thanks. Ah and one last thing :), I seem to have problem with joining node, kubeadm init has succeeded but when joinning the node got error that the token is invalid: "[discovery] Failed to connect to API Server "192.168.70.94:6443": there is no JWS signed token in the cluster-info ConfigMap. This token id "dad768" is invalid for this cluster,..."

Reply • Share ›

Alex Ellis Mod → Hai Thanh Tran Nguyen • 7 months ago

I'd suggest running the reset command kubeadm reset and starting over with it. Other than that you may want to raise an issue with kubeadm or check their docs for additional troubleshooting instructions.

Reply • Share ›

Whistl • 8 months ago

Hi Alex. I followed this process on a set of 3 VMs running on my work laptop, where we are stuck behind an HTTP_PROXY. I was able to follow most of your document to get a simple kubernetes cluster installed and running, after setting proxy environment variables in /etc/environment and /etc/systemd/system/docker.service.d/http-proxy.conf, pointing to my cntlm proxy running on my master node.

I was able to get guid and a busybox pod running, after some experimenting, but I never could get curl on the nodes, or a browser to connect to the k8s dashboard, because *something* (kubectl proxy?) kept sending my traffic to my cntlm proxy, instead of connecting directly.

Most frustrating is the NO_PROXY syntax does not accept any network ranges - you must list every IP address individually, which is impractical when k8s is using a /16 network range for containers.

After trying various configurations over a day and a half, I came away with the conclusion that I could not install Kubernetes unless Docker has it's HTTP_PROXY, HTTPS_PROXY and NO_PROXY environment variables set, but kubectl proxy does not appear to work properly when those same variables are set. It acts as if it's ignoring the NO_PROXY environment variables, and going to the proxy no matter what.

I'm going to try again on my home machine tonight, where I don't need any proxy settings.

Thanks for the write up, and big Congrats on making it to the front page of linux.com.

Patrick Wolfe

• Reply • Share ›

Alex Ellis Mod → Whistl • 8 months ago

I also face a proxy at work and use cntlm.. proxies do make life hard but sounds like you made some good progress.

• Reply • Share ›

Yuriy Levchuk → Alex Ellis • 7 months ago

Alex, I am new in Kubernetes. Could you explain please. what to do if I can't go with curl. how to override this problem in more detailed way?

• Reply • Share ›

Alex Ellis Mod → Yuriy Levchuk • 7 months ago

Hi Yuriy, thanks for the comment. What do you mean that you can't use curl? What is the error?

• Reply • Share ›

Yuriy Levchuk → Alex Ellis • 7 months ago

I mean, that after creation of the cluster I can't curl my pod. It's because of kube proxy may be. Could you explain in details what to do with it please?

• Reply • Share ›

Steve Wade • 8 months ago
did you look at trying out https://github.com/apprenda... i have done demos on Packet.net using it.
Reply • Share ›

Alex Ellis Mod → Steve Wade • 8 months ago
Looks interesting, how long did it take you to set up?
Reply • Share ›

Steve Wade → Alex Ellis • 8 months ago
It takes around the same time, myself and a colleague are going to write a blog post about it in the coming weeks.
Reply • Share ›

Alex Ellis Mod → Steve Wade • 8 months ago
Awesome - do share with me on Twitter @alexellisuk
Reply • Share ›

XaviAznar • 8 months ago
(sorry, it was done further into the article)
Reply • Share ›

Alex Ellis Mod → XaviAznar • 8 months ago

READ THIS NEXT

Serverless at Fusion meetup in Birmingham, UK

I headed to the West Midlands in the UK for my first visit to the Fusion meet-up group. I...

YOU MIGHT ENJOY

Ship Serverless FaaS functions with ease

In this post we'll look at how you can build and deploy functions quickly via the new FaaS CLI...