Deploy Kubernetes 1.9 from scratch on VMware vSphere

29 December 2017

This lab will go through the different steps needed to configure an HA Kubernetes cluster on VMware vSphere manually. The various communications between the Kubernetes components will be secured with TLS. If you are used to deploy Kubernetes with tools like kubeadm but would like to understand a bit more what is going on under the hood, this tutorial is for you.

However, it is good to have an understanding of the architecture of each Kubernetes node as described in the Kubernetes components documentation (https://kubernetes.io/docs/concepts/overview/components/).

This article is inspired by the awesome tutorial Kubernetes the hard way (https://github.com/kelseyhightower/kubernetes-the-hard-way), which explains how to deploy a Kubernetes cluster from scratch on Google Cloud Platform. Some of the steps are a bit different on VMware vSphere. For example, VMware vSphere doesn't provide a native load balancer and we will have to install and configure one ourselves.

Prerequisites

For this lab, we will use a standard Ubuntu 16.04 installation as a base image for the seven virtual machines needed. The virtual machines will all be configured on the same virtual network 10.10.40.0/24 and this network needs to have access to the Internet.

The first machine needed is the machine on which the HAProxy load balancer will be installed. We will assign the IP 10.10.40.63 to this machine.

We also need three Kubernetes master nodes. These virtual machines will have the IPs 10.10.40.60, 10.10.40.61, and 10.10.40.62.

Finally, we will also have three Kubernetes worker nodes with the IPs 10.10.40.70, 10.10.40.71, and 10.10.40.72.

We also need an IP range for the pods. This range will be 10.20.0.0/16, but it is only internal to Kubernetes and doesn't need to be configured on VMware vSphere.

I will use my Linux desktop as a client machine to generate all the necessary certificates, but also to manage the Kubernetes cluster. If you don't have a Linux desktop, you can use the HAProxy virtual machine to do the same thing.

Installation of the client tools

We will need two tools on the client machine. The Cloud Flare SSL tool to generate the different certificates, and the Kubernetes client, kubectl, to manage the Kubernetes cluster.

Installation of cfssl

1- Download the binaries.

```
$ wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
```

```
$ wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
```

2- Add the execution permission to the binaries.

```
$ chmod +x cfssl*
```

3- Move the binaries to /usr/local/bin.

```
$ sudo mv cfssl_linux-amd64 /usr/local/bin/cfssl
```

```
$ sudo mv cfssljson_linux-amd64 /usr/local/bin/cfssljson
```

4- Verify the installation.

```
$ cfssl version
```

Installation of kubectl

- Download the binary.

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

2- Add the execution permission to the binary.

```
$ chmod +x kubectl
```

3- Move the binary to /usr/local/bin.

```
$ sudo mv kubectl /usr/local/bin
```

4- Verify the installation.

```
$ kubectl version
```

Installation of the HAProxy load balancer

As we will deploy three Kubernetes master nodes, we need to deploy an HAPRoxy load balancer in front of them to distribute the traffic.

1- SSH to the 10.10.40.63 Ubuntu virtual machine.

2- Update the virtual machine.

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

3- Install HAProxy.

```
$ sudo apt-get install haproxy
```

4- Configure HAProxy to load balance the traffic between the three Kubernetes master nodes.

```
$ sudo vim /etc/haproxy/haproxy.cfg
global
...
default
...
frontend kubernetes
    bind        10.10.40.63:6443
    option      tcplog
    mode        tcp
    default_backend kubernetes-master-nodes



backend kubernetes-master-nodes
    mode    tcp
    balance roundrobin
    option  tcp-check
    server  k8s-master-0 10.10.40.60:6443 check fall 3 rise 2
    server  k8s-master-1 10.10.40.61:6443 check fall 3 rise 2
    server  k8s-master-2 10.10.40.62:6443 check fall 3 rise 2
```

5- Restart HAProxy.

```
$ sudo systemctl restart haproxy
```

Generate the TLS certificates

These steps can be done on your Linux desktop if you have one or on the HAProxy virtual machine depending on where you installed the cfssl tool.

Create a certificate authority

1- Create the certificate authority configuration file.

```
$ vim ca-config.json
{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": ["signing", "key encipherment", "server auth", "client auth"]
        "expiry": "8760h"
      }
    }
  }
}
```

2- Create the certificate authority signing request configuration file.

```
$ vim ca-csr.json
{
  "CN": "Kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "IE",
      "L": "Cork",
      "O": "Kubernetes",
      "OU": "CA",
      "ST": "Cork Co."
    }
  ]
}
```

3- Generate the certificate authority certificate and private key.

```
$ cfssl gencert -initca ca-csr.json | cfssljson -bare ca
```

4- Verify that the ca-key.pem and the ca.pem were generated.

```
$ ls -la
```

Generate the Kubernetes certificates

Each Kubernetes component will need a client and a server certificate to communicate over TLS. We will generate all these certificates in this section.

Admin client certificate

This certificate will be used to connect to the Kubernetes cluster as an administrator.

1- Create the certificate signing request configuration file.

```
$ vim admin-csr.json
{
  "CN": "admin",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "IE",
      "L": "Cork",
      "O": "system:masters",
      "OU": "Kubernetes",
      "ST": "Cork Co."
    }
  ]
}
```

2- Generate the certificate and the private key.

```
$ cfssl gencert \
-ca=ca.pem \
-ca-key=ca-key.pem \
-config=ca-config.json \
-profile=kubernetes admin-csr.json | \
cfssljson -bare admin
```

3- Verify that the admin-key.pem and the admin.pem were generated.

```
$ ls -la
```

Generate the Kubelet client certificates

We will deploy a kubelet on each Kubernetes worker node. Each of these kubelets will need a certificate to join the Kubernetes cluster.

1- Create a certificate signing request configuration file for the 10.10.40.70 worker node.

```
$ vim 10.10.40.70-csr.json
{
  "CN": "system:node:10.10.40.70",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "IE",
      "L": "Cork",
      "O": "system:nodes",
      "OU": "Kubernetes",
      "ST": "Cork Co."
    }
  ]
}
```

2- Generate the certificate and the private key.

```
$ cfssl gencert \
-ca=ca.pem \
-ca-key=ca-key.pem \
-config=ca-config.json \
-hostname=10.10.40.70,10.10.40.70 \
-profile=kubernetes 10.10.40.70-csr.json | \
cfssljson -bare 10.10.40.70
```

3- Create a certificate signing request configuration file for the 10.10.40.71 worker node.

```
$ vim 10.10.40.71-csr.json
{
  "CN": "system:node:10.10.40.71",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "IE",
      "L": "Cork",
      "O": "system:nodes",
      "OU": "Kubernetes",
      "ST": "Cork Co."
    }
  ]
}
```

4- Generate the certificate and the private key.

```
$ cfssl gencert \
-ca=ca.pem \
-ca-key=ca-key.pem \
-config=ca-config.json \
-hostname=10.10.40.71,10.10.40.71 \
-profile=kubernetes 10.10.40.71-csr.json | \
cfssljson -bare 10.10.40.71
```

5- Create a certificate signing request configuration file for the 10.10.40.72.

```
$ vim 10.10.40.72-csr.json
{
  "CN": "system:node:10.10.40.72",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "IE",
      "L": "Cork",
      "O": "system:nodes",
      "OU": "Kubernetes",
      "ST": "Cork Co."
    }
  ]
}
```

6- Generate the certificate and the private key.

```
$ cfssl gencert \
-ca=ca.pem \
-ca-key=ca-key.pem \
-config=ca-config.json \
-hostname=10.10.40.72,10.10.40.72 \
-profile=kubernetes 10.10.40.72-csr.json | \
cfssljson -bare 10.10.40.72
```

7- Verify that the files 10.10.40.70-key.pem, 10.10.40.70.pem, 10.10.40.71-key.pem, 10.10.40.71.pem, 10.10.40.72-key.pem, and 10.10.40.72.pem were generated.

```
$ ls -la
```

Generate the kube-proxy client certificate

Another component that we will install on all our worker nodes is the kube-proxy. The kube-proxy also needs a client certificate.

1- Create the certificate signing request configuration file.

```
$ vim kube-proxy-csr.json
{
  "CN": "system:kube-proxy",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "IE",
      "L": "Cork",
      "O": "system:node-proxier",
      "OU": "Kubernetes",
      "ST": "Cork Co."
    }
  ]
}
```

2- Generate the kube-proxy certificate and private key.

```
$ cfssl gencert \
-ca=ca.pem -ca-key=ca-key.pem \
-config=ca-config.json \
-profile=kubernetes kube-proxy-csr.json | \
cfssljson -bare kube-proxy
```

3- Verify that the kube-proxy-key.pem and the kube-proxy.pem files were generated.

```
$ ls -la
```

Generate the API server certificate

The last certificate we need to generate is the API server certificate. The API component will be installed on each of our Kubernetes master nodes.

1- Create the certificate signing request configuration file.

```
$ vim ca-config.json
{
  "CN": "kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "IE",
      "L": "Cork",
      "O": "Kubernetes",
      "OU": "Kubernetes",
      "ST": "Cork Co."
    }
  ]
}
```

2- Generate the API server certificate and private key.

```
$ cfssl gencert \
-ca=ca.pem \
-ca-key=ca-key.pem \
-config=ca-config.json \
-hostname=10.32.0.1,10.10.40.60,10.10.40.61,10.10.40.62,10.10.40.63,127.0.0.1,k
-profile=kubernetes kubernetes-csr.json | \
cfssljson -bare kubernetes
```

3- Verify that the kubernetes-key.pem and the kubernetes.pem file were generated.

```
$ ls -l
```

Copy the certificates to the nodes

Worker nodes

1- Copy the certificates to the 10.10.40.70 worker node.

```
$ scp ca.pem 10.10.40.70-key.pem 10.10.40.70.pem 10.10.40.70:~
```

2- Copy the certificates to the 10.10.40.71 worker node.

```
$ scp ca.pem 10.10.40.71-key.pem 10.10.40.71.pem 10.10.40.71:~
```

3- Copy the certificates to the 10.10.40.72 worker node.

```
$ scp ca.pem 10.10.40.72-key.pem 10.10.40.72.pem 10.10.40.72:~
```

Master nodes

1- Copy the certificates to 10.10.40.60.

```
$ scp ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem 10.10.40.60:~
```

2- Copy the certificates to 10.10.40.61.

```
$ scp ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem 10.10.40.61:~
```

3- Copy the certificates to the 10.10.40.62.

```
$ scp ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem 10.10.40.62:~
```

Generate the kubeconfig file for the worker nodes

We need to configure the kubelet and the kube-proxy with a kubeconfig file on each worker node to allow them to access the Kubernetes API.

Generate kubeconfig file for the kubelets

1- Add the cluster information for 10.10.40.70.

```
$ kubectl config set-cluster kubernetes \
--certificate-authority=ca.pem \
--embed-certs=true \
--server=https://10.10.40.63:6443 (https://10.10.40.63:6443) \
--kubeconfig=10.10.40.70.kubeconfig
```

2- Add the credentials for 10.10.40.70.

```
$ kubectl config set-credentials system:node:10.10.40.70 \
--client-certificate=10.10.40.70.pem \
--client-key=10.10.40.70-key.pem \
--embed-certs=true \
--kubeconfig=10.10.40.70.kubeconfig
```

3- Add the context for 10.10.40.70.

```
$ kubectl config set-context default \
--cluster=kubernetes \
--user=system:node:10.10.40.70 \
--kubeconfig=10.10.40.70.kubeconfig
```

4- Use the context for 10.10.40.70.

```
$ kubectl config use-context default --kubeconfig=10.10.40.70.kubeconfig
```

5- Add the cluster information for 10.10.40.71.

```
$ kubectl config set-cluster kubernetes \
--certificate-authority=ca.pem \
--embed-certs=true \
--server=https://10.10.40.63:6443 (https://10.10.40.63:6443) \
--kubeconfig=10.10.40.71.kubeconfig
```

6- Add the credentials for 10.10.40.71.

```
$ kubectl config set-cluster kubernetes --server=10.10.40.71 \
--client-certificate=10.10.40.71.pem \
--client-key=10.10.40.71-key.pem \
--embed-certs=true \
--kubeconfig=10.10.40.71.kubeconfig
```

7- Add the context for 10.10.40.71.

```
$ kubectl config set-context default \
--cluster=kubernetes \
--user=system:node:10.10.40.71 \
--kubeconfig=10.10.40.71.kubeconfig
```

8- Use the context for 10.10.40.71.

```
$ kubectl config use-context default --kubeconfig=10.10.40.71.kubeconfig
```

9- Add the cluster information for 10.10.40.72.

```
$ kubectl config set-cluster kubernetes \
--certificate-authority=ca.pem \
--embed-certs=true \
--server=https://10.10.40.63:6443 (https://10.10.40.63:6443) \
--kubeconfig=10.10.40.72.kubeconfig
```

10- Add the credentials for 10.10.40.72.

```
$ kubectl config set-cluster kubernetes \
--certificate-authority=ca.pem \
--embed-certs=true \
--server=https://10.10.40.72:6443 \
--client-certificate=10.10.40.72.pem \
--client-key=10.10.40.72-key.pem \
--embed-certs=true \
--kubeconfig=10.10.40.72.kubeconfig
```

11- Add the context for 10.10.40.72.

```
$ kubectl config set-context default \
--cluster=kubernetes \
--user=system:node:10.10.40.72 \
--kubeconfig=10.10.40.72.kubeconfig
```

12- Use the context for 10.10.40.72.

```
$ kubectl config use-context default --kubeconfig=10.10.40.72.kubeconfig
```

Generate the kubeconfig file for the kube-proxies

1- Add the cluster information.

```
$ kubectl config set-cluster kubernetes \
--certificate-authority=ca.pem \
--embed-certs=true \
--server=https://10.10.40.63:6443 (https://10.10.40.63:6443) \
--kubeconfig=kube-proxy.kubeconfig
```

2-Add the credentials.

```
$ kubectl config set-cluster kubernetes \
--client-certificate=kube-proxy.pem \
--client-key=kube-proxy-key.pem \
--embed-certs=true \
--kubeconfig=kube-proxy.kubeconfig
```

3- Add the context information.

```
$ kubectl config set-context default \
--cluster=kubernetes \
--user=kube-proxy \
--kubeconfig=kube-proxy.kubeconfig
```

4- Use the context.

```
$ kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig
```

Copy the kubeconfig configurations to the worker nodes

1- Copy the configuration to 10.10.40.70.

```
$ scp 10.10.40.70.kubeconfig kube-proxy.kubeconfig 10.10.40.70:~
```

2- Copy the configuration to 10.10.40.71.

```
$ scp 10.10.40.71.kubeconfig kube-proxy.kubeconfig 10.10.40.71:~
```

3- Copy the configuration to 10.10.40.72

```
$ scp 10.10.40.72.kubeconfig kube-proxy.kubeconfig 10.10.40.72:~
```

Generate the data encryption key for secrets

Kubernetes is able to store secrets for us in a key value store. To encrypt this data before storing it, we need an encryption key.

1- Generate a random encryption key.

```
$ ENCRYPTION_KEY=$(head -c 32 /dev/urandom | base64)
```

2- Create a Kubernetes manifest for the encryption configuration.

```
$ cat > encryption-config.yaml <<EOF
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: ${ENCRYPTION_KEY}
      - identity: {}
EOF
```

3- Copy the manifest to 10.10.40.60.

```
$ scp encryption-config.yaml 10.10.40.60:~
```

4- Copy the manifest to 10.10.40.61.

```
$ scp encryption-config.yaml 10.10.40.61:~
```

5- Copy the manifest to 10.10.40.62.

```
$ scp encryption-config.yaml 10.10.40.62:~
```

Create the etcd cluster

One of the main components of the master nodes is the etcd cluster. Etcd is a key value store and is used to store the Kubernetes cluster state. We are going to configure one etcd node per master. With this configuration, the Kubernetes cluster will still be available if one of the etcd fails.
Install etcd on the 10.10.40.60 master node

1- SSH to the 10.10.40.60 virtual machine.

2- Download the etcd binaries.

```
$ wget https://github.com/coreos (http://github.com/coreos)/etcd/releases/downl
```

3- Extract the etcd archive.

```
$ tar xvzf etcd-v3.2.11-linux-amd64.tar.gz
```

4- Move the etcd binaries to /usr/local/bin.

```
$ sudo mv etcd-v3.2.11-linux-amd64/etcd* /usr/local/bin/
```

5- Create the configuration directories.

```
$ sudo mkdir -p /etc/etcd /var/lib/etcd
```

6- Copy the certificates to the /etc/etcd configuration directory.

```
$ sudo cp ca.pem kubernetes-key.pem kubernetes.pem /etc/etcd/
```

7- Create an etcd systemd unit file.

```
$ sudo vim /etc/systemd/system/etcd.service
[Unit]
Description=etcd
Documentation=https://github.com/coreos (http://github.com/coreos)



[Service]
ExecStart=/usr/local/bin/etcd \
  --name 10.10.40.60 \
  --cert-file=/etc/etcd/kubernetes.pem \
  --key-file=/etc/etcd/kubernetes-key.pem \
  --peer-cert-file=/etc/etcd/kubernetes.pem \
  --peer-key-file=/etc/etcd/kubernetes-key.pem \
  --trusted-ca-file=/etc/etcd/ca.pem \
  --peer-trusted-ca-file=/etc/etcd/ca.pem \
  --peer-client-cert-auth \
  --client-cert-auth \
  --initial-advertise-peer-urls https://10.10.40.60:2380 \
  --listen-peer-urls https://10.10.40.60:2380 \
  --listen-client-urls https://10.10.40.60:2379,http://127.0.0.1:2379
  --advertise-client-urls https://10.10.40.60:2379 \
  --initial-cluster-token etcd-cluster-0 \
  --initial-cluster 10.10.40.60=https://10.10.40.60:2380,10.10.40.61=h
  --initial-cluster-state new \
```

```
    --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

8- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

9- Enable etcd to start at boot time.

```
$ sudo systemctl enable etcd
```

10- Start etcd.

```
$ sudo systemctl start etcd
```

Install etcd on the 10.10.40.61 master node

1- SSH to the 10.10.40.61 virtual machine.

2- Download the etcd binaries.

```
$ wget https://github.com/coreos (http://github.com/coreos)/etcd/releases/downl
```

3- Extract the etcd archive.

```
$ tar xvzf etcd-v3.2.11-linux-amd64.tar.gz
```

4- Move the etcd binaries to /usr/local/bin.

```
$ sudo mv etcd-v3.2.11-linux-amd64/etcd* /usr/local/bin/
```

5- Create the configuration directories.

```
$ sudo mkdir -p /etc/etcd /var/lib/etcd
```

6- Copy the certificates to the /etc/etcd configuration directory.

```
$ sudo cp kubernetes-key.pem kubernetes.pem /etc/etcd/
```

(http://blog.inkubate.io)

7- Create an etcd systemd unit file.

```
$ sudo vim /etc/systemd/system/etcd.service
[Unit]
Description=etcd
Documentation=https://github.com/coreos (http://github.com/coreos)



[Service]
ExecStart=/usr/local/bin/etcd \
  --name 10.10.40.61 \
  --cert-file=/etc/etcd/kubernetes.pem \
  --key-file=/etc/etcd/kubernetes-key.pem \
  --peer-cert-file=/etc/etcd/kubernetes.pem \
  --peer-key-file=/etc/etcd/kubernetes-key.pem \
  --trusted-ca-file=/etc/etcd/ca.pem \
  --peer-trusted-ca-file=/etc/etcd/ca.pem \
  --peer-client-cert-auth \
  --client-cert-auth \
  --initial-advertise-peer-urls https://10.10.40.61:2380 \
  --listen-peer-urls https://10.10.40.61:2380 \
  --listen-client-urls https://10.10.40.61:2379,http://127.0.0.1:2379
  --advertise-client-urls https://10.10.40.61:2379 \
  --initial-cluster-token etcd-cluster-0 \
  --initial-cluster 10.10.40.60=https://10.10.40.60:2380,10.10.40.61=h
  --initial-cluster-state new \
```

```
    --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

8- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

9- Enable etcd to start at boot time.

```
$ sudo systemctl enable etcd
```

10- Start etcd.

```
$ sudo systemctl start etcd
```

Install etcd on the 10.10.40.62 master node

1- SSH to the 10.10.40.62 virtual machine.

2- Download the etcd binaries.

```
$ wget https://github.com/coreos (http://github.com/coreos)/etcd/releases/downl
```

3- Extract the etcd archive.

```
$ tar xvzf etcd-v3.2.11-linux-amd64.tar.gz
```

4- Move the etcd binaries to /usr/local/bin.

```
$ sudo mv etcd-v3.2.11-linux-amd64/etcd* /usr/local/bin/
```

5- Create the configuration directories.

```
$ sudo mkdir -p /etc/etcd /var/lib/etcd
```

6- Copy the certificates to the /etc/etcd configuration directory.

```
$ sudo cp kubernetes-key.pem kubernetes.pem /etc/etcd/
```

(http://blog.inkubate.io)

7- Create an etcd systemd unit file.

```
$ sudo vim /etc/systemd/system/etcd.service
[Unit]
Description=etcd
Documentation=https://github.com/coreos (http://github.com/coreos)



[Service]
ExecStart=/usr/local/bin/etcd \
  --name 10.10.40.62 \
  --cert-file=/etc/etcd/kubernetes.pem \
  --key-file=/etc/etcd/kubernetes-key.pem \
  --peer-cert-file=/etc/etcd/kubernetes.pem \
  --peer-key-file=/etc/etcd/kubernetes-key.pem \
  --trusted-ca-file=/etc/etcd/ca.pem \
  --peer-trusted-ca-file=/etc/etcd/ca.pem \
  --peer-client-cert-auth \
  --client-cert-auth \
  --initial-advertise-peer-urls https://10.10.40.62:2380 \
  --listen-peer-urls https://10.10.40.62:2380 \
  --listen-client-urls https://10.10.40.62:2379,http://127.0.0.1:2379
  --advertise-client-urls https://10.10.40.62:2379 \
  --initial-cluster-token etcd-cluster-0 \
  --initial-cluster 10.10.40.60=https://10.10.40.60:2380,10.10.40.61=h
  --initial-cluster-state new \
```

```
    --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

8- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

9- Enable etcd to start at boot time.

```
$ sudo systemctl enable etcd
```

10- Start etcd.

```
$ sudo systemctl start etcd
```

11- Verify that the cluster is up and running.

```
$ ETCDCTL_API=3 etcdctl member list
```

Install the Kubernetes master nodes components

The Kubernetes master nodes need to run three other components: the API, the controller manager, and the scheduler. We will also install a Kubernetes client, kubectl, on each node.

Installation of the 10.10.40.60 master node

1- SSH to 10.10.40.60.

2- Download the binaries.

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

3- Add the execution permissions to the binaries.

```
$ sudo mv \
kube-apiserver \
kube-controller-manager \
kube-scheduler \
kubectl
```

4- Move the binaries to /usr/local/bin.

```
$ sudo mv \
kube-apiserver \
kube-controller-manager \
kube-scheduler \
kubectl \
/usr/local/bin/
```

5- Create the API configuration directory.

```
$ sudo mkdir -p /var/lib/kubernetes
```

6- Copy the certificates to the API configuration directory.

```
$ sudo cp \
ca.pem \
ca-key.pem \
kubernetes-key.pem \
kubernetes.pem \
encryption-config.yaml \
/var/lib/kubernetes
```

7- Create an API systemd unit file

```
$ sudo vim /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-apiserver \
  --admission-control=Initializers,NamespaceLifecycle,NodeRestriction,
  --advertise-address=10.10.40.60 \
  --allow-privileged=true \
  --apiserver-count=3 \
  --audit-log-maxage=30 \
  --audit-log-maxbackup=3 \
  --audit-log-maxsize=100 \
  --audit-log-path=/var/log/audit.log \
  --authorization-mode=Node,RBAC \
  --bind-address=0.0.0.0 \
  --client-ca-file=/var/lib/kubernetes/ca.pem \
  --enable-swagger-ui=true \
  --etcd-cafile=/var/lib/kubernetes/ca.pem \
  --etcd-certfile=/var/lib/kubernetes/kubernetes.pem \
  --etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \
  --etcd-servers=https://10.10.40.60:2379,https://10.10.40.61:2379,htt
  --event-ttl=1h \
```

```
  --experimental-encryption-provider-config=/var/lib/kubernetes/encryp
  --insecure-bind-address=127.0.0.1 \
  --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \
  --kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \
  --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \
  --kubelet-https=true \
  --runtime-config=api/all \
  --service-account-key-file=/var/lib/kubernetes/ca-key.pem \
  --service-cluster-ip-range=10.32.0.0/24 \
  --service-node-port-range=30000-32767 \
  --tls-ca-file=/var/lib/kubernetes/ca.pem \
  --tls-cert-file=/var/lib/kubernetes/kubernetes.pem \
  --tls-private-key-file=/var/lib/kubernetes/kubernetes-key.pem \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

8- Create a controller manager systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-controller-manager \
  --address=0.0.0.0 \
  --cluster-cidr=10.20.0.0/16 \
  --cluster-name=kubernetes \
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.pem \
  --cluster-signing-key-file=/var/lib/kubernetes/ca-key.pem \
  --leader-elect=true \
  --master=http://127.0.0.1:8080 (http://127.0.0.1:8080) \
  --root-ca-file=/var/lib/kubernetes/ca.pem \
  --service-account-private-key-file=/var/lib/kubernetes/ca-key.pem \
  --service-cluster-ip-range=10.32.0.0/24 \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

9- Create a scheduler systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-scheduler \
  --leader-elect=true \
  --master=http://127.0.0.1:8080 (http://127.0.0.1:8080) \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

10- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

11- Enable the services to start at boot time.

```
$ sudo systemctl enable \
kube-apiserver \
kube-controller-manager \
kube-scheduler
```

12- Start the services.

```
$ sudo systemctl start kube-apiserver kube-controller-manager kube-scheduler
```

13- Verify the different components.

```
$ kubectl get componentstatuses
```

Installation of the 10.10.40.61 master node

1- SSH to 10.10.40.61.

2- Download the binaries.

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```


```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```


3- Add the execution permissions to the binaries.

```
$ sudo chmod +x \
kube-apiserver \
kube-controller-manager \
kube-scheduler \
kubectl
```

4- Move the binaries to /usr/local/bin.

```
$ sudo mv \
kube-apiserver \
kube-controller-manager \
kube-scheduler \
kubectl \
/usr/local/bin/
```

5- Create the API configuration directory.

```
$ sudo mv kubernetes.default.svc.cluster.local.pem \
```

6- Copy the certificates to the API configuration directory.

```
$ sudo cp \
ca.pem \
ca-key.pem \
kubernetes-key.pem \
kubernetes.pem \
encryption-config.yaml \
/var/lib/kubernetes
```

7- Create an API systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-apiserver \
  --admission-control=Initializers,NamespaceLifecycle,NodeRestriction,
  --advertise-address=10.10.40.61 \
  --allow-privileged=true \
  --apiserver-count=3 \
  --audit-log-maxage=30 \
  --audit-log-maxbackup=3 \
  --audit-log-maxsize=100 \
  --audit-log-path=/var/log/audit.log \
  --authorization-mode=Node,RBAC \
  --bind-address=0.0.0.0 \
  --client-ca-file=/var/lib/kubernetes/ca.pem \
  --enable-swagger-ui=true \
  --etcd-cafile=/var/lib/kubernetes/ca.pem \
  --etcd-certfile=/var/lib/kubernetes/kubernetes.pem \
  --etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \
  --etcd-servers=https://10.10.40.60:2379,https://10.10.40.61:2379,htt
  --event-ttl=1h \
```

```
  --experimental-encryption-provider-config=/var/lib/kubernetes/encryp
  --insecure-bind-address=127.0.0.1 \
  --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \
  --kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \
  --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \
  --kubelet-https=true \
  --runtime-config=api/all \
  --service-account-key-file=/var/lib/kubernetes/ca-key.pem \
  --service-cluster-ip-range=10.32.0.0/24 \
  --service-node-port-range=30000-32767 \
  --tls-ca-file=/var/lib/kubernetes/ca.pem \
  --tls-cert-file=/var/lib/kubernetes/kubernetes.pem \
  --tls-private-key-file=/var/lib/kubernetes/kubernetes-key.pem \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

8- Create a controller manager systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-controller-manager \
  --address=0.0.0.0 \
  --cluster-cidr=10.20.0.0/16 \
  --cluster-name=kubernetes \
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.pem \
  --cluster-signing-key-file=/var/lib/kubernetes/ca-key.pem \
  --leader-elect=true \
  --master=http://127.0.0.1:8080 (http://127.0.0.1:8080) \
  --root-ca-file=/var/lib/kubernetes/ca.pem \
  --service-account-private-key-file=/var/lib/kubernetes/ca-key.pem \
  --service-cluster-ip-range=10.32.0.0/24 \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

9- Create a scheduler systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-scheduler \
  --leader-elect=true \
  --master=http://127.0.0.1:8080 (http://127.0.0.1:8080) \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

10- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

11- Enable the services to start at boot time.

```
$ sudo systemctl enable \
kube-apiserver \
kube-controller-manager \
kube-scheduler
```

12- Start the services.

```
$ sudo systemctl start kube-apiserver kube-controller-manager kube-scheduler
```

13- Verify the different components.

```
$ kubectl get componentstatuses
```

Installation of the 10.10.40.62 master node

1- SSH to 10.10.40.62.

2- Download the binaries.

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```


```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```


3- Add the execution permissions to the binaries.

```
$ sudo chmod +x \
kube-apiserver \
kube-controller-manager \
kube-scheduler \
kubectl
```

4- Move the binaries to /usr/local/bin.

```
$ sudo mv \
kube-apiserver \
kube-controller-manager \
kube-scheduler \
kubectl \
/usr/local/bin/
```

5- Create the API configuration directory.

```
$ sudo mv kube-scheduler.kubeconfig /var/lib/kubernetes/
```

6- Copy the certificates to the API configuration directory.

```
$ sudo cp \
ca.pem \
ca-key.pem \
kubernetes-key.pem \
kubernetes.pem \
encryption-config.yaml \
/var/lib/kubernetes
```

7- Create an API systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-apiserver \
  --admission-control=Initializers,NamespaceLifecycle,NodeRestriction,
  --advertise-address=10.10.40.62 \
  --allow-privileged=true \
  --apiserver-count=3 \
  --audit-log-maxage=30 \
  --audit-log-maxbackup=3 \
  --audit-log-maxsize=100 \
  --audit-log-path=/var/log/audit.log \
  --authorization-mode=Node,RBAC \
  --bind-address=0.0.0.0 \
  --client-ca-file=/var/lib/kubernetes/ca.pem \
  --enable-swagger-ui=true \
  --etcd-cafile=/var/lib/kubernetes/ca.pem \
  --etcd-certfile=/var/lib/kubernetes/kubernetes.pem \
  --etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \
  --etcd-servers=https://10.10.40.60:2379,https://10.10.40.61:2379,htt
  --event-ttl=1h \
```

```
  --experimental-encryption-provider-config=/var/lib/kubernetes/encryp
  --insecure-bind-address=127.0.0.1 \
  --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \
  --kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \
  --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \
  --kubelet-https=true \
  --runtime-config=api/all \
  --service-account-key-file=/var/lib/kubernetes/ca-key.pem \
  --service-cluster-ip-range=10.32.0.0/24 \
  --service-node-port-range=30000-32767 \
  --tls-ca-file=/var/lib/kubernetes/ca.pem \
  --tls-cert-file=/var/lib/kubernetes/kubernetes.pem \
  --tls-private-key-file=/var/lib/kubernetes/kubernetes-key.pem \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

8- Create a controller manager systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-controller-manager \
  --address=0.0.0.0 \
  --cluster-cidr=10.20.0.0/16 \
  --cluster-name=kubernetes \
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.pem \
  --cluster-signing-key-file=/var/lib/kubernetes/ca-key.pem \
  --leader-elect=true \
  --master=http://127.0.0.1:8080 (http://127.0.0.1:8080) \
  --root-ca-file=/var/lib/kubernetes/ca.pem \
  --service-account-private-key-file=/var/lib/kubernetes/ca-key.pem \
  --service-cluster-ip-range=10.32.0.0/24 \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

9- Create a scheduler systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-scheduler \
  --leader-elect=true \
  --master=http://127.0.0.1:8080 (http://127.0.0.1:8080) \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

10- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

11- Enable the services to start at boot time.

```
$ sudo systemctl enable \
kube-apiserver \
kube-controller-manager \
kube-scheduler
```

12- Start the services.

```
$ sudo systemctl start kube-apiserver kube-controller-manager kube-scheduler
```

13- Verify the different components.

```
$ kubectl get componentstatuses
```

Configure the API to access the kubelet

We need to configure each API server to be able to access the kubelet running on each worker node. To achieve this, we need to create a cluster role.

1- SSH to the 10.10.40.60 master node.

2- Create a Kubernetes manifest file kube-apiserver-to-kubelet.yaml.

```
$ vim kube-apiserver-to-kubelet.yaml
apiVersion: rbac.authorization.k8s.io/v1beta1 (http://rbac.authorization.k8s.io
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate (http://rbac.authorization.kube
  labels:
    kubernetes.io/bootstrapping (http://kubernetes.io/bootstrapping): rbac-defa
  name: system:kube-apiserver-to-kubelet
rules:
  - apiGroups:
      - ""
    resources:
      - nodes/proxy
      - nodes/stats
      - nodes/log
      - nodes/spec
      - nodes/metrics
    verbs:
      - "*"
```

3- Apply the configuration.

```
$ kubectl create -f kube-apiserver-to-kubelet.yaml
```

1- Create a manifest file to bind the cluster role to the kubernetes user used by the API server.

```
$ vim kube-apiserver-to-kubelet-bind.yaml
apiVersion: rbac.authorization.k8s.io/v1beta1 (http://rbac.authorization.k8s.io
kind: ClusterRoleBinding
metadata:
  name: system:kube-apiserver
  namespace: ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:kube-apiserver-to-kubelet
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: kubernetes
```

5- Apply the configuration.

```
$ kubectl create -f kube-apiserver-to-kubelet-bind.yaml
```

Verify that you can access the API server through the HAProxy load balancer

At this point, you should be able to access the API server through the HAProxy load balancer.

```
$ curl --cacert /var/lib/kubernetes/ca.pem https://10.10.40.68:6443/version
```

Install the Kubernetes worker nodes components

The Kubernetes worker nodes need to run various components. We will install the cni plugin, cri containerd, the kube-proxy, the kubelet, and the kubectl client on each nodes.

Installation of the 10.10.40.70 worker node

1- SSH to 10.10.40.70.

2- Disable the swap as the kubelet refuses to start with the swap enabled.

```
$ sudo swapoff -a
```

```
$ sudo sed -i '/ swap / s/^/#/' /etc/fstab
```

3- Modify the hostname to match the IP.

```
$ sudo hostnamectl set-hostname 10.10.40.70
```

4- Update the virtual machine.

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

5- Install socat. Socat is a dependency of the kubectl port-forward command. The port forwarding will not work if you don't install socat.

```
$ sudo apt-get install socat
```

6- Download the binaries.

```
$ wget https://github.com/containernetworking (http://github.com/containernetwo
```

```
$ wget https://github.com/kubernetes (http://github.com/kubernetes)-incubator/c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

7- Create the installation directories.

```
$ sudo mkdir -p \
/etc/cni/net.d \
/opt/cni/bin \
/var/lib/kubelet \
/var/lib/kube-proxy \
/var/lib/kubernetes \
/var/run/kubernetes
```

8- Install the binaries in their respective directory.

```
$ sudo tar -xvzf cni-plugins-amd64-v0.6.0.tgz -C /opt/cni/bin/
```

```
$ sudo tar -xvzf cri-containerd-1.0.0-beta.0.linux-amd64.tar.gz -C /
```

```
$ chmod +x kubectl kube-proxy kubelet
```

```
$ sudo mv kubectl kube-proxy kubelet /usr/local/bin/
```

9- Configure the CNI networking.

```
$ sudo vim /etc/cni/net.d/10-bridge.conf
{
    "cniVersion": "0.3.1",
    "name": "bridge",
    "type": "bridge",
    "bridge": "cnio0",
    "isGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "ranges": [
          [{"subnet": "10.20.0.0/24"}]
        ],
        "routes": [{"dst": "0.0.0.0/0"}]
    }
}
```

```
$ sudo vim /etc/cni/net.d/99-loopback.conf
{
    "cniVersion": "0.3.1",
    "type": "loopback"
}
```

10- Configure the kubelet.

```
$ sudo cp 10.10.40.70-key.pem 10.10.40.70.pem /var/lib/kubelet
```

```
$ sudo cp 10.10.40.70.kubeconfig /var/lib/kubelet/kubeconfig
```

11- Create the kubelet systemd unit file.

```
$ sudo vi /etc/systemd/system/kubelet.service
```

```
$ sudo vim /etc/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube
After=cri-containerd.service
Requires=cri-containerd.service



[Service]
ExecStart=/usr/local/bin/kubelet \
  --allow-privileged=true \
  --anonymous-auth=false \
  --authorization-mode=Webhook \
  --client-ca-file=/var/lib/kubernetes/ca.pem \
  --cloud-provider= \
  --cluster-dns=10.32.0.10 \
  --cluster-domain=cluster.local \
  --container-runtime=remote \
  --container-runtime-endpoint=unix:///var/run/cri-containerd.sock \
  --image-pull-progress-deadline=2m \
  --kubeconfig=/var/lib/kubelet/kubeconfig \
  --network-plugin=cni \
  --pod-cidr=10.20.0.0/24 \
  --register-node=true \
  --runtime-request-timeout=15m \
```

```
  --tls-cert-file=/var/lib/kubelet/10.10.40.70.pem \
  --tls-private-key-file=/var/lib/kubelet/10.10.40.70-key.pem \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

12- Configure the kube-proxy.

```
$ sudo cp kube-proxy.kubeconfig /var/lib/kube-proxy/kubeconfig
```

13- Create the kube-proxy systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-proxy \
  --cluster-cidr=10.20.0.0/16 \
  --kubeconfig=/var/lib/kube-proxy/kubeconfig \
  --proxy-mode=iptables \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

14- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

15- Enable the services to start at boot time.

```
$ sudo systemctl enable containerd cri-containerd kubelet kube-proxy
```

16- Start the services.

```
$ sudo systemctl start containerd cri-containerd kubelet kube-proxy
```

Installation of the 10.10.40.71 worker node

1- SSH to 10.10.40.71.

2- Disable the swap as the kubelet refuses to start with the swap enabled.

```
$ sudo swapoff -a
```

```
$ sudo sed -i '/ swap / s/^/#/' /etc/fstab
```

3- Modify the hostname to match the IP.

```
$ sudo hostnamectl set-hostname 10.10.40.71
```

4- Update the virtual machine.

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

5- Install socat. Socat is a dependency of the kubectl port-forward command. The port forwarding will not work if you don't install socat.

```
$ sudo apt-get install socat
```

6- Download the binaries.

```
$ wget https://github.com/containernetworking (http://github.com/containernetwo
```

```
$ wget https://github.com/kubernetes (http://github.com/kubernetes)-incubator/c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

7- Create the installation directories.

```
$ sudo mkdir -p \
/etc/cni/net.d \
/opt/cni/bin \
/var/lib/kubelet \
/var/lib/kube-proxy \
/var/lib/kubernetes \
/var/run/kubernetes
```

8- Install the binaries in their respective directory.

```
$ sudo tar -xvzf cni-plugins-amd64-v0.6.0.tgz -C /opt/cni/bin/
```

```
$ sudo tar -xvzf cri-containerd-1.0.0-beta.0.linux-amd64.tar.gz -C /
```

```
$ chmod +x kubectl kube-proxy kubelet
```

```
$ sudo mv kubectl kube-proxy kubelet /usr/local/bin/
```

9- Configure the CNI networking.

```
$ sudo vim /etc/cni/net.d/10-bridge.conf
{
    "cniVersion": "0.3.1",
    "name": "bridge",
    "type": "bridge",
    "bridge": "cnio0",
    "isGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "ranges": [
          [{"subnet": "10.20.1.0/24"}]
        ],
        "routes": [{"dst": "0.0.0.0/0"}]
    }
}
```

```
$ sudo vim /etc/cni/net.d/99-loopback.conf
{
    "cniVersion": "0.3.1",
    "type": "loopback"
}
```

10- Configure the kubelet.

```
$ sudo cp 10.10.40.71-key.pem 10.10.40.71.pem /var/lib/kubelet
```

```
$ sudo cp 10.10.40.71.kubeconfig /var/lib/kubelet/kubeconfig
```

11- Create the kubelet systemd unit file.

```
$ sudo vi /etc/systemd/system/kubelet.service
```

```
$ sudo vim /etc/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube
After=cri-containerd.service
Requires=cri-containerd.service



[Service]
ExecStart=/usr/local/bin/kubelet \
  --allow-privileged=true \
  --anonymous-auth=false \
  --authorization-mode=Webhook \
  --client-ca-file=/var/lib/kubernetes/ca.pem \
  --cloud-provider= \
  --cluster-dns=10.32.0.10 \
  --cluster-domain=cluster.local \
  --container-runtime=remote \
  --container-runtime-endpoint=unix:///var/run/cri-containerd.sock \
  --image-pull-progress-deadline=2m \
  --kubeconfig=/var/lib/kubelet/kubeconfig \
  --network-plugin=cni \
  --pod-cidr=10.20.1.0/24 \
  --register-node=true \
  --runtime-request-timeout=15m \
```

```
  --tls-cert-file=/var/lib/kubelet/10.10.40.71.pem \
  --tls-private-key-file=/var/lib/kubelet/10.10.40.71-key.pem \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

12- Configure the kube-proxy.

```
$ sudo cp kube-proxy.kubeconfig /var/lib/kube-proxy/kubeconfig
```

13- Create the kube-proxy systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-proxy \
  --cluster-cidr=10.20.0.0/16 \
  --kubeconfig=/var/lib/kube-proxy/kubeconfig \
  --proxy-mode=iptables \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

14- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

15- Enable the services to start at boot time.

```
$ sudo systemctl start containerd cri-containerd kubelet kube-proxy
```

16- Start the services.

```
$ sudo systemctl start containerd cri-containerd kubelet kube-proxy
```

Installation of the 10.10.40.72 worker node

1- SSH to 10.10.40.72.

2- Disable the swap as the kubelet refuses to start with the swap enabled.

```
$ sudo swapoff -a
```

```
$ sudo sed -i '/ swap / s/^/#/' /etc/fstab
```

3- Modify the hostname to match the IP.

```
$ sudo hostnamectl set-hostname 10.10.40.72
```

4- Update the virtual machine.

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

5- Install socat. Socat is a dependency of the kubectl port-forward command. The port forwarding will not work if you don't install socat.

```
$ sudo apt-get install socat
```

6- Download the binaries.

```
$ wget https://github.com/containernetworking (http://github.com/containernetwo
```

```
$ wget https://github.com/kubernetes (http://github.com/kubernetes)-incubator/c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

```
$ wget https://storage.googleapis.com/kubernetes- (https://storage.googleapis.c
```

7- Create the installation directories.

```
$ sudo mkdir -p \
/etc/cni/net.d \
/opt/cni/bin \
/var/lib/kubelet \
/var/lib/kube-proxy \
/var/lib/kubernetes \
/var/run/kubernetes
```

8- Install the binaries in their respective directory.

```
$ sudo tar -xvzf cni-plugins-amd64-v0.6.0.tgz -C /opt/cni/bin/
```

```
$ sudo tar -xvzf cri-containerd-1.0.0-beta.0.linux-amd64.tar.gz -C /
```

```
$ chmod +x kubectl kube-proxy kubelet
```

```
$ sudo mv kubectl kube-proxy kubelet /usr/local/bin/
```

9- Configure the CNI networking.

```
$ sudo vim /etc/cni/net.d/10-bridge.conf
{
    "cniVersion": "0.3.1",
    "name": "bridge",
    "type": "bridge",
    "bridge": "cnio0",
    "isGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "ranges": [
          [{"subnet": "10.20.2.0/24"}]
        ],
        "routes": [{"dst": "0.0.0.0/0"}]
    }
}
```

```
$ sudo vim /etc/cni/net.d/99-loopback.conf
{
    "cniVersion": "0.3.1",
    "type": "loopback"
}
```

10- Configure the kubelet.

```
$ sudo cp 10.10.40.72-key.pem 10.10.40.72.pem /var/lib/kubelet
```

```
$ sudo cp 10.10.40.72.kubeconfig /var/lib/kubelet/kubeconfig
```

(http://blog.inkubate.io)

11- Create the kubelet systemd unit file.

```
$ sudo vim /etc/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube
After=cri-containerd.service
Requires=cri-containerd.service



[Service]
ExecStart=/usr/local/bin/kubelet \
  --allow-privileged=true \
  --anonymous-auth=false \
  --authorization-mode=Webhook \
  --client-ca-file=/var/lib/kubernetes/ca.pem \
  --cloud-provider= \
  --cluster-dns=10.32.0.10 \
  --cluster-domain=cluster.local \
  --container-runtime=remote \
  --container-runtime-endpoint=unix:///var/run/cri-containerd.sock \
  --image-pull-progress-deadline=2m \
  --kubeconfig=/var/lib/kubelet/kubeconfig \
  --network-plugin=cni \
  --pod-cidr=10.20.2.0/24 \
  --register-node=true \
  --runtime-request-timeout=15m \
```

```
  --tls-cert-file=/var/lib/kubelet/10.10.40.72.pem \
  --tls-private-key-file=/var/lib/kubelet/10.10.40.72-key.pem \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

12- Configure the kube-proxy.

```
$ sudo cp kube-proxy.kubeconfig /var/lib/kube-proxy/kubeconfig
```

13- Create the kube-proxy systemd unit file.

```
$ sudo vim /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes (https://github.com/kube



[Service]
ExecStart=/usr/local/bin/kube-proxy \
  --cluster-cidr=10.20.0.0/16 \
  --kubeconfig=/var/lib/kube-proxy/kubeconfig \
  --proxy-mode=iptables \
  --v=2
Restart=on-failure
RestartSec=5



[Install]
WantedBy=multi-user.target
```

14- Reload the daemon configuration.

```
$ sudo systemctl daemon-reload
```

15- Enable the services to start at boot time.

```
$ sudo systemctl start containerd cri-containerd kubelet kube-proxy
```

16- Start the services.

```
$ sudo systemctl start containerd cri-containerd kubelet kube-proxy
```

Verify that the Kubernetes cluster is up and running

1- SSH to 10.10.40.60.

2- List the worker nodes.

```
$ kubectl get nodes
```

The three worker nodes should be listed and in a ready state.

Configure your client machine to access the Kubernetes cluster

We need to create a kubeconfig file on the client machine to be able to manage the Kubernetes cluster with kubectl from this machine.

1- Add the cluster information.

```
$ kubectl config set-cluster kubernetes \
--certificate-authority=ca.pem \
--embed-certs=true \
--server=https://10.10.40.63:6443 (https://10.10.40.63:6443)
```

2- Add the credentials.

```
$ kubectl config set-credentials admin \
--client-certificate=admin.pem \
--client-key=admin-key.pem
```

3- Add the context.

```
$ kubectl config set-context kubernetes --cluster=kubernetes --user=admin
```

4- Use the context.

```
$ kubectl config use-context kubernetes
```

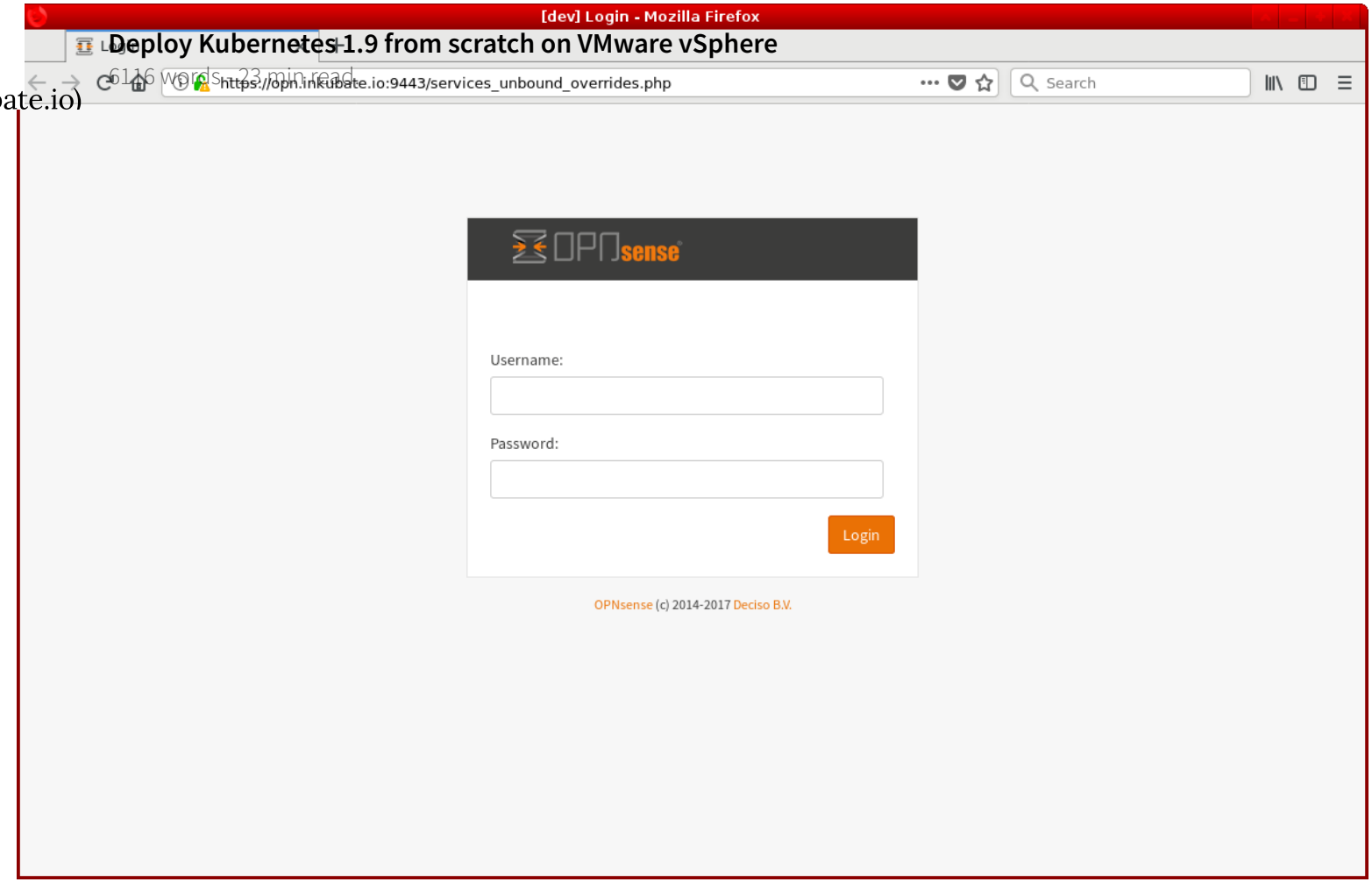
5- Verify that you can access the Kubernetes cluster from the client machine.

```
$ kubectl get nodes
```

Configuring the networking route between pods

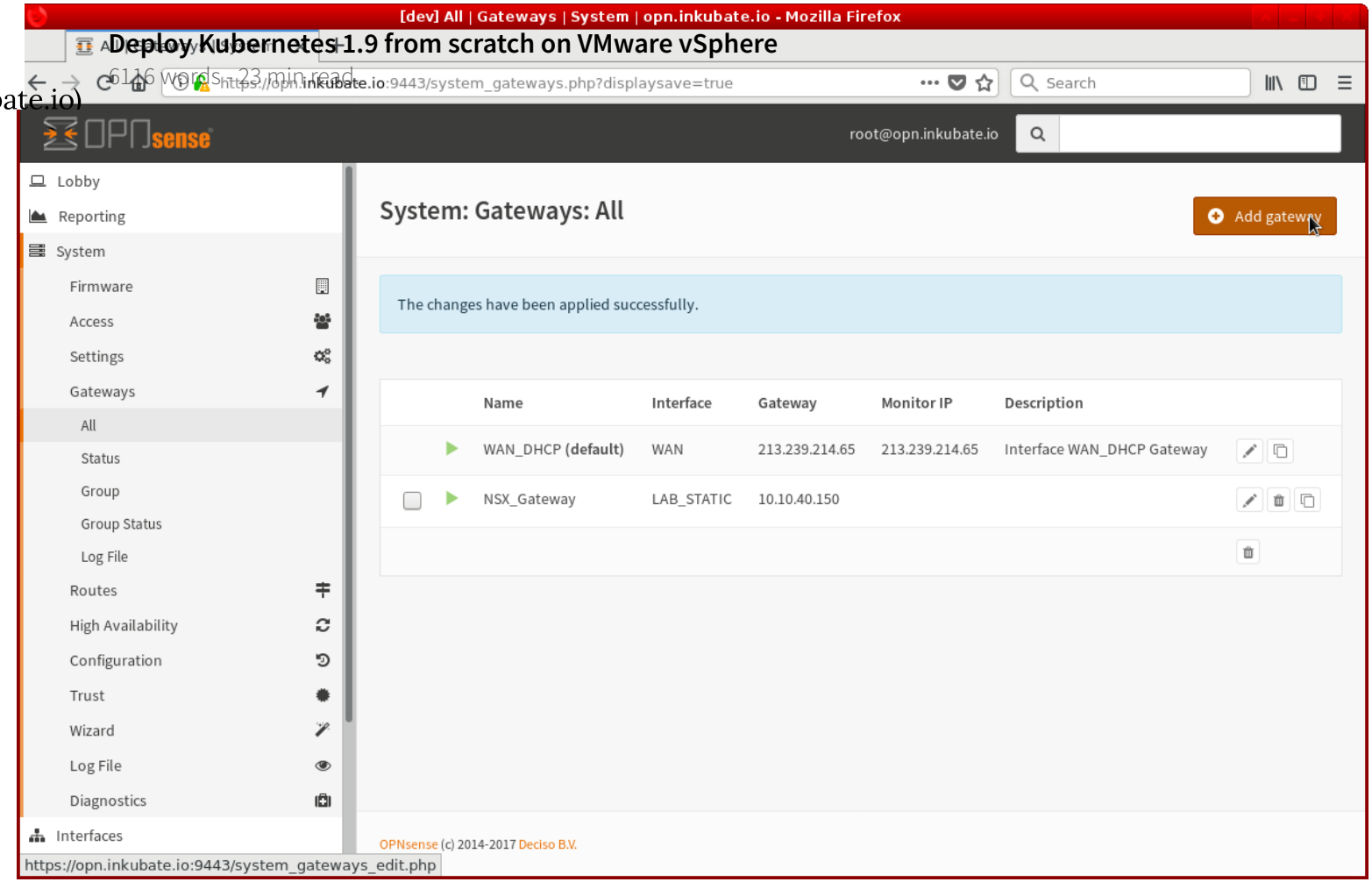
The kubernetes cluster is now up and running, but we still have one issue to solve. Each worker node uses a different network to assign an IP to the pods. It means that two pods located on two different worker nodes cannot communicate between them. To solve this issue, we need to configure static routes on the gateway of the network 10.10.40.0/24. In my case, the gateway is 10.10.40.1 and is an OPNSense server.
If you don't have access to your network gateway, you can still solve this problem by creating an overlay network for the pods with something like flannel, weave net, or calico.

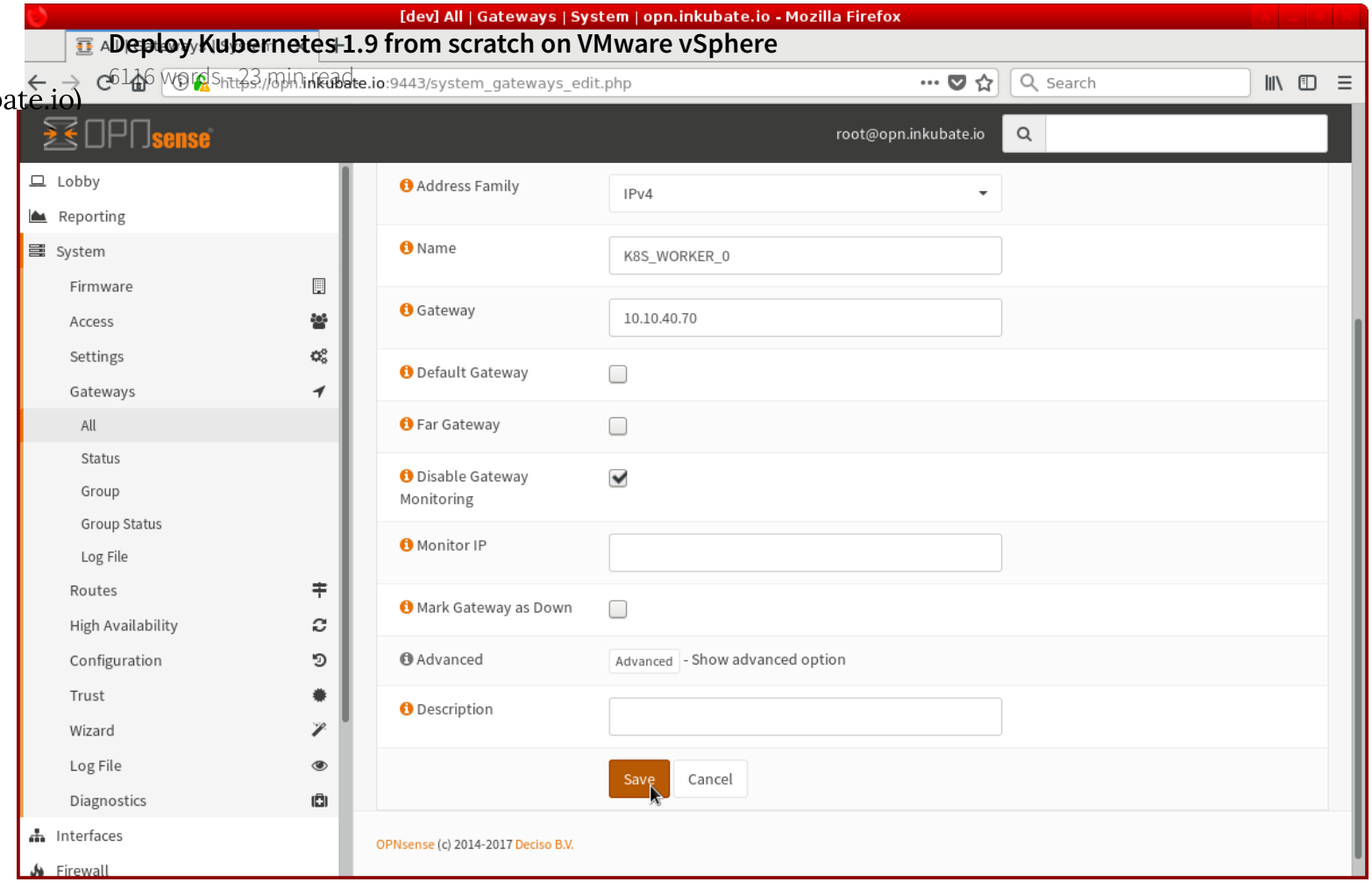
1- Login to your OPNSense gateway.



2- Add 10.10.40.70 as a new gateway.

Deploy Kubernetes 1.9 from scratch on VMware vSphere

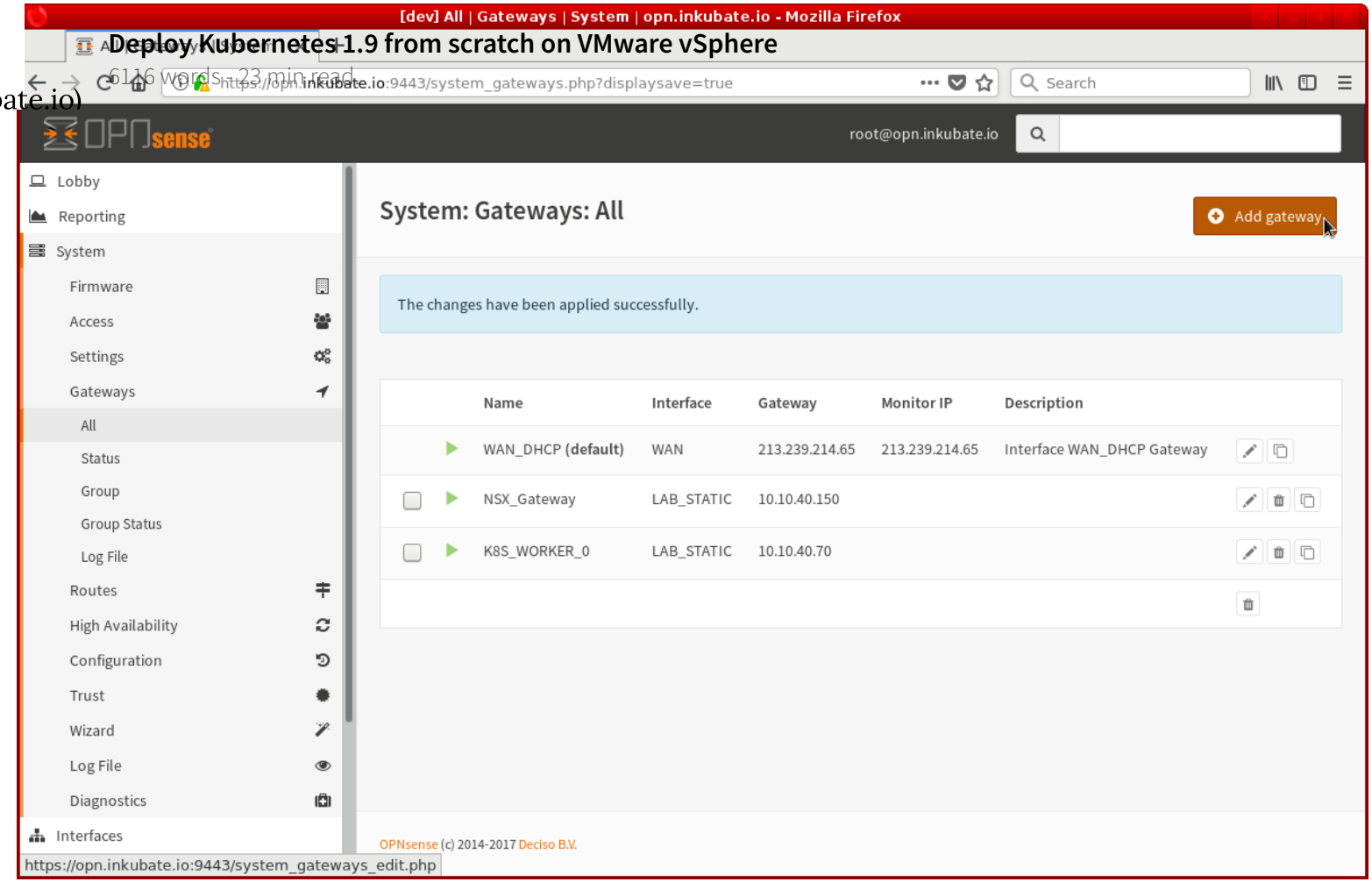


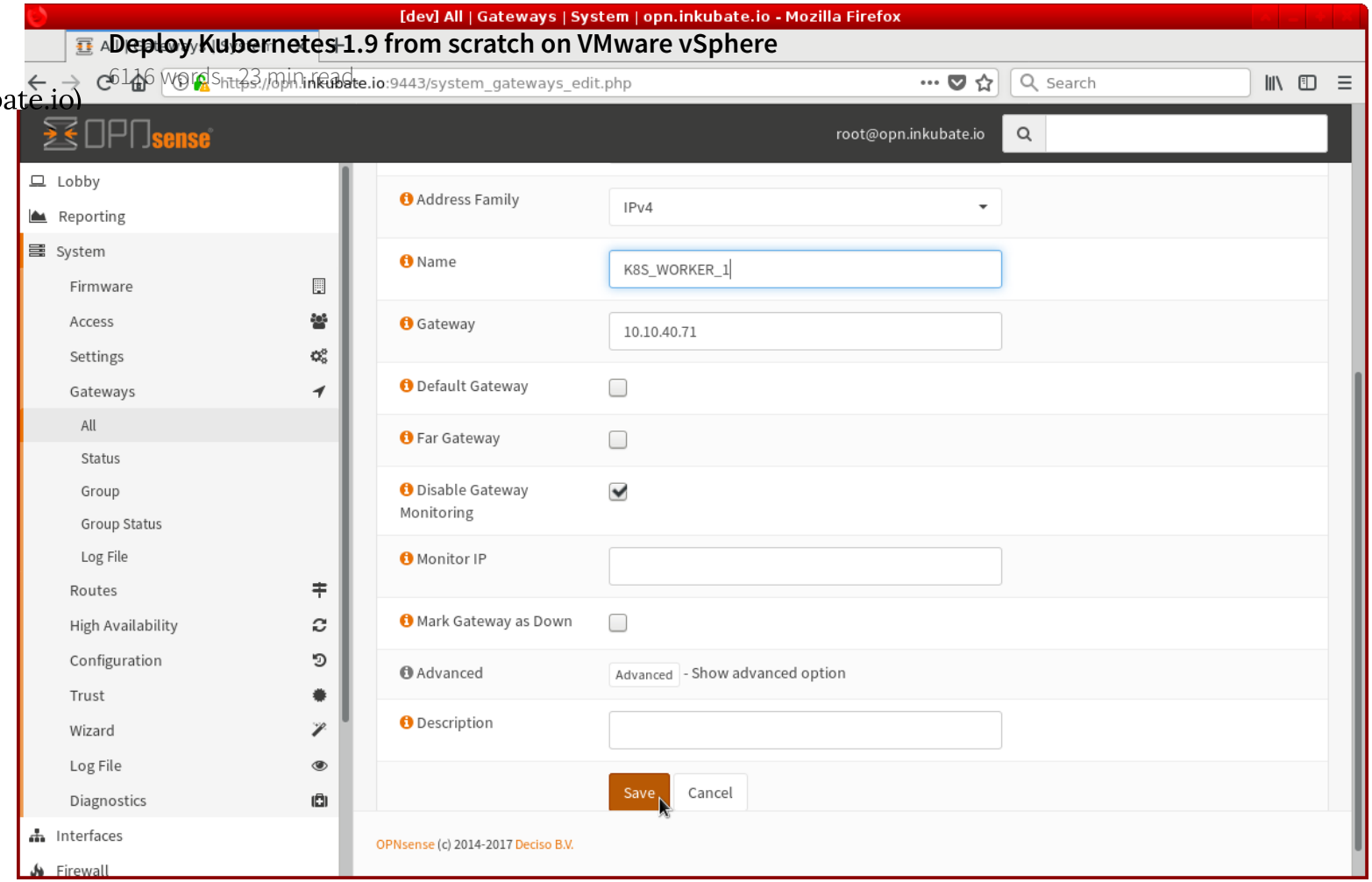


3- Add 10.10.40.71 as a new gateway.

Deploy Kubernetes 1.9 from scratch on VMware vSphere

6116 words - 23 min read

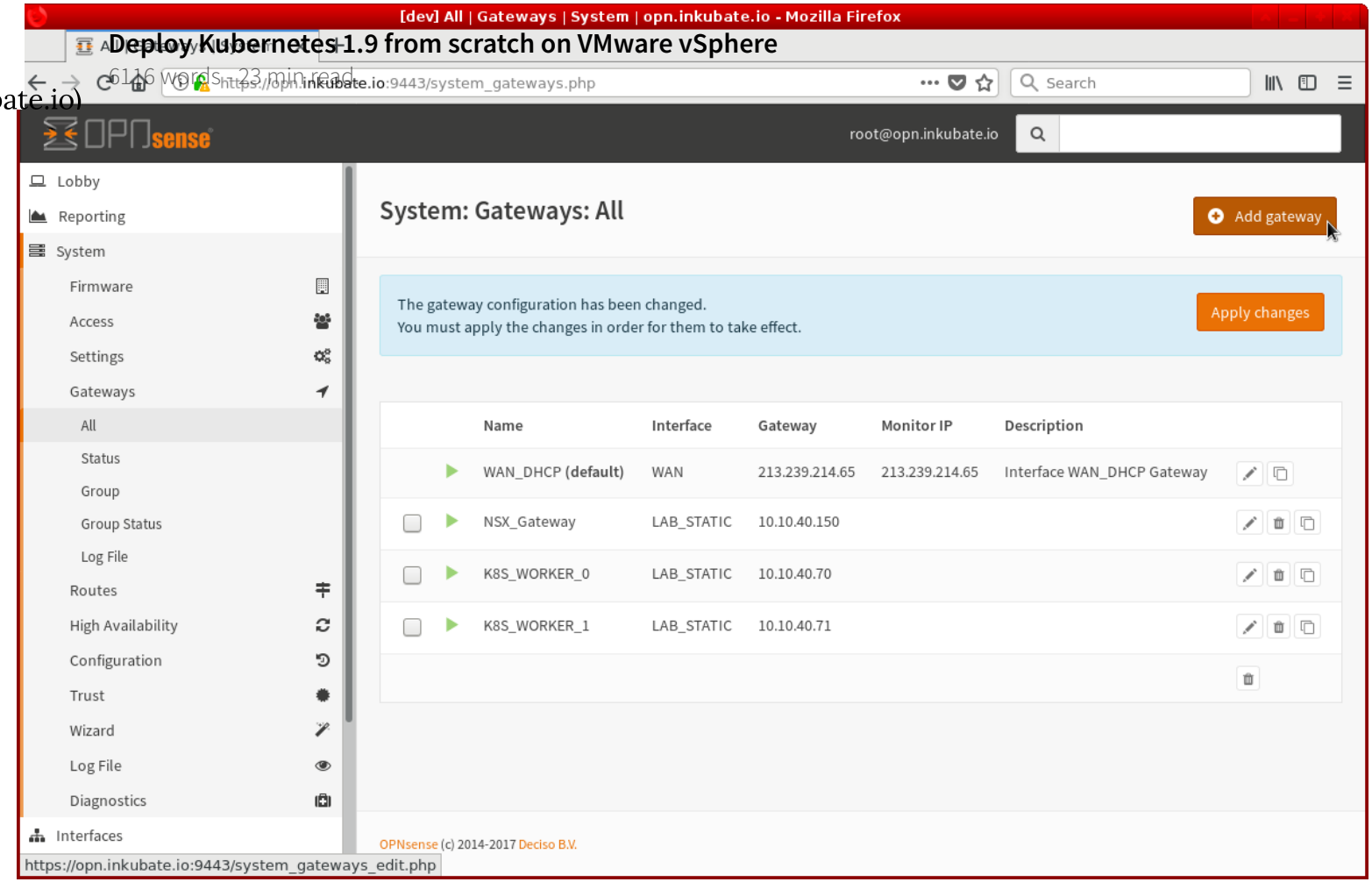


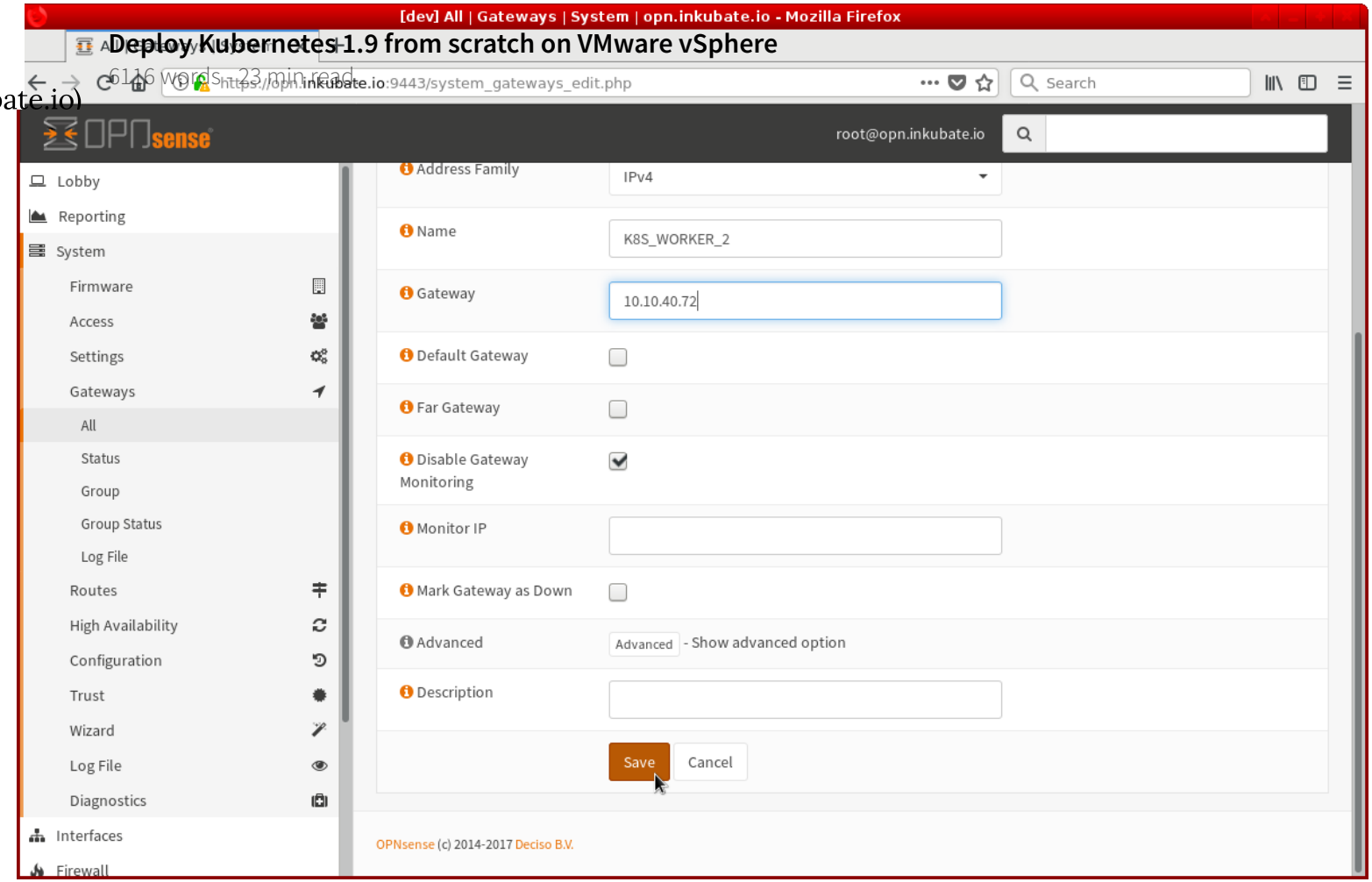


4- Add 10.10.40.72 as a new gateway.

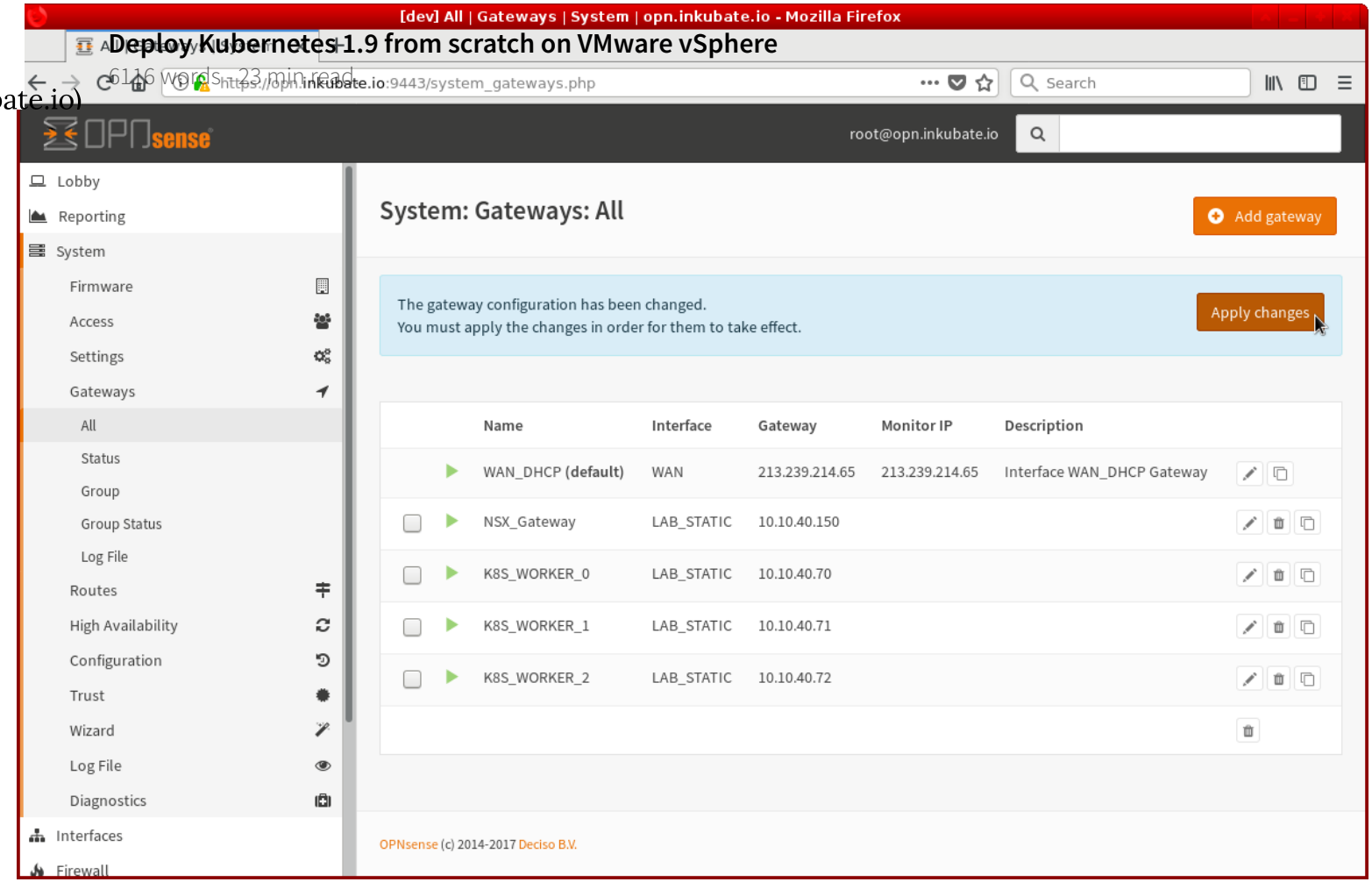
Deploy Kubernetes 1.9 from scratch on VMware vSphere

6116 words - 23 min read





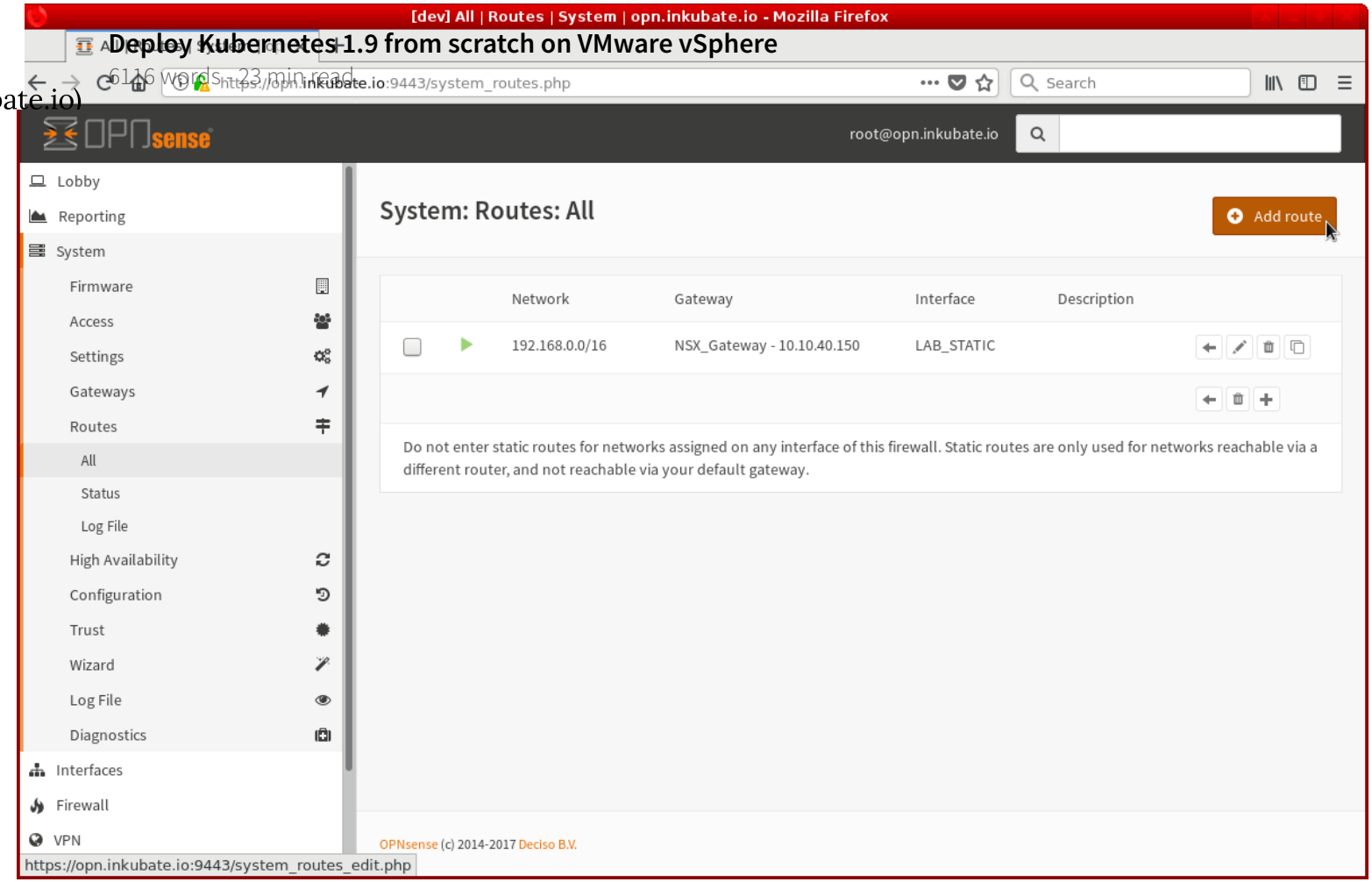
5- Apply the changes.

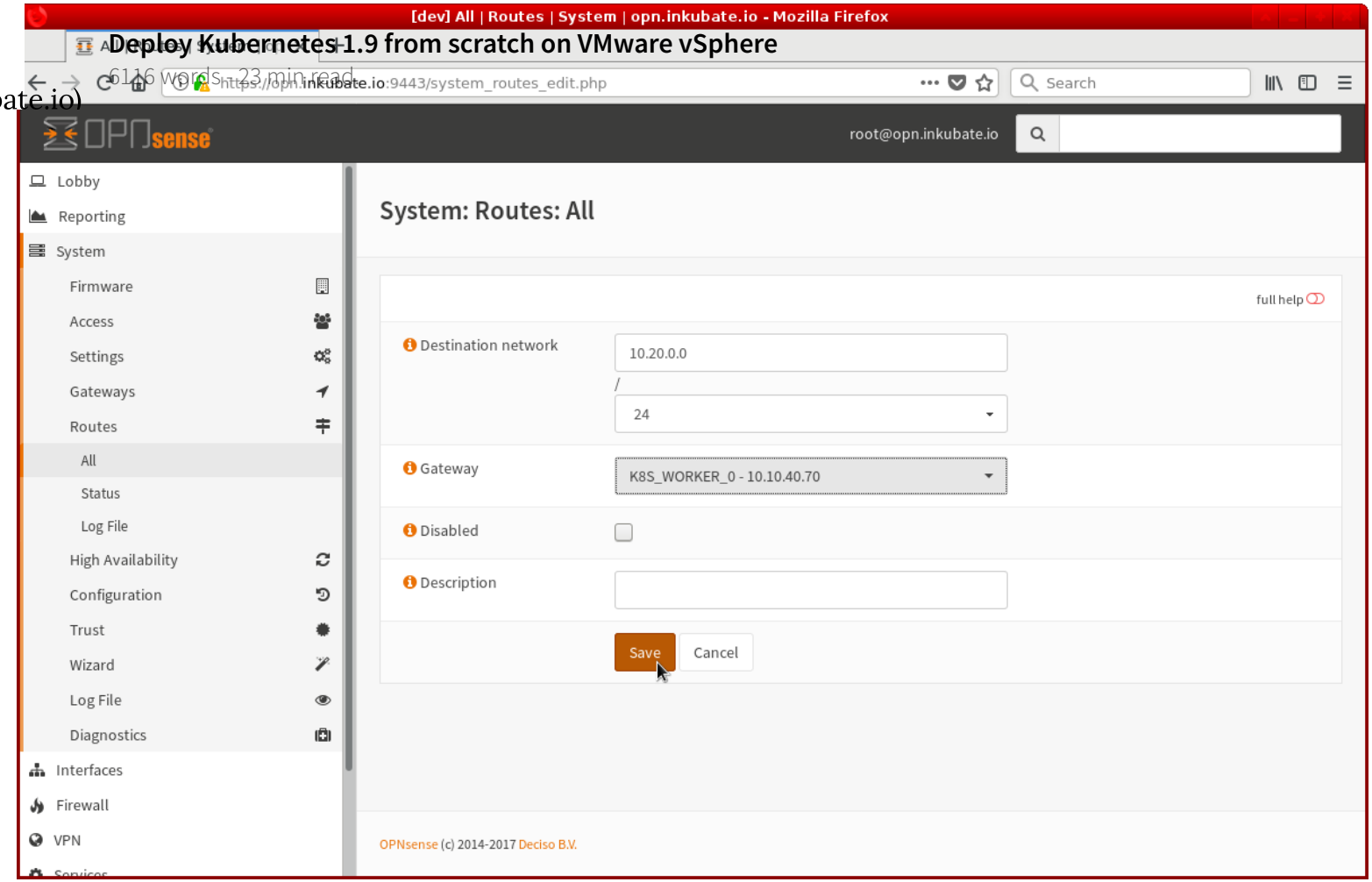


6- Create a new route for the pods running on 10.10.40.70.

Deploy Kubernetes 1.9 from scratch on VMware vSphere

6116 words - 23 min read

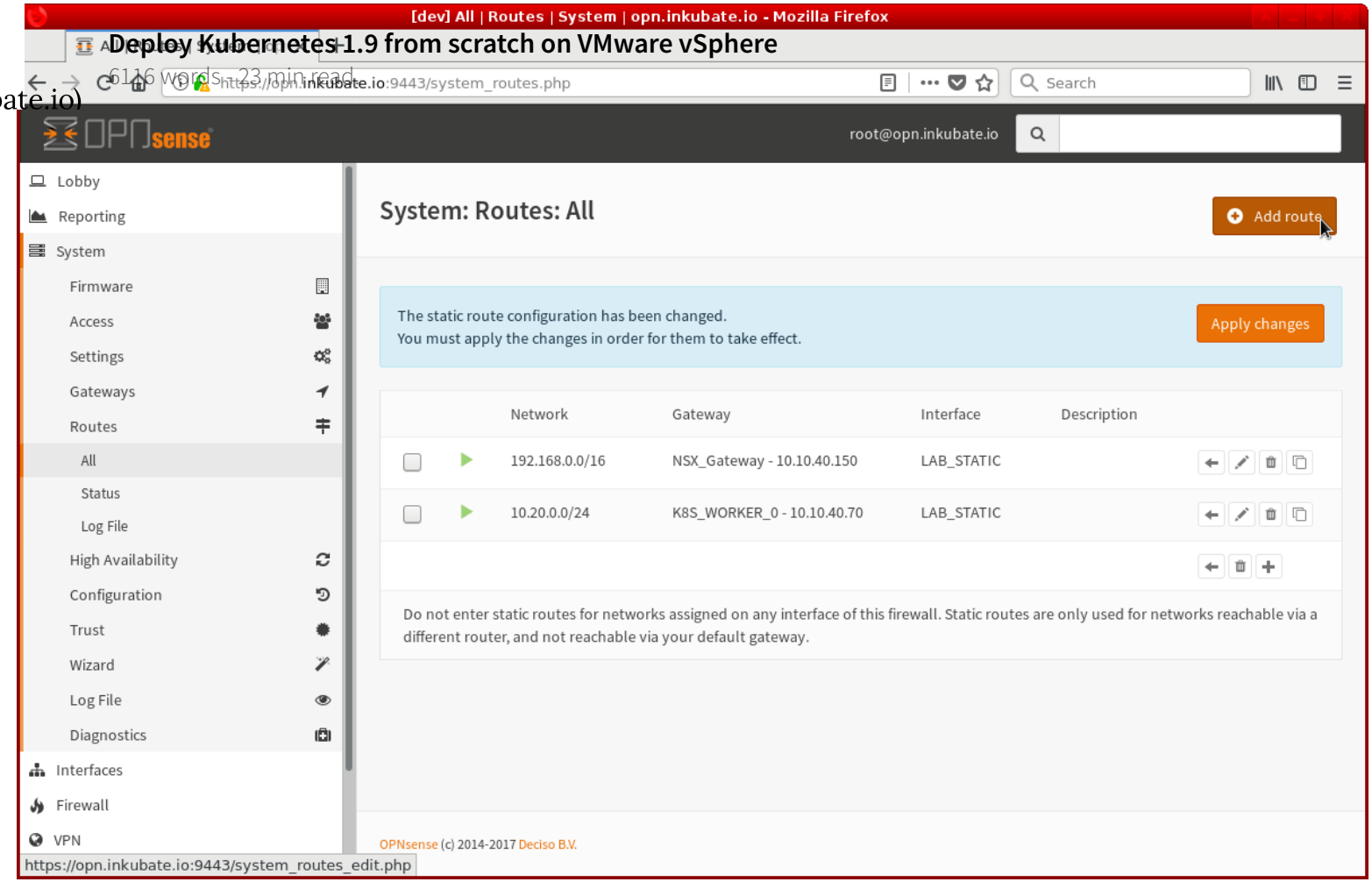


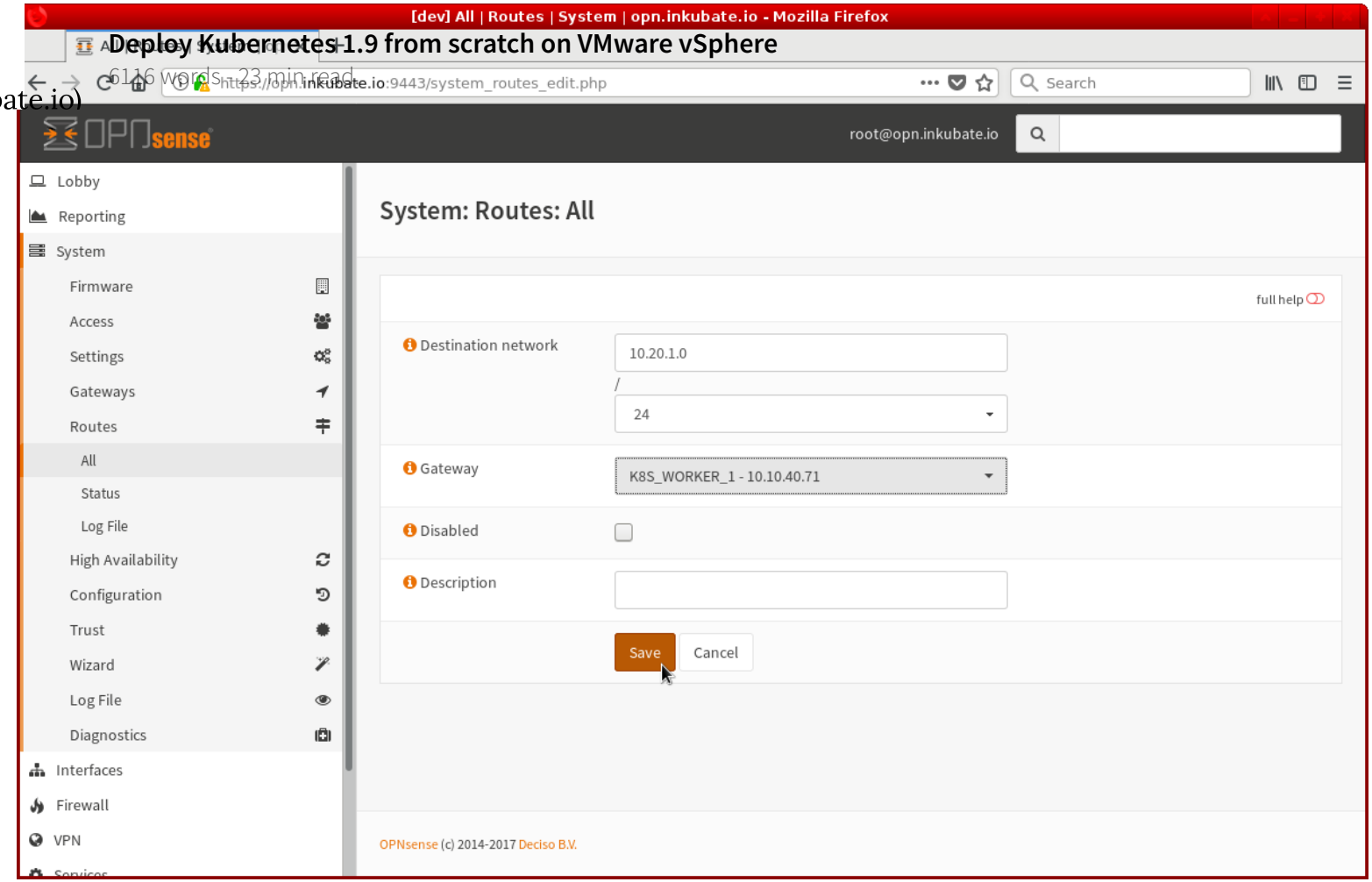


7- Create a new route for the pods running on 10.10.40.71.

Deploy Kubernetes 1.9 from scratch on VMware vSphere

6116 words - 23 min read

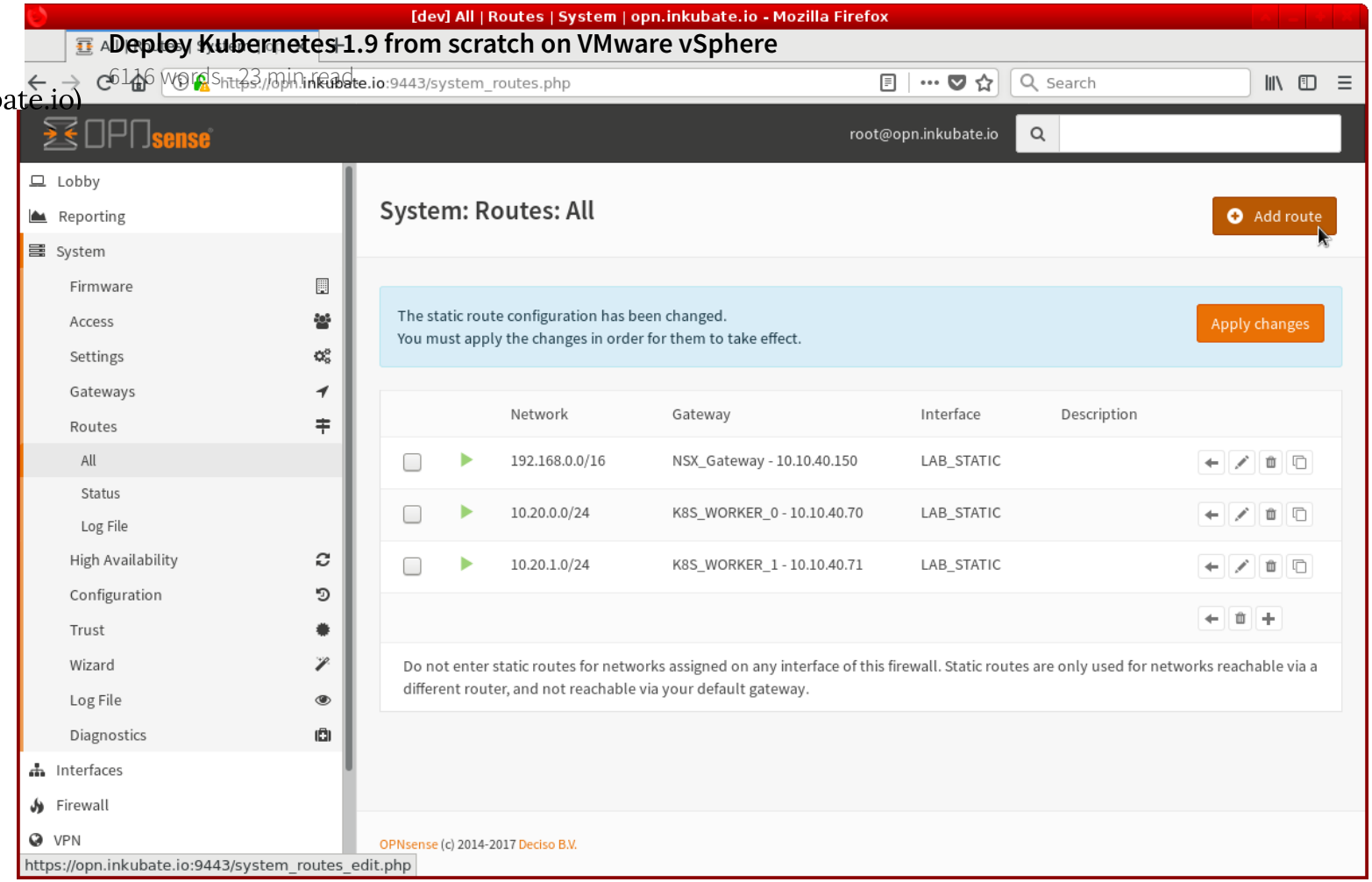


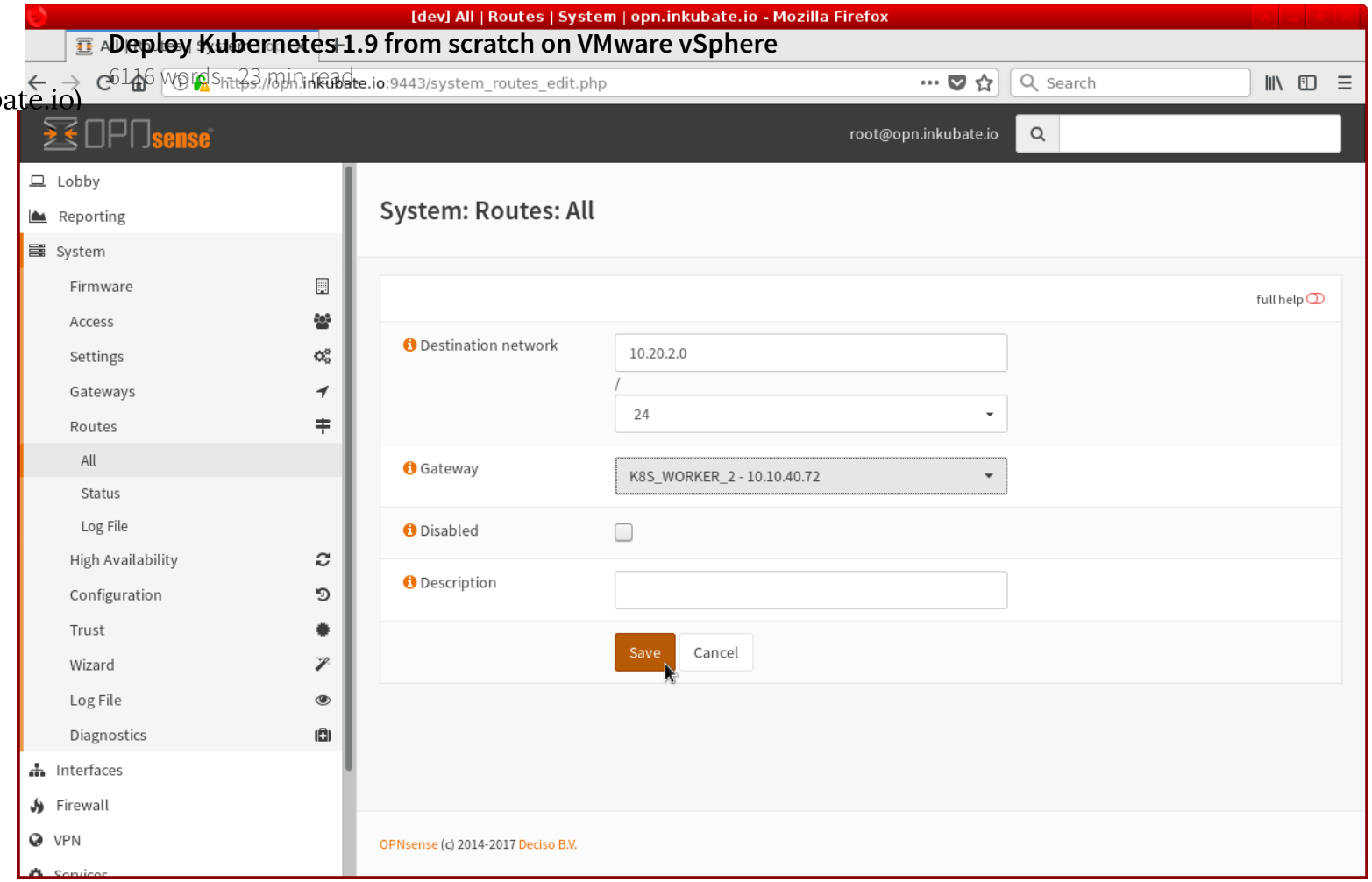


8- Create a new route for the pods running on 10.10.40.72.

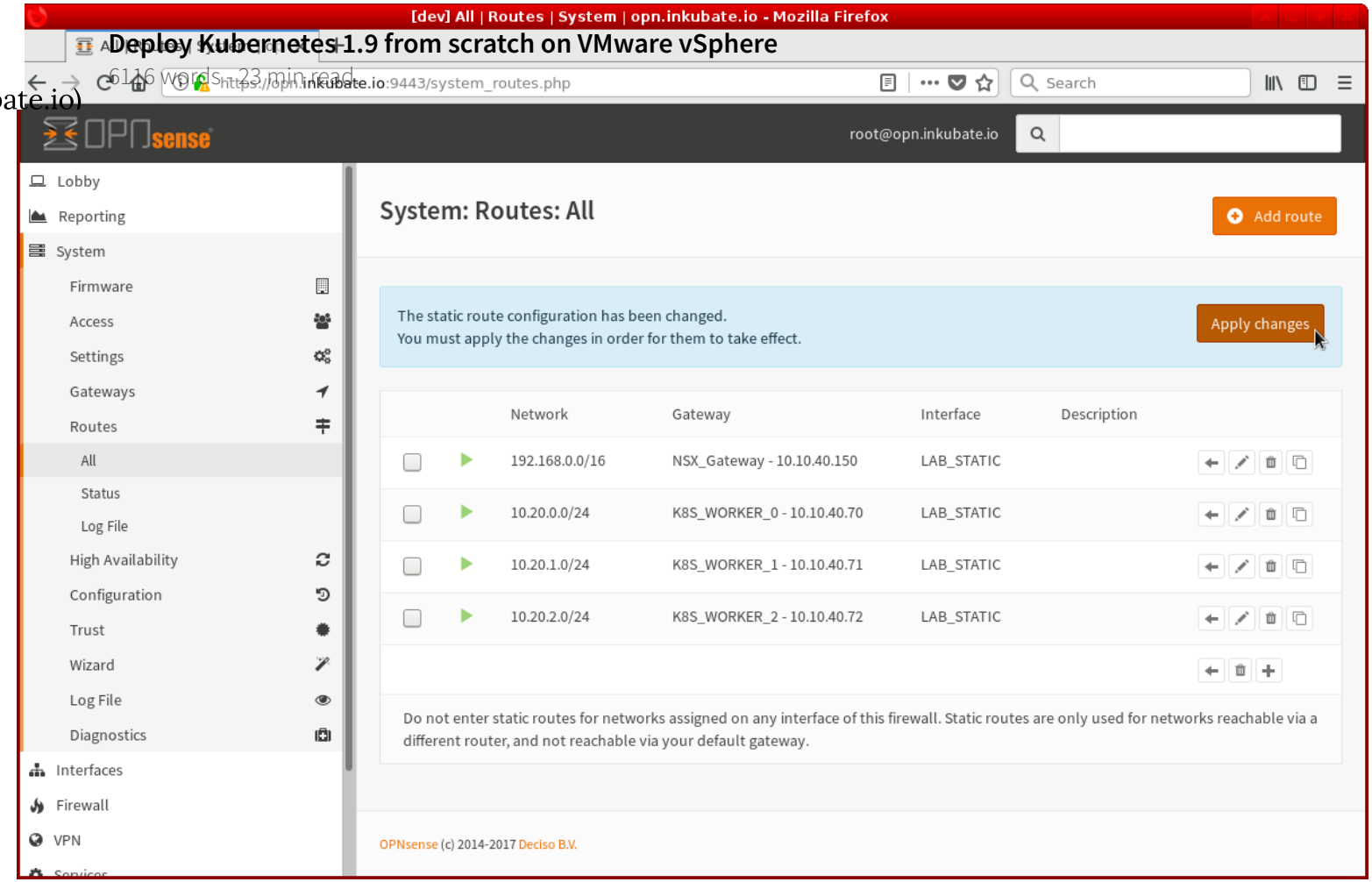
Deploy Kubernetes 1.9 from scratch on VMware vSphere

6116 words - 23 min read





9- Apply the changes.



Installing Kubernetes add-ons

We will deploy three Kubernetes add-ons on our new cluster. The kube-dns add-on which provides a DNS based service discovery for the pods running in the cluster, the dashboard add-on to have a graphical view of the cluster and the Heapster add-on to monitor our workload.

1- Create the kube-dns manifest.

```
$ vim kube-dns.yaml
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster- (http://kubernetes.io/cluster-)service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "KubeDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 10.32.0.10
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    kubernetes.io/cluster- (http://kubernetes.io/cluster-)service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
```

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster- (http://kubernetes.io/cluster-)service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  # replicas: not specified here:
  # 1. In order to make Addon Manager do not reconcile this replicas parameter.
  # 2. Default is 1.
  # 3. Will be tuned in real time if DNS horizontal auto-scaling is turned on.
  strategy:
    rollingUpdate:
      maxSurge: 10%
      maxUnavailable: 0
  selector:
    matchLabels:
      k8s-app: kube-dns
  template:
```

```
metadata:
  labels:
    k8s-app: kube-dns
  annotations:
    scheduler.alpha.kubernetes.io/critical- (http://scheduler.alpha.kuberne
spec:
  tolerations:
  - key: "CriticalAddonsOnly"
    operator: "Exists"
  volumes:
  - name: kube-dns-config
    configMap:
      name: kube-dns
      optional: true
  containers:
  - name: kubedns
    image: k8s.gcr.io/k8s-dns-kube-dns- (http://k8s.gcr.io/k8s-dns-kube-dns
    resources:
      # TODO: Set memory limits when we've profiled the container for large
      # clusters, then set request = limit to keep this container in
      # guaranteed class. Currently, this container falls into the
      # "burstable" category so the kubelet doesn't backoff from restarting
      limits:
        memory: 170Mi
      requests:
        cpu: 100m
        memory: 70Mi
    livenessProbe:
      httpGet:
        path: /healthcheck/kubedns
        port: 10054
```

```
      scheme: HTTP
      initialDelaySeconds: 60
      timeoutSeconds: 5
      successThreshold: 1
      failureThreshold: 5
    readinessProbe:
      httpGet:
        path: /readiness
        port: 8081
        scheme: HTTP
      # we poll on pod startup for the Kubernetes master service and
      # only setup the /readiness HTTP server once that's available.
      initialDelaySeconds: 3
      timeoutSeconds: 5
    args:
    - --domain=cluster.local.
    - --dns-port=10053
    - --config-dir=/kube-dns-config
    - --v=2
    env:
    - name: PROMETHEUS_PORT
      value: "10055"
    ports:
    - containerPort: 10053
      name: dns-local
      protocol: UDP
    - containerPort: 10053
      name: dns-tcp-local
      protocol: TCP
    - containerPort: 10055
      name: metrics
```

```
      protocol: TCP
    volumeMounts:
    - name: kube-dns-config
      mountPath: /kube-dns-config
  - name: dnsmasq
    image: k8s.gcr.io/k8s-dns-dnsmasq-nanny- (http://k8s.gcr.io/k8s-dns-dns
    livenessProbe:
      httpGet:
        path: /healthcheck/dnsmasq
        port: 10054
        scheme: HTTP
      initialDelaySeconds: 60
      timeoutSeconds: 5
      successThreshold: 1
      failureThreshold: 5
    args:
    - -v=2
    - -logtostderr
    - -configDir=/etc/k8s/dns/dnsmasq-nanny
    - -restartDnsmasq=true
    - --
    - -k
    - --cache-size=1000
    - --no-negcache
    - --log-facility=-
    - --server=/cluster.local/127.0.0.1#10053
    - --server=/in-addr.arpa/127.0.0.1#10053
    - --server=/ip6.arpa/127.0.0.1#10053
    ports:
    - containerPort: 53
      name: dns
```

```
      protocol: UDP
    - containerPort: 53
      name: dns-tcp
      protocol: TCP
    # see: https://github.com/kubernetes/kubernetes/issues/29055 (https://g
    resources:
      requests:
        cpu: 150m
        memory: 20Mi
    volumeMounts:
    - name: kube-dns-config
      mountPath: /etc/k8s/dns/dnsmasq-nanny
  - name: sidecar
    image: k8s.gcr.io/k8s-dns-sidecar- (http://k8s.gcr.io/k8s-dns-sidecar-)
    livenessProbe:
      httpGet:
        path: /metrics
        port: 10054
        scheme: HTTP
      initialDelaySeconds: 60
      timeoutSeconds: 5
      successThreshold: 1
      failureThreshold: 5
    args:
    - --v=2
    - --logtostderr
    - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local,
    - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local,5,S
    ports:
    - containerPort: 10054
      name: metrics
```

```
      protocol: TCP
    resources:
      requests:
        memory: 20Mi
        cpu: 10m
  dnsPolicy: Default  # Don't use cluster DNS.
  serviceAccountName: kube-dns
```

2- Deploy kube-dns.

```
$ kubectl create -f kube-dns.yaml
```

3- Verify the deployment.

```
$ kubectl get pods -l k8s-app=kube-dns -n kube-system
```

Installing the Kubernetes dashboard

1- Create the Kubernetes dashboard manifest.

```
$ vim kubernetes-dashboard.yaml
apiVersion: v1
kind: Secret
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-certs
  namespace: kube-system
type: Opaque
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1 (http://rbac.authorization.k8s.io/v1)
metadata:
  name: kubernetes-dashboard-minimal
  namespace: kube-system
rules:
  # Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create"]
  # Allow Dashboard to create 'kubernetes-dashboard-settings' config map.
- apiGroups: [""]
  resources: ["configmaps"]
```

```
  verbs: ["create"]
  # Allow Dashboard to get, update and delete Dashboard exclusive secrets.
- apiGroups: [""]
  resources: ["secrets"]
  resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-cert
  verbs: ["get", "update", "delete"]
  # Allow Dashboard to get and update 'kubernetes-dashboard-settings' config ma
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["kubernetes-dashboard-settings"]
  verbs: ["get", "update"]
  # Allow Dashboard to get metrics from heapster.
- apiGroups: [""]
  resources: ["services"]
  resourceNames: ["heapster"]
  verbs: ["proxy"]
- apiGroups: [""]
  resources: ["services/proxy"]
  resourceNames: ["heapster", "http:heapster:", "https:heapster:"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1 (http://k8s.io/v1)
kind: RoleBinding
metadata:
  name: kubernetes-dashboard-minimal
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kubernetes-dashboard-minimal
subjects:
```

```
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system
---
kind: Deployment
apiVersion: apps/v1beta2
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      containers:
      - name: kubernetes-dashboard
        image: k8s.gcr.io/kubernetes-dashboard-amd64:v1 (http://k8s.gcr.io/kube
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          - --auto-generate-certificates
          # Uncomment the following line to manually specify Kubernetes API ser
```

```
          # If not specified, Dashboard will attempt to auto discover the API s
          # to it. Uncomment only if the default does not work.
          # - --apiserver-host=http://my-address:port (http://my-address:port)
        volumeMounts:
        - name: kubernetes-dashboard-certs
          mountPath: /certs
          # Create on-disk volume to store exec logs
        - mountPath: /tmp
          name: tmp-volume
        livenessProbe:
          httpGet:
            scheme: HTTPS
            path: /
            port: 8443
          initialDelaySeconds: 30
          timeoutSeconds: 30
      volumes:
      - name: kubernetes-dashboard-certs
        secret:
          secretName: kubernetes-dashboard-certs
      - name: tmp-volume
        emptyDir: {}
      serviceAccountName: kubernetes-dashboard
      # Comment the following tolerations if Dashboard must not be deployed on
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
---
kind: Service
apiVersion: v1
metadata:
```

```
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  ports:
    - port: 443
      targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
```

2- Deploy the dashboard.

```
$ kubectl create -f kubernetes-dashboard.yaml
```

Installing Heapster

1- Create a manifest for Heapster.

```
$ vim deploy-kubernetes1
apiVersion: v1
kind: ServiceAccount
metadata:
  name: heapster
  namespace: kube-system
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heapster
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: heapster
    spec:
      serviceAccountName: heapster
      containers:
      - name: heapster
        image: k8s.gcr.io/heapster-amd64:v1.4.2 (http://k8s.gcr.io/heapster-amd
        imagePullPolicy: IfNotPresent
        command:
        - /heapster
        - --source=kubernetes:https://kubernetes.default (https://kubernetes.de
---
apiVersion: v1
kind: Service
```

```
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tre
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service (http://kubernetes.io/cluster-service): 'true
    kubernetes.io/name (http://kubernetes.io/name): Heapster
  name: heapster
  namespace: kube-system
spec:
  ports:
  - port: 80
    targetPort: 8082
  selector:
    k8s-app: heapster
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1 (http://k8s.io/v1beta1)
metadata:
  name: heapster
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:heapster
subjects:
- kind: ServiceAccount
  name: heapster
  namespace: kube-system
```

2- Deploy Heapster.

Access the Kubernetes dashboard

1- Create an admin user manifest.

```
$ vim kubernetes-dashboard-admin.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1 (http://rbac.authorization.k8s.io
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system
```

2- Create the admin user.

```
$ kubectl create -f kubernetes-dashboard-admin.yaml
```

3- Get the admin user token.

```
$ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret |
```

1- Copy the token

5- Start the proxy to access the dashboard.

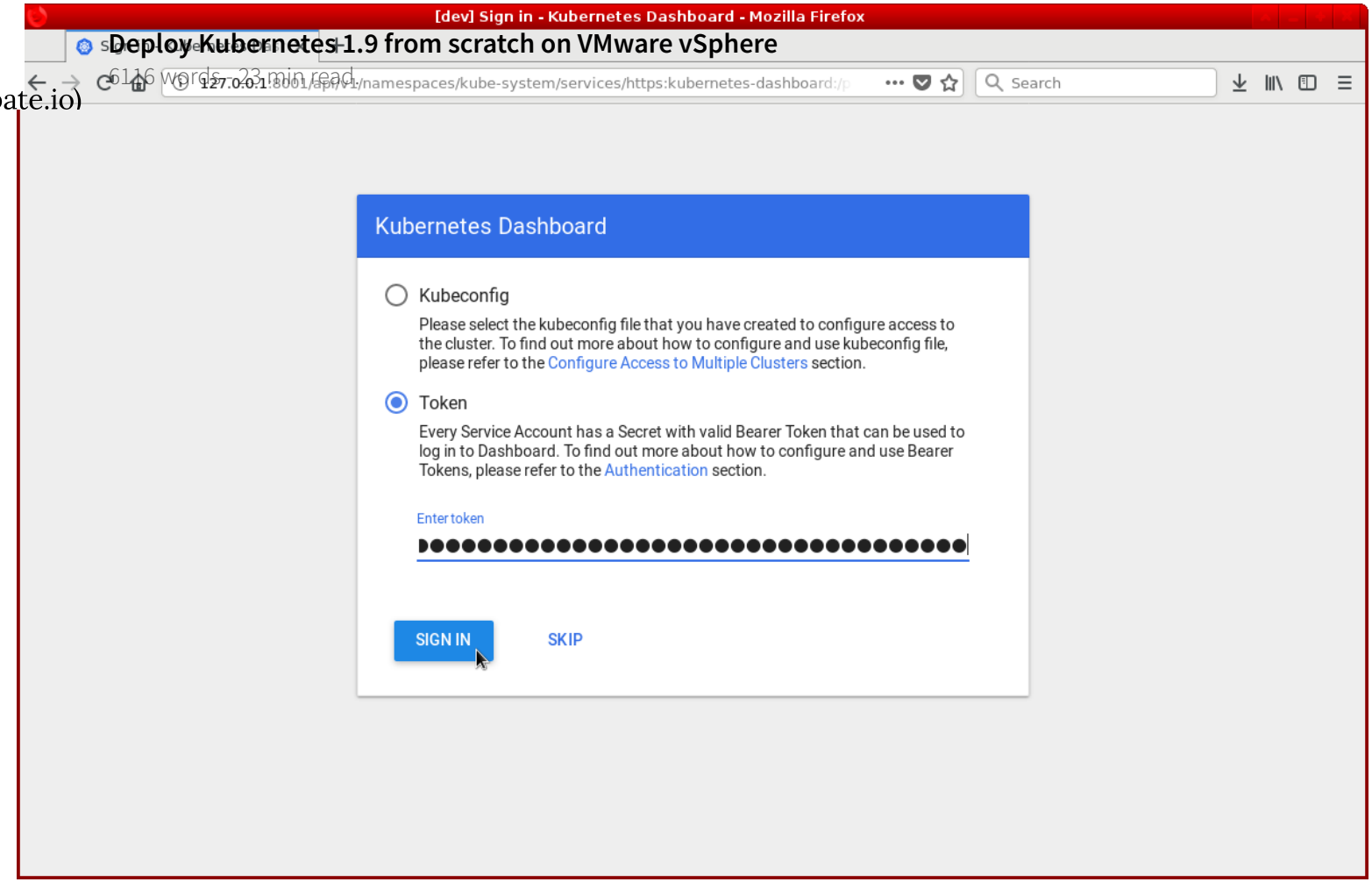
```
$ kubectl proxy
```

6- Browse to http://localhost:8001/ui (http://localhost:8001/ui).

7- Select Token and paste the token from step 4.

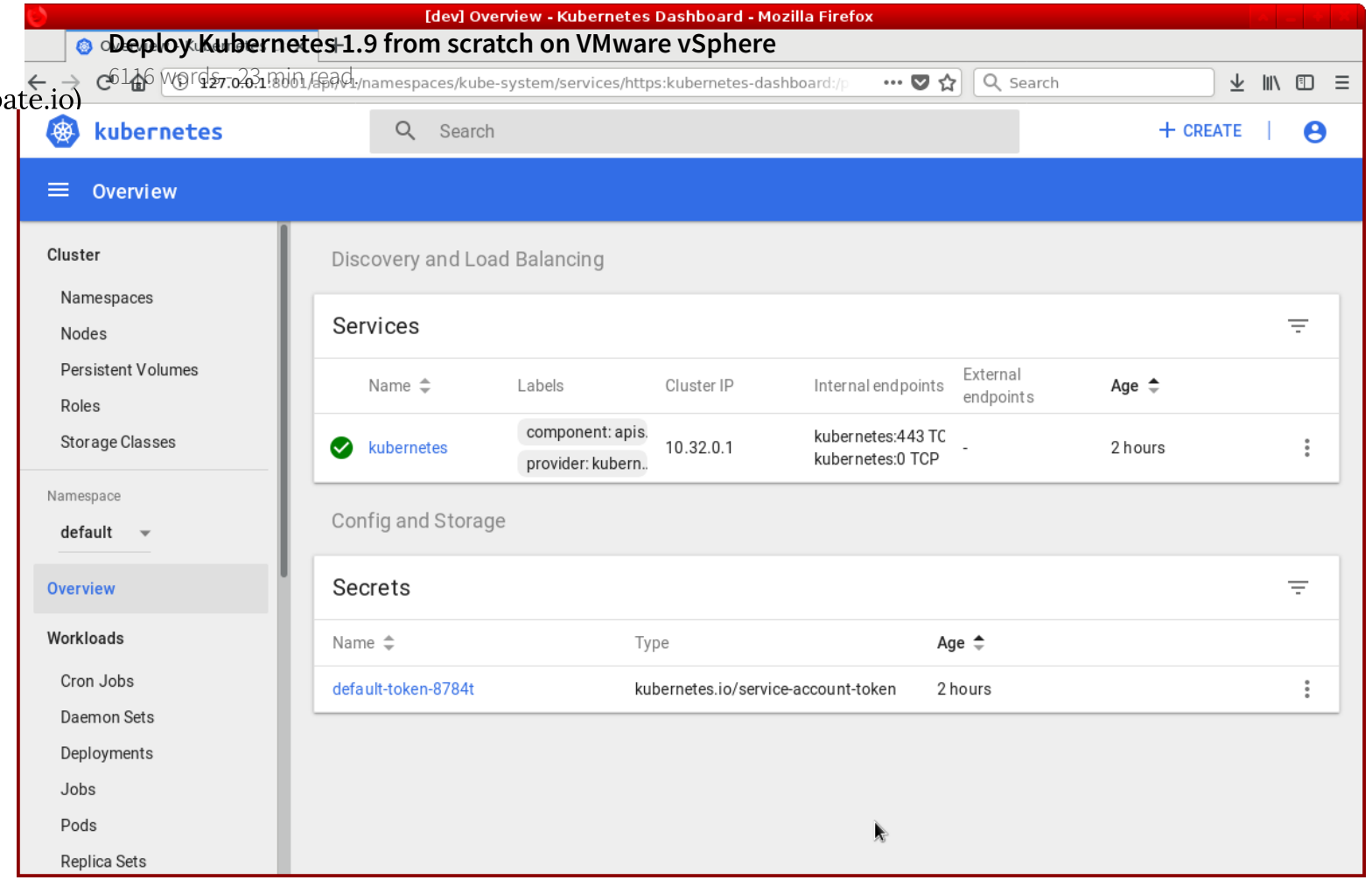
Deploy Kubernetes 1.9 from scratch on VMware vSphere

6116 words - 23 min read



Deploy Kubernetes 1.9 from scratch on VMware vSphere

6116 words - 23 min read



Tags: Cloud (/tag/cloud/), VMware (/tag/vmware/), vSphere (/tag/vsphere/), Kubernetes (/tag/kubernetes/), Container (/tag/container/)

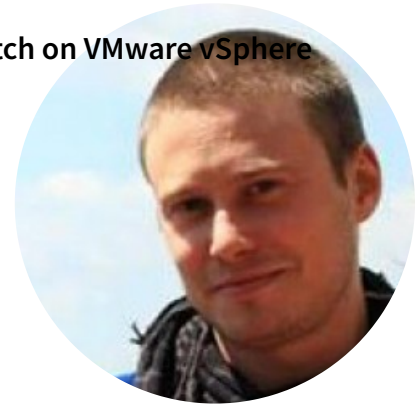
Share this post:

(https://twitter.com/share?text=Deploy%20Kubernetes%201.9%20from%20scratch%20on%20VM

kubernetes-1-9-from-scratch-on-vmware-vsphere/)
(https://www.facebook.com/sharer/sharer.php?u=https://blog.inkubate.io/deploy-kubernetes-1-9-from-scratch-on-vmware-vsphere/)
(https://plus.google.com/share?url=https://blog.inkubate.io/deploy-kubernetes-1-9-from-scratch-on-vmware-vsphere/)

« PREVIOUS (/DEPLOY-VMWARE-VSPHERE-INTEGRATED-CONTAINERS/)
Deploy VMware vSphere Integrated Containers 1.2.1 (/deploy-vmware-vsphere-integrated-containers/)

NEXT » (/MONITOR-KUBERNETES-WITH-PROMETHEUS/)
Monitor Kubernetes with Prometheus (/monitor-kubernetes-with-prometheus/)



Author

SIMON GUYENNET (/AUTHOR/SIMON/)

Comments

ALSO ON **BLOG.INKUBATE.IO**

Access your Kubernetes cluster ...

3 years ago • 30 comments

Out of the box, the Kubernetes authentication is not very user-friendly for ...

Configure Swift on OpenStack Ocata ...

4 years ago • 3 comments

Swift is an OpenStack project which provides object storage to your ...

Automatically generate signed ...

3 years ago • 6 comments

In this lab, we will see how to automatically generate signed SSL certificates for ...

Monitor Kubern with Prometheus

4 years ago • 4 comment

Prometheus is a moni tool originally created SoundCloud. The proj

Vojtech Vavra • 2 years ago

Hi Simon
Thank you for your job but I am missing configuration for "containerd ,cri-containerd" unit files.
Thanks !