Kubernetes by kubeadm config yamls

Kosta Klevensky · Follow
7 min read · Jan 12, 2019

This post describes how to use `kubeadm init|join --config <config-file.yaml>` to spin up Kubernetes clusters on any hardware with any set of parameters for kubelet, api, controller, scheduler and with our own ssl certificates. We will be able to commit all cluster setting to the git repo like "Kubernetes cluster as a code"

Why kubeadm?

- kubeadm with `--config` gives almost the same flexibility as "from scratch" installation
- kubeadm and Kubernetes follow the same release cycle and deprecation policy so we can apply latest Kubernetes featutes and do not depend on 3rd party vendors like kops, acs-engine, kubespray, etc. and their *bugs and exclussions.*

Flow Summary:

- Create templates for `kubeadm init` and `kubeadm join` config files

- Using master names and endpoints we will generate the token, all needed certificates, kubeconfig and config yamls *on our admin station*
- Upload generated certificates and config file to master and run: `kubeadm init --skip-phases certs --config init-config.yaml`.
- Join node by runnig `kubeadm join --config join-config.yaml` from `cloud-init`

Lets Start

Assuming that we are able to spin up master(s) and nodes or have an auto-scaling group(s) for nodes.

0. Prerequisite: install kubeadm on admin station (or laptop)

https://kubernetes.io/docs/setup/independent/install-kubeadm/

1. kubeadm init config template

Refer kubeadm code or godoc for full spec, convert Golang struct field to config file parameter and change first letter of the field to lowercase. Get hints from these docs: control-plane-flags , kubeadm h/a , etcd h/a , kubeadm cli

This example uses `envsubst` , but for more complex automations you can use gomplate , jinja2, or others.

```yaml
apiVersion: kubeadm.k8s.io/v1beta1
kind: InitConfiguration
bootstrapTokens:
- token: "${KUBEADM_TOKEN}"
  description: "default kubeadm bootstrap token"
  ttl: "0"
localAPIEndpoint:
  advertiseAddress: ${K8S_API_ADDVERTISE_IP_1}
  bindPort: 6443
---
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
kubernetesVersion: v${K8S_VERSION}
clusterName: ${K8S_CLUSTER_NAME}
controlPlaneEndpoint: ${K8S_API_ENDPOINT_INTERNAL}:6443
certificatesDir: ${LOCAL_CERTS_DIR}
networking:
  podSubnet: 10.244.0.0/16
apiServer:
  certSANs:
  - ${K8S_API_ENDPOINT_INTERNAL}
  - ${K8S_API_ENDPOINT}

  # https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/
  extraArgs:
    max-requests-inflight: "1000"
    max-mutating-requests-inflight: "500"
    default-watch-cache-size: "500"
    watch-cache-sizes: "persistentvolumeclaims#1000,persistentvolumes#1000"

controllerManager:
  # https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-ma
  extraArgs:
    deployment-controller-sync-period: "50s"
# scheduler:
#   # https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/
#   extraArgs:
#     address: 0.0.0.0
```

kubeadm-init-config.tmpl.yaml hosted with ❤ by **GitHub** **view raw**

2. kubeadm join template

Refer to JoinConfiguration godoc for full spec of parameters.

```yaml
apiVersion: kubeadm.k8s.io/v1beta1
kind: JoinConfiguration
nodeRegistration:
  kubeletExtraArgs:
    enable-controller-attach-detach: "false"
    node-labels: "node-type=rook"
discovery:
  bootstrapToken:
    apiServerEndpoint: ${K8S_API_ENDPOINT_INTERNAL}
    token: ${KUBEADM_TOKEN}
    caCertHashes:
    - ${CA_CERT_HASH}
```

kubeadm-join-config.tmpl.yaml hosted with ❤ by **GitHub** **view raw**

3. Input Parameters

```
# export addresses and other vars
set -a
K8S_API_ENDPOINT=medium-1-api.mydomain.io
K8S_API_ENDPOINT_INTERNAL=medium-1-api-int.mydomain.io
K8S_API_ADDVERTISE_IP_1=172.16.100.10

K8S_VERSION=1.13.1
K8S_CLUSTER_NAME=medium-1

OUTPUT_DIR=$(realpath -m ./_clusters/${K8S_CLUSTER_NAME})
LOCAL_CERTS_DIR=${OUTPUT_DIR}/pki
KUBECONFIG=${OUTPUT_DIR}/kubeconfig

mkdir -p ${OUTPUT_DIR}

MASTER_SSH_ADDR_1=ubuntu@3.4.5.6
set +a
```

4. Generating kubeadm token

```
export KUBEADM_TOKEN=$(kubeadm token generate)
ghr903.k455adquq3ustxob
```

5. Applying parameters to the template

```
envsubst < kubeadm-init-config.tmpl.yaml > ${OUTPUT_DIR}/kubeadm-
init-config.yaml
```

```yaml
apiVersion: kubeadm.k8s.io/v1beta1
kind: InitConfiguration
bootstrapTokens:
- token: "ghr903.k455adquq3ustxob"
  description: "default kubeadm bootstrap token"
  ttl: "0"
localAPIEndpoint:
  advertiseAddress: 172.16.100.10
  bindPort: 6443
---
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
kubernetesVersion: v1.13.1
clusterName: medium-1
controlPlaneEndpoint: medium-1-api-int.mydomain.io:6443
certificatesDir: /home/kosta/devel/kubeadm-config/_clusters/medium-1/pki
networking:
  podSubnet: 10.244.0.0/16
apiServer:
  certSANs:
  - medium-1-api-int.mydomain.io
  - medium-1-api.mydomain.io

  # https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/
  extraArgs:
    max-requests-inflight: "1000"
    max-mutating-requests-inflight: "500"
    default-watch-cache-size: "500"
    watch-cache-sizes: "persistentvolumeclaims#1000,persistentvolumes#1000"

controllerManager:
  # https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-mai
  extraArgs:
    deployment-controller-sync-period: "50"
# scheduler:
#   # https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/
#   extraArgs:
#     address: 0.0.0.0
```

kubeadm-init-config.yaml hosted with ❤ by **GitHub** **view raw**

The simplest way is to use `kubeadm init phase certs` , but it is possible to use any other method, see pki requirements doc

```
kubeadm init phase certs all --config ${OUTPUT_DIR}/kubeadm-init-
config.yaml
```

```
[certs] Using certificateDir folder "/home/kosta/devel/kubeadm-
config/_clusters/medium-1/pki"
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names
[localhost.localdomain localhost] and IPs [172.16.100.10 127.0.0.1
::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names
[localhost.localdomain localhost] and IPs [172.16.100.10 127.0.0.1
::1]
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names
[localhost.localdomain kubernetes kubernetes.default
kubernetes.default.svc kubernetes.default.svc.cluster.local medium-1-
api-int.mydomain.io medium-1-api-int.mydomain.io medium-1-
api.mydomain.io] and IPs [10.96.0.1 172.16.100.10]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "sa" key and public key
```

All needed certificates will be in the `LOCAL_CERTS_DIR=${OUTPUT_DIR}/pki` folder. Keep the certificates and kubeconfig secure, you can use transcrypt or sops to save them in git.

7. Generate CA Certificate Hash

We need it for joining node. See this doc

```
export CA_CERT_HASH=$(openssl x509 -pubkey -in
${LOCAL_CERTS_DIR}/ca.crt | openssl rsa -pubin -outform der
2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* /sha256:/')
```

8. Generate kubeconfig for accessing cluster by public k8s endpoint

Now we are able to generate kubeconfig. Note: *we still have not done anything on servers*.

We will generate client certs with subject `/O=system:masters/CN=kubernetes-admin` signed by `$LOCAL_CERTS_DIR/{ca.crt,ca.key}` using the script below and then execute kubeconfig template.

```
#!/usr/bin/env bash
CERTS_DIR=${1:-$LOCAL_CERTS_DIR}
CA="${CERTS_DIR}"/ca.crt
CA_KEY="${CERTS_DIR}"/ca.key

if [[ ! -f ${CA} || ! -f ${CA_KEY} ]]; then
   echo "Error: CA files ${CA}  ${CA_KEY} are missing "
   exit 1
fi

CLIENT_SUBJECT=${CLIENT_SUBJECT:-"/O=system:masters/CN=kubernetes-admin"}
CLIENT_CSR=${CERTS_DIR}/kubeadmin.csr
CLIENT_CERT=${CERTS_DIR}/kubeadmin.crt
CLIENT_KEY=${CERTS_DIR}/kubeadmin.key
CLIENT_CERT_EXTENSION=${CERTS_DIR}/cert-extension

# We need faketime for cases when your client time is on UTC+
which faketime >/dev/null 2>&1
if [[ $? == 0 ]]; then
  OPENSSL="faketime -f -1d openssl"
else
  echo "Warning, faketime is missing, you might have a problem if your server time is le
  OPENSSL=openssl
fi

echo "OPENSSL = $OPENSSL "
echo "Creating Client KEY $CLIENT_KEY "
$OPENSSL genrsa -out "$CLIENT_KEY" 2048

echo "Creating Client CSR $CLIENT_CSR "
$OPENSSL req -subj "${CLIENT_SUBJECT}" -sha256 -new -key "${CLIENT_KEY}" -out "${CLIENT_

echo "--- create  ca extfile"
echo "extendedKeyUsage=clientAuth" > "$CLIENT_CERT_EXTENSION"

echo "--- sign  certificate ${CLIENT_CERT} "
$OPENSSL x509 -req -days 1096 -sha256 -in "$CLIENT_CSR" -CA "$CA" -CAkey "$CA_KEY" \
-CAcreateserial -out "$CLIENT_CERT" -extfile "$CLIENT_CERT_EXTENSION" -passin pass:"$CA_

```

generate-admin-client-certs.sh hosted with ❤ by **GitHub** **view raw**

generate-admin-client-certs.sh

```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ${CA_DATA_B64}
    server: https://${K8S_API_ENDPOINT}:6443
  name: ${K8S_CLUSTER_NAME}
contexts:
- context:
    cluster: ${K8S_CLUSTER_NAME}
    user: ${K8S_CLUSTER_NAME}-admin
    namespace: default
  name: ${K8S_CLUSTER_NAME}
current-context: ${K8S_CLUSTER_NAME}
kind: Config
preferences: {}
users:
- name: ${K8S_CLUSTER_NAME}-admin
  user:
    client-certificate-data: ${CLIENT_CERT_B64}
    client-key-data: ${CLIENT_KEY_B64}
```

kubeconfig-template.yaml hosted with ❤ by **GitHub** **view raw**

```
set -a
CLIENT_CERT_B64=$(base64 -w0  < $LOCAL_CERTS_DIR/kubeadmin.crt)
CLIENT_KEY_B64=$(base64 -w0  < $LOCAL_CERTS_DIR/kubeadmin.key)
CA_DATA_B64=$(base64 -w0  < $LOCAL_CERTS_DIR/ca.crt)
set +a
```

execute template to `KUBECONFIG=${OUTPUT_DIR}/kubeconfig`

```
envsubst < kubeconfig-template.yaml > ${OUTPUT_DIR}/kubeconfig
```

9. Install prerequisites on master

On master we need to setup docker, kubelet, kubeadm, kubectl: https://kubernetes.io/docs/setup/independent/install-kubeadm/

```
#!/bin/bash
echo "--- Installing Docker and kube"
K8S_VERSION=${K8S_VERSION:-1.13.2}

apt-get update && apt-get install -y apt-transport-https curl zip unzip
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
apt-get install -y docker.io kubelet=${K8S_VERSION}-00 kubeadm=${K8S_VERSION}-00 kubectl=

```

kubeadm-prepare-master-ubuntu-tmpl hosted with ❤ by **GitHub** **view raw**

```
envsubst < kubeadm-install-master-ubuntu-tmpl >
${OUTPUT_DIR}/prepare-master.sh
```

Execute `prepare-master.sh` on the master

```
ssh $MASTER_SSH_ADDR_1 'sudo bash -s' < ${OUTPUT_DIR}/prepare-
master.sh
```

10. Copy certificates to the master

```
tar -cz --directory=$LOCAL_CERTS_DIR . | ssh $MASTER_SSH_ADDR_1 'sudo
mkdir -p /etc/kubernetes/pki; sudo tar -xz --
directory=/etc/kubernetes/pki/'
```

11. Copy kubeadm config file to the master

Copy kubeadm config file to the master with removing `certificatesDir` that points to $LOCAL_CERTS_DIR

```
sed '/certificatesDir:/d' $OUTPUT_DIR/kubeadm-init-config.yaml | ssh
$MASTER_SSH_ADDR_1 sudo dd of=/root/kubeadm-init-config.yaml
```

12. Run `kubeadm init` without certs phase

```
ssh $MASTER_SSH_ADDR_1 sudo kubeadm init --skip-phases certs --config
/root/kubeadm-init-config.yaml
```

— —

```
[init] Using Kubernetes version: v1.13.1
[preflight] Running pre-flight checks
        [WARNING Service-Docker]: docker service is not enabled,
please run 'systemctl enable docker.service'
[preflight] Pulling images required for setting up a Kubernetes
cluster
[preflight] This might take a minute or two, depending on the speed
of your internet connection
[preflight] You can also perform this action in beforehand using
'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-
manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in
"/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control
plane as static Pods from directory "/etc/kubernetes/manifests". This
can take up to 4m0s
```

```
[apiclient] All control plane components are healthy after 21.521772
seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-
config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.13" in namespace
kube-system with the configuration for the kubelets in the cluster
[patchnode] Uploading the CRI Socket information
"/var/run/dockershim.sock" to the Node API object "ip-172-16-100-10"
as an annotation
[mark-control-plane] Marking the node ip-172-16-100-10 as control-
plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node ip-172-16-100-10 as control-
plane by adding the taints [node-
role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: ghr903.k455adquq3ustxob
[bootstrap-token] Configuring bootstrap tokens, cluster-info
ConfigMap, RBAC Roles
[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: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following 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:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

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

kubeadm join medium-1-api-int.mydomain.io:6443 --token
ghr903.k455adquq3ustxob --discovery-token-ca-cert-hash sha256:...
```

13. Ensure that it is running

We have already generated kubeconfig on our local admin station.

```
export KUBECONFIG=$OUTPUT_DIR/kubeconfig
kubectl get pods --all-namespaces
```

```
NAMESPACE     NAME                                        READY
STATUS    RESTARTS   AGE
kube-system   coredns-86c58d9df4-2sbxq                    0/1
Pending   0          1h
kube-system   coredns-86c58d9df4-75nmt                    0/1
Pending   0          1h
kube-system   etcd-ip-172-16-100-10                       1/1
Running   0          1h
kube-system   kube-apiserver-ip-172-16-100-10             1/1
Running   0          1h
kube-system   kube-controller-manager-ip-172-16-100-10    1/1
Running   0          1h
kube-system   kube-proxy-ll6fq                            1/1
Running   0          1h
kube-system   kube-scheduler-ip-172-16-100-10             1/1
Running   0          1h
```

14. Installing Pod Network

See https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/#pod-network .

We will save the applied yaml in our $OUTPUT_DIR for future reference. For flannel it will look like this:

```
curl -s --output $OUTPUT_DIR/kube-flannel.yaml
https://raw.githubusercontent.com/coreos/flannel/bc79dd1505b0c8681ece
4de4c0d86c5cd2643275/Documentation/kube-flannel.yml
```

```
kubectl apply -f $OUTPUT_DIR/kube-flannel.yaml
```

```
clusterrole.rbac.authorization.k8s.io "flannel" created
clusterrolebinding.rbac.authorization.k8s.io "flannel" created
serviceaccount "flannel" created
configmap "kube-flannel-cfg" created
daemonset.extensions "kube-flannel-ds-amd64" created
```

```
daemonset.extensions "kube-flannel-ds-arm64" created
daemonset.extensions "kube-flannel-ds-arm" created
daemonset.extensions "kube-flannel-ds-ppc64le" created
daemonset.extensions "kube-flannel-ds-s390x" created
```

We have already set all needed kernel params and `podSubnet: 10.244.0.0/16` Ensure that master node appears as Ready.

```
kubectl get nodes
NAME              STATUS    ROLES     AGE       VERSION
ip-172-16-0-185   Ready     master    1h        v1.13.1
```

15. Joining Nodes

Generally, we need to install docker.io, kubeadm and kubelet on node, execute the `kubeadm-join-config.tmpl.yaml` template we created in (3) and then `kubeadm join --config kubeadm-join-config.yaml`

To join node by `cloud-init` enter into the user-data a script like this:

kubeadm join config from cloud-init user-data

And the nodes will be joined:

```
kubectl get nodes
NAME                STATUS    ROLES     AGE       VERSION
ip-172-16-100-158   Ready     <none>    11s       v1.13.1
ip-172-16-100-121   Ready     <none>    28s       v1.13.1
ip-172-16-100-10    Ready     master    9h        v1.13.1
```

What's next

To upgrade the cluster refer to this doc. Set new `kubernetesVersion` and run `kubeadm upgrade apply --config init-config.yaml`

To change configuration without an actual upgrade just add the `--force` flag:

```
kubeadm upgrade apply --config init-config.yaml --force
```

Conclusion

Using this `kubeadm --config` approach we can template any Kubernetes configuration and save all the configuration data about the cluster. Some hints are below:

- To use cloud-specific Kubernetes features, add `cloud-provider` , `cloud-config` and `extraVolumes` parameters to kubelet, kube-api, controller and scheduler. See cloud providers docs . Assign appropriative iam role to master nodes.
- To create H/A master, K8S_API_ENDPOINT and K8S_API_ENDPOINT_INTERNAL should be set to load balancers addresses. We will need three parameter sets for masters and execute kubeadm-init-config templates 3 times. See kubeadm h/a , etcd h/a.
- Never commit cluster data unencrypted. Use transcrypt , sops , vaults and .gitignore to avoid accidental commits.

Kubernetes Kubeadm Cloud Agnostic Infrastructure As Code Gitops

Written by Kosta Klevensky

28 Followers · 3 Following

System architect and developer

Follow

Responses (2)

What are your thoughts?

Respond

Alex Flom
Apr 11, 2020

This article is amazing. I haven't seen any other resource out there that provides this level of detail of the end-to-end automated kubeadm bootstrapping/join process. Thank you.

Reply

David Sanders
Jul 12, 2019

Thanks for this article Kosta! This really helped me when I was working through using a config file with kubeadm. While I didn't follow every step, it was really helpful in getting my provisioner setup to use the config file instead of passing arguments.

Thank you :)

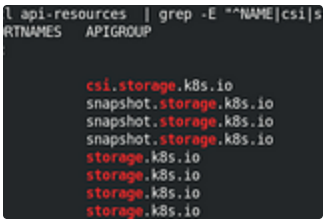
Reply

More from Kosta Klevensky

Kosta Klevensky

Kubernetes CSI in action

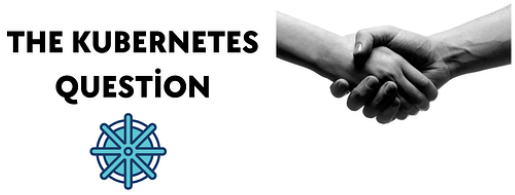
This post we will dig into Kubernetes Container Storage Interface. We will install CSI Driver for Amazon EBS and see what really...



May 6, 2019 75

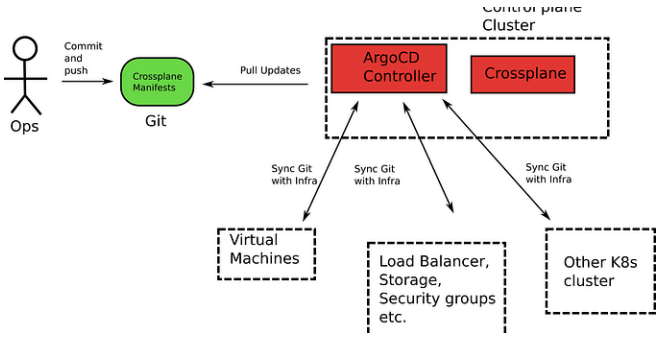
See all from Kosta Klevensky

Recommended from Medium



In Level Up Coding by Rahul Sharma

I Asked This Kubernetes Question in Every Interview — And Here's th...



Mr.PlanB

Simplifying Kubernetes Infrastructure with Crossplane an...

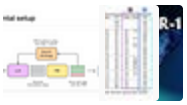
When I interview candidates, I prefer a real-world problem that demonstrates the...

Jan 15 689 19

In the modern cloud-native landscape, deploying and managing infrastructure...

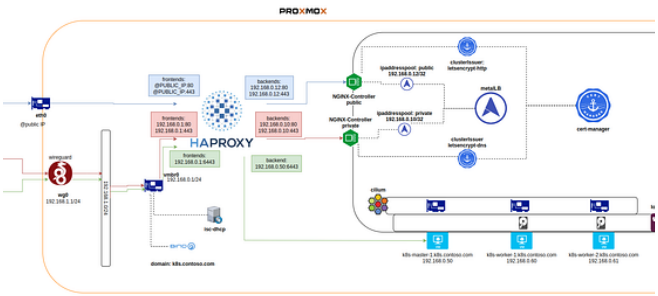
Nov 7, 2024 5

Lists



Natural Language Processing

1899 stories · 1556 saves

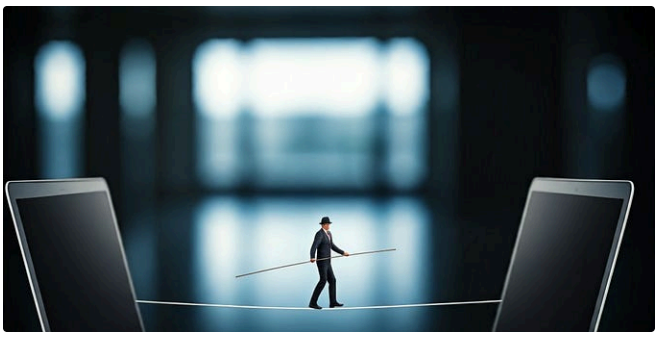


In overcast blog by Genesta Sebastien

Kubernetes—Kubernetes cluster deployment on Proxmox 8—Part ...

This article deals with how to deploy Proxmox hosted on an OVH dedicated server to set u...

Dec 18, 2024 25



Tyler Auerbeck

MetalLB and KinD: Loads Balanced Locally

When You Need LoadBalancer Services On The Go, MetalLB and KinD Are There For You

Sep 24, 2024 7

Emircan Agac

Calico vs. Cilium: Choosing the Best CNI for Your Kubernetes Cluster

Choosing the Best CNI: Calico vs. Cilium

Aug 29, 2024 34 1

Rajesh k

Kubernetes Operators vs. Helm Charts: Understanding the...

As Kubernetes continues to evolve as the de facto standard for container orchestration,...

Aug 4, 2024 2

See more recommendations