Edit This Page

Setup

This section provides instructions for installing Kubernetes and setting up a Kubernetes cluster. For an overview of the different options, see Picking the Right Solution.

Edit this Page

Edit This Page

Troubleshooting kubeadm

-
  -
    -
      - `ebtables` or some similar executable not found during installation
      - kubeadm blocks waiting for control plane during installation
      - kubeadm blocks when removing managed containers
      - Pods in `RunContainerError`, `CrashLoopBackOff` or `Error` state
      - `coredns` (or `kube-dns`) is stuck in the `Pending` state
      - `HostPort` services do not work
      - Pods are not accessible via their Service IP
      - TLS certificate errors
      - Default NIC When using flannel as the pod network in Vagrant
      - Non-public IP used for containers
      - Services with externalTrafficPolicy=Local are not reachable

`ebtables` or some similar executable not found during installation

If you see the following warnings while running `kubeadm init`

```
[preflight] WARNING: ebtables not found in system path
[preflight] WARNING: ethtool not found in system path
```

Then you may be missing `ebtables`, `ethtool` or a similar executable on your node. You can install them with the following commands:

- For Ubuntu/Debian users, run `apt install ebtables ethtool`.
- For CentOS/Fedora users, run `yum install ebtables ethtool`.

kubeadm blocks waiting for control plane during installation

If you notice that `kubeadm init` hangs after printing out the following line:

```
[apiclient] Created API client, waiting for the control plane to become ready
```

This may be caused by a number of problems. The most common are:

- network connection problems. Check that your machine has full network connectivity before continuing.
- the default cgroup driver configuration for the kubelet differs from that used by Docker. Check the system log file (e.g. `/var/log/message`) or examine the output from `journalctl -u kubelet`. If you see something like the following:

```
error: failed to run Kubelet: failed to create kubelet:
misconfiguration: kubelet cgroup driver: "systemd" is different from docker cgroup driver
```

There are two common ways to fix the cgroup driver problem:

1. Install docker again following instructions here.
2. Change the kubelet config to match the Docker cgroup driver manually, you can refer to Configure cgroup driver used by kubelet on Master Node for detailed instructions.

- control plane Docker containers are crashlooping or hanging. You can check this by running `docker ps` and investigating each container by running `docker logs`.

kubeadm blocks when removing managed containers

The following could happen if Docker halts and does not remove any Kubernetes-managed containers:

```
sudo kubeadm reset
[preflight] Running pre-flight checks
[reset] Stopping the kubelet service
[reset] Unmounting mounted directories in "/var/lib/kubelet"
[reset] Removing kubernetes-managed containers
(block)
```

A possible solution is to restart the Docker service and then re-run `kubeadm reset`:

```
sudo systemctl restart docker.service
sudo kubeadm reset
```

Inspecting the logs for docker may also be useful:

```
journalctl -ul docker
```

Pods in `RunContainerError`, `CrashLoopBackOff` or `Error` state

Right after `kubeadm init` there should not be any pods in these states.

- If there are pods in one of these states *right after* `kubeadm init`, please open an issue in the kubeadm repo. `coredns` (or `kube-dns`) should be in the `Pending` state until you have deployed the network solution.
- If you see Pods in the `RunContainerError`, `CrashLoopBackOff` or `Error` state after deploying the network solution and nothing happens to `coredns` (or `kube-dns`), it's very likely that the Pod Network solution that you installed is somehow broken. You might have to grant it more RBAC privileges or use a newer version. Please file an issue in the Pod Network providers' issue tracker and get the issue triaged there.

`coredns` (or `kube-dns`) is stuck in the `Pending` state

kubeadm does not install a pod network solution by default. You have to install a Pod Network before `coredns` (or `kube-dns`) pods will be scheduled.

`HostPort` services do not work

The `HostPort` and `HostIP` functionality is available depending on your Pod Network provider. Please contact the author of the Pod Network solution to find out whether `HostPort` and `HostIP` functionality are available.

Calico, Canal, and Flannel CNI providers are verified to support HostPort.

For more information, see the CNI portmap documentation.

If your network provider does not support the portmap CNI plugin, you may need to use the NodePort feature of services or use `HostNetwork=true`.

Pods are not accessible via their Service IP

- Many network add-ons do not yet enable hairpin mode which allows pods to access themselves via their Service IP. This is an issue related to CNI. Please contact the network add-on provider to get the latest status of their support for hairpin mode.
- If you are using VirtualBox (directly or via Vagrant), you will need to ensure that `hostname -i` returns a routable IP address. By default the first interface is connected to a non-routable host-only network. A work around is to modify `/etc/hosts`, see this Vagrantfile for an example.

TLS certificate errors

The following error indicates a possible certificate mismatch.

```
# kubectl get pods
Unable to connect to the server: x509: certificate signed by unknown authority (possibly bec
```

- Verify that the `$HOME/.kube/config` file contains a valid certificate, and regenerate a certificate if necessary. The certificates in a kubeconfig file are base64 encoded. The `base64 -d` command can be used to decode the certificate and `openssl x509 -text -noout` can be used for viewing the certificate information.
- Another workaround is to overwrite the existing `kubeconfig` for the "admin" user:

```
mv  $HOME/.kube $HOME/.kube.bak
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Default NIC When using flannel as the pod network in Vagrant

The following error might indicate that something was wrong in the pod network:

```
Error from server (NotFound): the server could not find the requested resource
```

- If you're using flannel as the pod network inside Vagrant, then you will have to specify the default interface name for flannel.

Vagrant typically assigns two interfaces to all VMs. The first, for which all hosts are assigned the IP address `10.0.2.15`, is for external traffic that gets NATed.

This may lead to problems with flannel, which defaults to the first interface on a host. This leads to all hosts thinking they have the same public IP address. To prevent this, pass the `--iface eth1` flag to flannel so that the second interface is chosen.

Non-public IP used for containers

In some situations `kubectl logs` and `kubectl run` commands may return with the following errors in an otherwise functional cluster:

```
Error from server: Get https://10.19.0.41:10250/containerLogs/default/mysql-ddc65b868-glc5m/
```

- This may be due to Kubernetes using an IP that can not communicate with other IPs on the seemingly same subnet, possibly by policy of the machine provider.
- Digital Ocean assigns a public IP to `eth0` as well as a private one to be used internally as anchor for their floating IP feature, yet `kubelet` will pick the latter as the node's `InternalIP` instead of the public one.

Use `ip addr show` to check for this scenario instead of `ifconfig` because `ifconfig` will not display the offending alias IP address. Alternatively an API endpoint specific to Digital Ocean allows to query for the anchor IP from the droplet:

```
curl http://169.254.169.254/metadata/v1/interfaces/public/0/anchor_ipv4/address
```

The workaround is to tell `kubelet` which IP to use using `--node-ip`. When using Digital Ocean, it can be the public one (assigned to `eth0`) or the private one (assigned to `eth1`) should you want to use the optional private network. The KubeletExtraArgs section of the MasterConfiguration file can be used for this.

Then restart `kubelet`:

```
systemctl daemon-reload
systemctl restart kubelet
```

Services with externalTrafficPolicy=Local are not reachable

On nodes where the hostname for the kubelet is overridden using the `--hostname-override` option, kube-proxy will default to treating 127.0.0.1 as the node IP, which results in rejecting connections for Services configured for `externalTrafficPolicy=Local`. This situation can be verified by checking the output of `kubectl -n kube-system logs <kube-proxy pod name>`:

```
W0507 22:33:10.372369       1 server.go:586] Failed to retrieve node info: nodes "ip-10-0-23
W0507 22:33:10.372474       1 proxier.go:463] invalid nodeIP, initializing kube-proxy with 1
```

A workaround for this is to modify the kube-proxy DaemonSet in the following way:

```
kubectl -n kube-system patch --type json daemonset kube-proxy -p "$(cat <<'EOF'
[
    {
        "op": "add",
        "path": "/spec/template/spec/containers/0/env",
        "value": [
            {
                "name": "NODE_NAME",
                "valueFrom": {
                    "fieldRef": {
                        "apiVersion": "v1",
                        "fieldPath": "spec.nodeName"
                    }
                }
            }
        ]
    },
    {
        "op": "add",
        "path": "/spec/template/spec/containers/0/command/-",
        "value": "--hostname-override=${NODE_NAME}"
    }
]
EOF
)"
```

Create an Issue Edit this Page

Edit This Page

Creating HA clusters with kubeadm

This guide shows you how to install and set up a highly available Kubernetes cluster using kubeadm.

This document shows you how to perform setup tasks that kubeadm doesn't perform: provision hardware; configure multiple systems; and load balancing.

> **Note:** This guide is only one potential solution, and there are many ways to configure a highly available cluster. If a better solution works for you, please use it. If you find a better solution that can be adopted by the community, feel free to contribute it back.

- Before you begin
- Installing prerequisites on masters
- Setting up an HA etcd cluster
- Acquire etcd certs
- Run kubeadm init on master0
- Run kubeadm init on master1 and master2
- Add master1 and master2 to load balancer
- Install CNI network
- Install workers
- Configure workers

Before you begin

- Three machines that meet kubeadm's minimum requirements for the masters
- Three machines that meet kubeadm's minimum requirements for the workers
- **Optional:** At least three machines that meet kubeadm's minimum requirements if you intend to host etcd on dedicated nodes (see information below)

- 1GB or more of RAM per machine (any less will leave little room for your apps)
- Full network connectivity between all machines in the cluster (public or private network is fine)

Installing prerequisites on masters

For each master that has been provisioned, follow the installation guide on how to install kubeadm and its dependencies. At the end of this step, you should have all the dependencies installed on each master.

Setting up an HA etcd cluster

For highly available setups, you will need to decide how to host your etcd cluster. A cluster is composed of at least 3 members. We recommend one of the following models:

- Hosting etcd cluster on separate compute nodes (Virtual Machines)
- Hosting etcd cluster on the master nodes.

While the first option provides more performance and better hardware isolation, it is also more expensive and requires an additional support burden.

For **Option 1**: create 3 virtual machines that follow CoreOS's hardware recommendations. For the sake of simplicity, we will refer to them as `etcd0`, `etcd1` and `etcd2`.

For **Option 2**: you can skip to the next step. Any reference to `etcd0`, `etcd1` and `etcd2` throughout this guide should be replaced with `master0`, `master1` and `master2` accordingly, since your master nodes host etcd.

Create etcd CA certs

1. Install `cfssl` and `cfssljson` on all etcd nodes:

   ```
   curl -o /usr/local/bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
   curl -o /usr/local/bin/cfssljson https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
   chmod +x /usr/local/bin/cfssl*
   ```

2. SSH into `etcd0` and run the following:

   ```
   mkdir -p /etc/kubernetes/pki/etcd
   cd /etc/kubernetes/pki/etcd

   cat >ca-config.json <<EOF
   {
      "signing": {
   ```

```
        "default": {
            "expiry": "43800h"
        },
        "profiles": {
            "server": {
                "expiry": "43800h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "client auth"
                ]
            },
            "client": {
                "expiry": "43800h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "client auth"
                ]
            },
            "peer": {
                "expiry": "43800h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "client auth"
                ]
            }
        }
    }
}
EOF
```

```
cat >ca-csr.json <<EOF
{
    "CN": "etcd",
    "key": {
        "algo": "rsa",
        "size": 2048
    }
}
EOF
```

Optional: You can modify `ca-csr.json` to add a section for `names`. See the CFSSL wiki for an example.

3. Next, generate the CA certs:

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

Generate etcd client certs

Generate the client certificates. While on `etcd0`, run the following:

```
cat >client.json <<EOF
{
  "CN": "client",
  "key": {
      "algo": "ecdsa",
      "size": 256
  }
}
EOF
```

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=client client.js
```

Both `client.pem` and `client-key.pem` are created.

Create SSH access

In order to copy certs between machines, you must enable SSH access for `scp`.

1. First, open new tabs in your shell for `etcd1` and `etcd2`. Ensure you are SSHed into all three machines and then run the following (it will be a lot quicker if you use tmux syncing - to do this in iTerm enter `cmd+shift+i`):

   ```
   export PEER_NAME=$(hostname)
   export PRIVATE_IP=$(ip addr show eth1 | grep -Po 'inet \K[\d.]+')
   ```

   Make sure that `eth1` corresponds to the network interface for the IPv4 address of the private network. This might vary depending on your networking setup, so please check by running `echo $PRIVATE_IP` before continuing.

2. Next, generate some SSH keys for the boxes:

   ```
   ssh-keygen -t rsa -b 4096 -C "<email>"
   ```

   Make sure to replace `<email>` with your email, a placeholder, or an empty string. Keep hitting enter until files exist in `~/.ssh`.

3. Output the contents of the public key file for `etcd1` and `etcd2`:

   ```
   cat ~/.ssh/id_rsa.pub
   ```

4. Finally, copy the output for each and paste them into `etcd0`'s `~/.ssh/authorized_keys` file. This will permit `etcd1` and `etcd2` to SSH in to the machine.

Generate etcd server and peer certs

1. In order to generate certs, each etcd machine needs the root CA generated by `etcd0`. On `etcd1` and `etcd2`, run the following:

   ```
   mkdir -p /etc/kubernetes/pki/etcd
   cd /etc/kubernetes/pki/etcd
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/ca.pem .
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/ca-key.pem .
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/client.pem .
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/client-key.pem .
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/ca-config.json .
   ```

   Where `<etcd0-ip-address>` corresponds to the public or private IPv4 of `etcd0`.

2. Once this is done, run the following on all etcd machines:

   ```
   cfssl print-defaults csr > config.json
   sed -i '0,/CN/{s/example\.net/'"$PEER_NAME"'/}' config.json
   sed -i 's/www\.example\.net/'"$PRIVATE_IP"'/' config.json
   sed -i 's/example\.net/'"$PEER_NAME"'/' config.json
   ```

   ```
   cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=server con
   cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=peer confi
   ```

   The above will replace the default configuration with your machine's hostname as the peer name, and its IP addresses. Make sure these are correct before generating the certs. If you found an error, reconfigure `config.json` and re-run the `cfssl` commands.

This results in the following files: `peer.pem`, `peer-key.pem`, `server.pem`, `server-key.pem`.

- Choose one...
- systemd
- Static Pods

Please select one of the tabs to see installation instructions for the respective way to set up a virtual IP.

1. First, install etcd binaries:

   ```
   ETCD_VERSION="v3.1.12" curl -sSL https://github.com/coreos/etcd/releases/download/${ET
   ```

   It is worth noting that etcd v3.1.12 is the preferred version for Kubernetes v1.10. For other versions of Kubernetes please consult the changelog.

Also, please realize that most distributions of Linux already have a version of etcd installed, so you will be replacing the system default.

2. Next, generate the environment file that systemd will use:

```
touch /etc/etcd.env
echo "PEER_NAME=${PEER_NAME}" >> /etc/etcd.env
echo "PRIVATE_IP=${PRIVATE_IP}" >> /etc/etcd.env
```

3. Now copy the systemd unit file:

```
cat >/etc/systemd/system/etcd.service <<EOF
[Unit]
Description=etcd
Documentation=https://github.com/coreos/etcd
Conflicts=etcd.service
Conflicts=etcd2.service

[Service]
EnvironmentFile=/etc/etcd.env
Type=notify
Restart=always
RestartSec=5s
LimitNOFILE=40000
TimeoutStartSec=0

ExecStart=/usr/local/bin/etcd --name <name> --data-dir /var/lib/etcd --listen-client-u

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

Make sure you replace `<etcd0-ip-address>`, `<etcd1-ip-address>` and `<etcd2-ip-address>` with the appropriate IPv4 addresses. Replace `<name>` with the name of this etcd member. Replace `<etcd-listen-ip>` with the IPv4 address of this etcd node. Replace `<etcd0>`, `<etcd1>` and `<etcd2>` with real hostnames of each machine. These machines must be able to reach each other using DNS or by adding records to `/etc/hosts`.

4. Finally, launch etcd:

```
systemctl daemon-reload
systemctl start etcd
```

5. Check that it launched successfully:

```
systemctl status etcd
```

Note: This is only supported on nodes that have the all dependencies for the kubelet installed. If you are hosting etcd on the master nodes, this has already been set up. If you are hosting etcd on dedicated nodes, you should either use systemd or run the installation guide on each dedicated etcd machine.

Run the following to generate the manifest file:

```
cat >/etc/kubernetes/manifests/etcd.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  labels:
    component: etcd
    tier: control-plane
  name: <name>
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd --name <name>
    - --data-dir /var/lib/etcd
    - --listen-client-urls https://<etcd-listen-ip>:2379
    - --advertise-client-urls https://<etcd-listen-ip>:2379
    - --listen-peer-urls https://<etcd-listen-ip>:2380
    - --initial-advertise-peer-urls https://<etcd-listen-ip>:2380
    - --cert-file=/certs/server.pem
    - --key-file=/certs/server-key.pem
    - --client-cert-auth
    - --trusted-ca-file=/certs/ca.pem
    - --peer-cert-file=/certs/peer.pem
    - --peer-key-file=/certs/peer-key.pem
    - --peer-client-cert-auth
    - --peer-trusted-ca-file=/certs/ca.pem
    - --initial-cluster etcd0=https://<etcd0-ip-address>:2380,etcd1=https://<etcd1-ip-addr
    - --initial-cluster-token my-etcd-token
    - --initial-cluster-state new
    image: k8s.gcr.io/etcd-amd64:3.1.10
    livenessProbe:
      httpGet:
        path: /health
        port: 2379
        scheme: HTTP
      initialDelaySeconds: 15
      timeoutSeconds: 15
    name: etcd
    env:
    - name: PUBLIC_IP
      valueFrom:
        fieldRef:
        fieldPath: status.hostIP
    - name: PRIVATE_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: PEER_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    volumeMounts:
    - mountPath: /var/lib/etcd
      name: etcd
    - mountPath: /certs
      name: certs
hostNetwork: true
volumes:
- hostPath:
    path: /var/lib/etcd
    type: DirectoryOrCreate
  name: etcd
- hostPath:
    path: /etc/kubernetes/pki/etcd
  name: certs
EOF
```

Make sure you replace:

- `<name>` with the name of the node you're running on (for example, `etcd0`, `etcd1` or `etcd2`)
- `<etcd-listen-ip>` with the public IPv4 address of the node you're running on
- `<etcd0-ip-address>`, `<etcd1-ip-address>` and `<etcd2-ip-address>` with the public IPv4s of the other machines that host etcd.

- Choose one...
- Cloud
- On-Site

Please select one of the tabs to see installation instructions for the respective way to set up a virtual IP.

Some examples of cloud provider solutions are:

- AWS Elastic Load Balancer
- GCE Load Balancing
- Azure

You will need to ensure that the load balancer routes to **just `master0` on port 6443**. This is because kubeadm will perform health checks using the load balancer IP. Since `master0` is set up individually first, the other masters will not have running apiservers, which will result in kubeadm hanging indefinitely.

If possible, use a smart load balancing algorithm like “least connections”, and use health checks so unhealthy nodes can be removed from circulation. Most providers will provide these features.

In an on-site environment there may not be a physical load balancer available. Instead, a virtual IP pointing to a healthy master node can be used. There are a number of solutions for this including keepalived, Pacemaker and probably many others, some with and some without load balancing.

As an example we outline a simple setup based on keepalived. Depending on environment and requirements people may prefer different solutions. The configuration shown here provides an *active/passive* failover without load balancing. If required, load balancing can by added quite easily by setting up HAProxy, NGINX or similar on the master nodes (not covered in this guide).

1. Install keepalived, e.g. using your distribution’s package manager. The configuration shown here works with version `1.3.5` but is expected to work with may other versions. Make sure to have it enabled (chkconfig, systemd, ...) so that it starts automatically when the respective node comes up.

2. Create the following configuration file `/etc/keepalived/keepalived.conf` on all master nodes:

```
! Configuration File for keepalived
global_defs {
 router_id LVS_DEVEL
}

vrrp_script check_apiserver {
 script "/etc/keepalived/check_apiserver.sh"
 interval 3
 weight -2
 fall 10
 rise 2
}

vrrp_instance VI_1 {
   state <STATE>
   interface <INTERFACE>
   virtual_router_id 51
   priority <PRIORITY>
   authentication {
       auth_type PASS
       auth_pass 4be37dc3b4c90194d1600c483e10ad1d
   }
   virtual_ipaddress {
       <VIRTUAL-IP>
   }
   track_script {
       check_apiserver
   }
}
```

In the section `vrrp_instance VI_1`, change the following lines depending on your setup:

- `state` is either `MASTER` (on the first master nodes) or `BACKUP` (the other master nodes).
- `interface` is the name of an existing public interface to bind the virtual IP to (usually the primary interface).
- `priority` should be higher for the first master node, e.g. 101, and lower for the others, e.g. 100.
- `auth_pass` use any random string here.
- `virtual_ipaddresses` should contain the virtual IP for the master nodes.

3. Install the following health check script to `/etc/keepalived/check_apiserver.sh` on all master nodes:

```
#!/bin/sh

errorExit() {
   echo "*** $*" 1>&2
   exit 1
}

curl --silent --max-time 2 --insecure https://localhost:6443/ -o /dev/null || errorExi
if ip addr | grep -q <VIRTUAL-IP>; then
   curl --silent --max-time 2 --insecure https://<VIRTUAL-IP>:6443/ -o /dev/null || er
fi
```

Replace the `<VIRTUAL-IP>` by your chosen virtual IP.

4. Restart keepalived. While no Kubernetes services are up yet it will log health check fails on all master nodes. This will stop as soon as the first master node has been bootstrapped.

Acquire etcd certs

Only follow this step if your etcd is hosted on dedicated nodes (**Option 1**). If you are hosting etcd on the masters (**Option 2**), you can skip this step since you've already generated the etcd certificates on the masters.

1. Generate SSH keys for each of the master nodes by following the steps in the create ssh access section. After doing this, each master will have an SSH key in `~/.ssh/id_rsa.pub` and an entry in `etcd0`'s `~/.ssh/authorized_keys` file.

2. Run the following:

   ```
   mkdir -p /etc/kubernetes/pki/etcd
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/ca.pem /etc/kubernetes/pki/etcd
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/client.pem /etc/kubernetes/pki/et
   scp root@<etcd0-ip-address>:/etc/kubernetes/pki/etcd/client-key.pem /etc/kubernetes/pk
   ```

Run kubeadm init on master0

1. In order for kubeadm to run, you first need to write a configuration file:

```
cat >config.yaml <<EOF
apiVersion: kubeadm.k8s.io/v1alpha1
kind: MasterConfiguration
api:
  advertiseAddress: <virtual-ip>
  controlPlaneEndpoint: <virtual-ip>
etcd:
  endpoints:
  - https://<etcd0-ip-address>:2379
  - https://<etcd1-ip-address>:2379
  - https://<etcd2-ip-address>:2379
  caFile: /etc/kubernetes/pki/etcd/ca.pem
  certFile: /etc/kubernetes/pki/etcd/client.pem
  keyFile: /etc/kubernetes/pki/etcd/client-key.pem
networking:
  podSubnet: <podCIDR>
apiServerCertSANs:
- <virtual-ip>
- <private-ip>
apiServerExtraArgs:
  apiserver-count: "3"
EOF
```

Ensure that the following placeholders are replaced:

- `<private-ip>` with the private IPv4 of the master server.
- `<etcd0-ip>`, `<etcd1-ip>` and `<etcd2-ip>` with the IP addresses of your three etcd nodes
- `<podCIDR>` with your Pod CIDR. Please read the CNI network section of the docs for more information. Some CNI providers do not require a value to be set.

- `<virtual-ip>` with the virtual IP. Please read setting up a master load balancer section of the docs for more information.

  Note: If you are using Kubernetes 1.9+, you can replace the `apiserver-count: 3` extra argument with `endpoint-reconciler-type: lease`. For more information, see the documentation.

1. When this is done, run kubeadm: `bash    kubeadm init --config=config.yaml`

Run kubeadm init on master1 and master2

Before running kubeadm on the other masters, you need to first copy the K8s CA cert from `master0`. To do this, you have two options:

Option 1: Copy with scp

1. Follow the steps in the create ssh access section, but instead of adding to `etcd0`'s `authorized_keys` file, add them to `master0`.
2. Once you've done this, run:

   ```
   scp root@<master0-ip-address>:/etc/kubernetes/pki/* /etc/kubernetes/pki
   rm /etc/kubernetes/pki/apiserver*
   ```

Option 2: Copy paste

Copy the contents of `/etc/kubernetes/pki/ca.crt`, `/etc/kubernetes/pki/ca.key`, `/etc/kubernetes/pki/sa.key` and `/etc/kubernetes/pki/sa.pub` and create these files manually on `master1` and `master2`.

When this is done, you can follow the previous step to install the control plane with kubeadm.

Add master1 and master2 to load balancer

Once kubeadm has provisioned the other masters, you can add them to the load balancer pool.

Install CNI network

Follow the instructions here to install the pod network. Make sure this corresponds to whichever pod CIDR you provided in the master configuration file.

Install workers

Next provision and set up the worker nodes. To do this, you will need to provision at least 3 Virtual Machines.

1. To configure the worker nodes, follow the same steps as non-HA workloads.

Configure workers

1. Reconfigure kube-proxy to access kube-apiserver via the load balancer:

```
kubectl get configmap -n kube-system kube-proxy -o yaml > kube-proxy-cm.yaml
sed -i 's#server:.*#server: https://<masterLoadBalancerFQDN>:6443#g' kube-proxy-cm.yam
kubectl apply -f kube-proxy-cm.yaml --force
# restart all kube-proxy pods to ensure that they load the new configmap
kubectl delete pod -n kube-system -l k8s-app=kube-proxy
```

2. Reconfigure the kubelet to access kube-apiserver via the load balancer:

```
sudo sed -i 's#server:.*#server: https://<masterLoadBalancerFQDN>:6443#g' /etc/kuberne
sudo systemctl restart kubelet
```

Create an Issue Edit this Page

Edit This Page

Creating a single master cluster with kubeadm

kubeadm helps you bootstrap a minimum viable Kubernetes cluster that conforms to best practices. With kubeadm, your cluster should pass Kubernetes Conformance tests. Kubeadm also supports other cluster lifecycle functions, such as upgrades, downgrade, and managing bootstrap tokens.

Because you can install kubeadm on various types of machine (e.g. laptop, server, Raspberry Pi, etc.), it's well suited for integration with provisioning systems such as Terraform or Ansible.

kubeadm's simplicity means it can serve a wide range of use cases:

- New users can start with kubeadm to try Kubernetes out for the first time.
- Users familiar with Kubernetes can spin up clusters with kubeadm and test their applications.
- Larger projects can include kubeadm as a building block in a more complex system that can also include other installer tools.

kubeadm's overall feature state is **Beta**.

Before you begin

- One or more machines running a deb/rpm-compatible OS, for example Ubuntu or CentOS
- 2 GB or more of RAM per machine. Any less leaves little room for your apps.
- 2 CPUs or more on the master
- Full network connectivity among all machines in the cluster. A public or private network is fine.

Objectives

- Install a single master Kubernetes cluster or high availability cluster
- Install a Pod network on the cluster so that your Pods can talk to each other

Instructions

Installing kubeadm on your hosts

See "Installing kubeadm".

> **Note:** If you have already installed kubeadm, run `apt-get update && apt-get upgrade` or `yum update` to get the latest version of kubeadm.
>
> When you upgrade, the kubelet restarts every few seconds as it waits in a crashloop for kubeadm to tell it what to do. This crashloop is expected and normal. After you initialize your master, the kubelet runs normally.

Initializing your master

The master is the machine where the control plane components run, including etcd (the cluster database) and the API server (which the kubectl CLI communicates with).

1. Choose a pod network add-on, and verify whether it requires any arguments to be passed to kubeadm initialization. Depending on which third-party provider you choose, you might need to set the `--pod-network-cidr` to a provider-specific value. See Installing a pod network add-on.
2. (Optional) Unless otherwise specified, kubeadm uses the network interface associated with the default gateway to advertise the master's IP. To use a different network interface, specify the `--apiserver-advertise-address=<ip-address>` argument to `kubeadm init`. To deploy an IPv6 Kubernetes cluster using IPv6 addressing, you must specify an IPv6 address, for example `--apiserver-advertise-address=fd00::101`

Now run:

```
kubeadm init <args>
```

More information

For more information about `kubeadm init` arguments, see the kubeadm reference guide.

For a complete list of configuration options, see the configuration file documentation.

To customize control plane components, including optional IPv6 assignment to liveness probe for control plane components and etcd server, provide extra arguments to each component as documented in custom arguments.

To run `kubeadm init` again, you must first tear down the cluster.

If you join a node with a different architecture to your cluster, create a separate Deployment or DaemonSet for `kube-proxy` and `kube-dns` on the node. This is because the Docker images for these components do not currently support multi-architecture.

`kubeadm init` first runs a series of prechecks to ensure that the machine is ready to run Kubernetes. These prechecks expose warnings and exit on errors. `kubeadm init` then downloads and installs the cluster control plane components. This may take several minutes. The output should look like:

```
[init] Using Kubernetes version: vX.Y.Z
[preflight] Running pre-flight checks

... (log output of initialization workflow) ...

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run (as a regular user):

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the addon options listed at:
  http://kubernetes.io/docs/admin/addons/

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

  kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha2
```

To make kubectl work for your non-root user, run these commands, which are also part of the `kubeadm init` output:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Alternatively, if you are the `root` user, you can run:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

Make a record of the `kubeadm join` command that `kubeadm init` outputs. You need this command to join nodes to your cluster.

The token is used for mutual authentication between the master and the joining nodes. The token included here is secret. Keep it safe, because anyone with this token can add authenticated nodes to your cluster. These tokens can be listed, created, and deleted with the `kubeadm token` command. See the kubeadm reference guide.

Installing a pod network add-on

> **Caution:** This section contains important information about installation and deployment order. Read it carefully before proceeding.

You must install a pod network add-on so that your pods can communicate with each other.

The network must be deployed before any applications. An internal helper service, kube-dns, will not start up before a network is installed. kubeadm supports only Container Network Interface (CNI) based networks. It does not support kubenet.

Several projects provide Kubernetes pod networks using CNI, some of which also support Network Policy. See the add-ons page for a complete list of available network add-ons. - IPv6 support was added in CNI v0.6.0. - CNI bridge and local-ipam are the only supported IPv6 network plugins in Kubernetes version 1.9.

Note that kubeadm sets up a more secure cluster by default and enforces use of [RBAC]. Make sure that your network manifest supports RBAC.

You can install a pod network add-on with the following command:

```
kubectl apply -f <add-on.yaml>
```

You can install only one pod network per cluster.

- Choose one...
- Calico
- Canal
- Flannel
- Kube-router
- Romana
- Weave Net

Please select one of the tabs to see installation instructions for the respective third-party Pod Network Provider.

For more information about using Calico, see Quickstart for Calico on Kubernetes, Installing Calico for policy and networking, and other related resources.

In order for Network Policy to work correctly, you need to pass `--pod-network-cidr=192.168.0.0/16` to `kubeadm init`. Note that Calico works on `amd64` only.

```
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installatio
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installatio
```

Canal uses Calico for policy and Flannel for networking. Refer to the Calico documentation for the official getting started guide.

For Canal to work correctly, `--pod-network-cidr=10.244.0.0/16` has to be passed to `kubeadm init`. Note that Canal works on `amd64` only.

```
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installatio
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installatio
```

For `flannel` to work correctly, `--pod-network-cidr=10.244.0.0/16` has to be passed to `kubeadm init`. Note that `flannel` works on `amd64`, `arm`, `arm64` and `ppc64le`. For it to work on a platform other than `amd64`, you must manually download the manifest and replace `amd64` occurrences with your chosen platform.

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/v0.10.0/Documentation/kube
```

For more information about `flannel`, see the CoreOS flannel repository on GitHub.

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

Kube-router relies on kube-controller-manager to allocate pod CIDR for the nodes. Therefore, use `kubeadm init` with the `--pod-network-cidr` flag.

Kube-router provides pod networking, network policy, and high-performing IP Virtual Server(IPVS)/Linux Virtual Server(LVS) based service proxy.

For information on setting up Kubernetes cluster with Kube-router using kubeadm, please see official setup guide.

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

The official Romana set-up guide is here.

Romana works on `amd64` only.

```
kubectl apply -f https://raw.githubusercontent.com/romana/romana/master/containerize/specs/r
```

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

The official Weave Net set-up guide is here.

Weave Net works on `amd64`, `arm`, `arm64` and `ppc64le` without any extra action required. Weave Net sets hairpin mode by default. This allows Pods to access themselves via their Service IP address if they don't know their PodIP.

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 
```

Once you install a pod network, you can confirm that it works by checking that the kube-dns pod is Running in the output of `kubectl get pods --all-namespaces`. Once the kube-dns pod is up and running, you can continue by joining your nodes.

If your network is not working or kube-dns is not in the Running state, check out our troubleshooting docs.

Master Isolation

By default, your cluster will not schedule pods on the master for security reasons. If you want to be able to schedule pods on the master, e.g. for a single-machine Kubernetes cluster for development, run:

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

With output looking something like:

```
node "test-01" untainted
taint key="dedicated" and effect="" not found.
taint key="dedicated" and effect="" not found.
```

This will remove the `node-role.kubernetes.io/master` taint from any nodes that have it, including the master node, meaning that the scheduler will then be able to schedule pods everywhere.

Joining your nodes {join-nodes}

The nodes are where your workloads (containers and pods, etc) run. To add new nodes to your cluster do the following for each machine:

- SSH to the machine
- Become root (e.g. `sudo su -`)
- Run the command that was output by `kubeadm init`. For example:

```
kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha25
```

> **Note:** To specify an IPv6 tuple for `<master-ip>:<master-port>`, IPv6 address must be enclosed in square brackets, for example: `[fd00::101]:2073`.

The output should look something like:

```
[preflight] Running pre-flight checks

... (log output of join workflow) ...

Node join complete:
* Certificate signing request sent to master and response
  received.
* Kubelet informed of new secure connection details.

Run 'kubectl get nodes' on the master to see this machine join.
```

A few seconds later, you should notice this node in the output from `kubectl get nodes` when run on the master.

(Optional) Controlling your cluster from machines other than the master

In order to get a kubectl on some other computer (e.g. laptop) to talk to your cluster, you need to copy the administrator kubeconfig file from your master to your workstation like this:

```
scp root@<master ip>:/etc/kubernetes/admin.conf .
kubectl --kubeconfig ./admin.conf get nodes
```

> **Note:** The example above assumes SSH access is enabled for root. If that is not the case, you can copy the `admin.conf` file to be accessible by some other user and `scp` using that other user instead.
>
> The `admin.conf` file gives the user *superuser* privileges over the cluster. This file should be used sparingly. For normal users, it's recommended to generate an unique credential to which you whitelist privileges. You can do this with the `kubeadm alpha phase kubeconfig user --client-name <CN>` command. That command will print out a KubeConfig file to STDOUT which you should save to a file and distribute to your user. After that, whitelist privileges by using `kubectl create (cluster)rolebinding`.

(Optional) Proxying API Server to localhost

If you want to connect to the API Server from outside the cluster you can use `kubectl proxy`:

```
scp root@<master ip>:/etc/kubernetes/admin.conf .
kubectl --kubeconfig ./admin.conf proxy
```

You can now access the API Server locally at `http://localhost:8001/api/v1`

Tear down

To undo what kubeadm did, you should first drain the node and make sure that the node is empty before shutting it down.

Talking to the master with the appropriate credentials, run:

```
kubectl drain <node name> --delete-local-data --force --ignore-daemonsets
kubectl delete node <node name>
```

Then, on the node being removed, reset all kubeadm installed state:

```
kubeadm reset
```

If you wish to start over simply run `kubeadm init` or `kubeadm join` with the appropriate arguments.

More options and information about the `kubeadm reset command`.

Maintaining a cluster

Instructions for maintaining kubeadm clusters (e.g. upgrades,downgrades, etc.) can be found here.

Explore other add-ons

See the list of add-ons to explore other add-ons, including tools for logging, monitoring, network policy, visualization & control of your Kubernetes cluster.

What's next

- Verify that your cluster is running properly with Sonobuoy
- Learn about kubeadm's advanced usage in the kubeadm reference documentation
- Learn more about Kubernetes concepts and `kubectl`.
- Configure log rotation. You can use **logrotate** for that. When using Docker, you can specify log rotation options for Docker daemon, for example `--log-driver=json-file --log-opt=max-size=10m --log-opt=max-file=5`. See Configure and troubleshoot the Docker daemon for more details.

Feedback

- For bugs, visit kubeadm Github issue tracker
- For support, visit kubeadm Slack Channel: #kubeadm
- General SIG Cluster Lifecycle Development Slack Channel: #sig-cluster-lifecycle
- SIG Cluster Lifecycle SIG information
- SIG Cluster Lifecycle Mailing List: kubernetes-sig-cluster-lifecycle

Version skew policy

The kubeadm CLI tool of version vX.Y may deploy clusters with a control plane of version vX.Y or vX.(Y-1). kubeadm CLI vX.Y can also upgrade an existing kubeadm-created cluster of version vX.(Y-1).

Due to that we can't see into the future, kubeadm CLI vX.Y may or may not be able to deploy vX.(Y+1) clusters.

Example: kubeadm v1.8 can deploy both v1.7 and v1.8 clusters and upgrade v1.7 kubeadm-created clusters to v1.8.

Please also check our installation guide for more information on the version skew between kubelets and the control plane.

kubeadm works on multiple platforms

kubeadm deb/rpm packages and binaries are built for amd64, arm (32-bit), arm64, ppc64le, and s390x following the multi-platform proposal.

Only some of the network providers offer solutions for all platforms. Please consult the list of network providers above or the documentation from each provider to figure out whether the provider supports your chosen platform.

Limitations

Please note: kubeadm is a work in progress and these limitations will be addressed in due course.

1. The cluster created here has a single master, with a single etcd database running on it. This means that if the master fails, your cluster may lose data and may need to be recreated from scratch. Adding HA support (multiple etcd servers, multiple API servers, etc) to kubeadm is still a work-in-progress.

Workaround: regularly back up etcd. The etcd data directory configured by kubeadm is at `/var/lib/etcd` on the master.

Troubleshooting

If you are running into difficulties with kubeadm, please consult our troubleshooting docs.

Create an Issue Edit this Page

Edit This Page

Troubleshooting kubeadm

-
  -
    -
      - `ebtables` or some similar executable not found during installation
      - kubeadm blocks waiting for control plane during installation
      - kubeadm blocks when removing managed containers
      - Pods in `RunContainerError`, `CrashLoopBackOff` or `Error` state
      - `coredns` (or `kube-dns`) is stuck in the `Pending` state
      - `HostPort` services do not work
      - Pods are not accessible via their Service IP
      - TLS certificate errors
      - Default NIC When using flannel as the pod network in Vagrant
      - Non-public IP used for containers
      - Services with externalTrafficPolicy=Local are not reachable

`ebtables` or some similar executable not found during installation

If you see the following warnings while running `kubeadm init`

```
[preflight] WARNING: ebtables not found in system path
[preflight] WARNING: ethtool not found in system path
```

Then you may be missing `ebtables`, `ethtool` or a similar executable on your node. You can install them with the following commands:

- For Ubuntu/Debian users, run `apt install ebtables ethtool`.
- For CentOS/Fedora users, run `yum install ebtables ethtool`.

kubeadm blocks waiting for control plane during installation

If you notice that `kubeadm init` hangs after printing out the following line:

```
[apiclient] Created API client, waiting for the control plane to become ready
```

This may be caused by a number of problems. The most common are:

- network connection problems. Check that your machine has full network connectivity before continuing.
- the default cgroup driver configuration for the kubelet differs from that used by Docker. Check the system log file (e.g. `/var/log/message`) or examine the output from `journalctl -u kubelet`. If you see something like the following:

```
error: failed to run Kubelet: failed to create kubelet:
misconfiguration: kubelet cgroup driver: "systemd" is different from docker cgroup driver:
```

There are two common ways to fix the cgroup driver problem:

1. Install docker again following instructions here.
2. Change the kubelet config to match the Docker cgroup driver manually, you can refer to Configure cgroup driver used by kubelet on Master Node for detailed instructions.

- control plane Docker containers are crashlooping or hanging. You can check this by running `docker ps` and investigating each container by running `docker logs`.

kubeadm blocks when removing managed containers

The following could happen if Docker halts and does not remove any Kubernetes-managed containers:

```
sudo kubeadm reset
[preflight] Running pre-flight checks
[reset] Stopping the kubelet service
[reset] Unmounting mounted directories in "/var/lib/kubelet"
[reset] Removing kubernetes-managed containers
(block)
```

A possible solution is to restart the Docker service and then re-run `kubeadm reset`:

```
sudo systemctl restart docker.service
sudo kubeadm reset
```

Inspecting the logs for docker may also be useful:

```
journalctl -ul docker
```

Pods in `RunContainerError`, `CrashLoopBackOff` or `Error` state

Right after `kubeadm init` there should not be any pods in these states.

- If there are pods in one of these states *right after* `kubeadm init`, please open an issue in the kubeadm repo. `coredns` (or `kube-dns`) should be in the `Pending` state until you have deployed the network solution.

- If you see Pods in the `RunContainerError`, `CrashLoopBackOff` or `Error` state after deploying the network solution and nothing happens to `coredns` (or `kube-dns`), it's very likely that the Pod Network solution that you installed is somehow broken. You might have to grant it more RBAC privileges or use a newer version. Please file an issue in the Pod Network providers' issue tracker and get the issue triaged there.

`coredns` (or `kube-dns`) is stuck in the `Pending` state

kubeadm does not install a pod network solution by default. You have to install a Pod Network before `coredns` (or `kube-dns`) pods will be scheduled.

`HostPort` services do not work

The `HostPort` and `HostIP` functionality is available depending on your Pod Network provider. Please contact the author of the Pod Network solution to find out whether `HostPort` and `HostIP` functionality are available.

Calico, Canal, and Flannel CNI providers are verified to support HostPort.

For more information, see the CNI portmap documentation.

If your network provider does not support the portmap CNI plugin, you may need to use the NodePort feature of services or use `HostNetwork=true`.

Pods are not accessible via their Service IP

- Many network add-ons do not yet enable hairpin mode which allows pods to access themselves via their Service IP. This is an issue related to CNI. Please contact the network add-on provider to get the latest status of their support for hairpin mode.
- If you are using VirtualBox (directly or via Vagrant), you will need to ensure that `hostname -i` returns a routable IP address. By default the first interface is connected to a non-routable host-only network. A work around is to modify `/etc/hosts`, see this Vagrantfile for an example.

TLS certificate errors

The following error indicates a possible certificate mismatch.

```
# kubectl get pods
Unable to connect to the server: x509: certificate signed by unknown authority (possibly bec
```

- Verify that the `$HOME/.kube/config` file contains a valid certificate, and regenerate a certificate if necessary. The certificates in a kubeconfig file are base64 encoded. The `base64 -d` command can be used to decode the

  certificate and `openssl x509 -text -noout` can be used for viewing the certificate information.
- Another workaround is to overwrite the existing `kubeconfig` for the "admin" user:

```
mv  $HOME/.kube $HOME/.kube.bak
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Default NIC When using flannel as the pod network in Vagrant

The following error might indicate that something was wrong in the pod network:

```
Error from server (NotFound): the server could not find the requested resource
```

- If you're using flannel as the pod network inside Vagrant, then you will have to specify the default interface name for flannel.

Vagrant typically assigns two interfaces to all VMs. The first, for which all hosts are assigned the IP address `10.0.2.15`, is for external traffic that gets NATed.

This may lead to problems with flannel, which defaults to the first interface on a host. This leads to all hosts thinking they have the same public IP address. To prevent this, pass the `--iface eth1` flag to flannel so that the second interface is chosen.

Non-public IP used for containers

In some situations `kubectl logs` and `kubectl run` commands may return with the following errors in an otherwise functional cluster:

```
Error from server: Get https://10.19.0.41:10250/containerLogs/default/mysql-ddc65b868-glc5m/
```

- This may be due to Kubernetes using an IP that can not communicate with other IPs on the seemingly same subnet, possibly by policy of the machine provider.
- Digital Ocean assigns a public IP to `eth0` as well as a private one to be used internally as anchor for their floating IP feature, yet `kubelet` will pick the latter as the node's `InternalIP` instead of the public one.

Use `ip addr show` to check for this scenario instead of `ifconfig` because `ifconfig` will not display the offending alias IP address. Alternatively an API endpoint specific to Digital Ocean allows to query for the anchor IP from the droplet:

```
curl http://169.254.169.254/metadata/v1/interfaces/public/0/anchor_ipv4/address
```

The workaround is to tell `kubelet` which IP to use using `--node-ip`. When using Digital Ocean, it can be the public one (assigned to `eth0`) or the private one (assigned to `eth1`) should you want to use the optional private network. The KubeletExtraArgs section of the MasterConfiguration file can be used for this.

Then restart `kubelet`:

```
systemctl daemon-reload
systemctl restart kubelet
```

Services with externalTrafficPolicy=Local are not reachable

On nodes where the hostname for the kubelet is overridden using the `--hostname-override` option, kube-proxy will default to treating 127.0.0.1 as the node IP, which results in rejecting connections for Services configured for `externalTrafficPolicy=Local`. This situation can be verified by checking the output of `kubectl -n kube-system logs <kube-proxy pod name>`:

```
W0507 22:33:10.372369       1 server.go:586] Failed to retrieve node info: nodes "ip-10-0-23
W0507 22:33:10.372474       1 proxier.go:463] invalid nodeIP, initializing kube-proxy with 1
```

A workaround for this is to modify the kube-proxy DaemonSet in the following way:

```
kubectl -n kube-system patch --type json daemonset kube-proxy -p "$(cat <<'EOF'
[
    {
        "op": "add",
        "path": "/spec/template/spec/containers/0/env",
        "value": [
            {
                "name": "NODE_NAME",
                "valueFrom": {
                    "fieldRef": {
                        "apiVersion": "v1",
                        "fieldPath": "spec.nodeName"
                    }
                }
            }
        ]
    },
    {
        "op": "add",
        "path": "/spec/template/spec/containers/0/command/-",
        "value": "--hostname-override=${NODE_NAME}"
    }
]
EOF
)"
```

Create an Issue Edit this Page

Edit This Page

Running Kubernetes on AWS EC2

Supported Production Grade Tools

- conjure-up is an open-source installer for Kubernetes that creates Kubernetes clusters with native AWS integrations on Ubuntu.
- Kubernetes Operations - Production Grade K8s Installation, Upgrades, and Management. Supports running Debian, Ubuntu, CentOS, and RHEL in AWS.
- CoreOS Tectonic includes the open-source Tectonic Installer that creates Kubernetes clusters with Container Linux nodes on AWS.
- CoreOS originated and the Kubernetes Incubator maintains a CLI tool, `kube-aws`, that creates and manages Kubernetes clusters with Container Linux nodes, using AWS tools: EC2, CloudFormation and Autoscaling.

Getting started with your cluster

Command line administration tool: `kubectl`

The cluster startup script will leave you with a `kubernetes` directory on your workstation. Alternately, you can download the latest Kubernetes release from this page.

Next, add the appropriate binary folder to your `PATH` to access kubectl:

```
# OS X
export PATH=<path/to/kubernetes-directory>/platforms/darwin/amd64:$PATH

# Linux
export PATH=<path/to/kubernetes-directory>/platforms/linux/amd64:$PATH
```

An up-to-date documentation page for this tool is available here: kubectl manual

By default, `kubectl` will use the `kubeconfig` file generated during the cluster startup for authenticating against the API. For more information, please read kubeconfig files

Examples

See a simple nginx example to try out your new cluster.

The "Guestbook" application is another popular example to get started with Kubernetes: guestbook example

For more complete applications, please look in the examples directory

Scaling the cluster

Adding and removing nodes through `kubectl` is not supported. You can still scale the amount of nodes manually through adjustments of the 'Desired' and 'Max' properties within the Auto Scaling Group, which was created during the installation.

Tearing down the cluster

Make sure the environment variables you used to provision your cluster are still exported, then call the following script inside the `kubernetes` directory:

```
cluster/kube-down.sh
```

Support Level

IaaS Provider	Config. Mgmt	OS	Networking	Docs	Conforms	Support Level
AWS	kops	Debian	k8s (VPC)	docs		Community (@justinsb
AWS	CoreOS	CoreOS	flannel	docs		Community
AWS	Juju	Ubuntu	flannel, calico, canal	docs	100%	Commercial, Communi

For support level information on all solutions, see the Table of solutions chart.

Further reading

Please see the Kubernetes docs for more details on administering and using a Kubernetes cluster.

Create an Issue Edit this Page

Edit This Page

Running Kubernetes on AWS EC2

Supported Production Grade Tools

- conjure-up is an open-source installer for Kubernetes that creates Kubernetes clusters with native AWS integrations on Ubuntu.
- Kubernetes Operations - Production Grade K8s Installation, Upgrades, and Management. Supports running Debian, Ubuntu, CentOS, and RHEL in AWS.
- CoreOS Tectonic includes the open-source Tectonic Installer that creates Kubernetes clusters with Container Linux nodes on AWS.
- CoreOS originated and the Kubernetes Incubator maintains a CLI tool, `kube-aws`, that creates and manages Kubernetes clusters with Container Linux nodes, using AWS tools: EC2, CloudFormation and Autoscaling.

Getting started with your cluster

Command line administration tool: `kubectl`

The cluster startup script will leave you with a `kubernetes` directory on your workstation. Alternately, you can download the latest Kubernetes release from this page.

Next, add the appropriate binary folder to your `PATH` to access kubectl:

```
# OS X
export PATH=<path/to/kubernetes-directory>/platforms/darwin/amd64:$PATH

# Linux
export PATH=<path/to/kubernetes-directory>/platforms/linux/amd64:$PATH
```

An up-to-date documentation page for this tool is available here: kubectl manual

By default, `kubectl` will use the `kubeconfig` file generated during the cluster startup for authenticating against the API. For more information, please read kubeconfig files

Examples

See a simple nginx example to try out your new cluster.

The "Guestbook" application is another popular example to get started with Kubernetes: guestbook example

For more complete applications, please look in the examples directory

Scaling the cluster

Adding and removing nodes through `kubectl` is not supported. You can still scale the amount of nodes manually through adjustments of the 'Desired' and 'Max' properties within the Auto Scaling Group, which was created during the installation.

Tearing down the cluster

Make sure the environment variables you used to provision your cluster are still exported, then call the following script inside the `kubernetes` directory:

```
cluster/kube-down.sh
```

Support Level

IaaS Provider	Config. Mgmt	OS	Networking	Docs	Conforms	Support Level
AWS	kops	Debian	k8s (VPC)	docs		Community (@justinsb
AWS	CoreOS	CoreOS	flannel	docs		Community
AWS	Juju	Ubuntu	flannel, calico, canal	docs	100%	Commercial, Communi

For support level information on all solutions, see the Table of solutions chart.

Further reading

Please see the Kubernetes docs for more details on administering and using a Kubernetes cluster.

Create an Issue Edit this Page

Edit This Page

Running Kubernetes on Alibaba Cloud

- - Alibaba Cloud Container Service
  - Custom Deployments

Alibaba Cloud Container Service

The Alibaba Cloud Container Service lets you run and manage Docker applications on a cluster of Alibaba Cloud ECS instances. It supports the popular open source container orchestrators: Docker Swarm and Kubernetes.

To simplify cluster deployment and management, use Kubernetes Support for Alibaba Cloud Container Service. You can get started quickly by following the Kubernetes walk-through, and there are some tutorials for Kubernetes Support on Alibaba Cloud in Chinese.

To use custom binaries or open source Kubernetes, follow the instructions below.

Custom Deployments

The source code for Kubernetes with Alibaba Cloud provider implementation is open source and available on GitHub.

For more information, see "Quick deployment of Kubernetes - VPC environment on Alibaba Cloud" in English and Chinese.

Create an Issue Edit this Page

Edit This Page

Running Kubernetes on Azure

- - Azure Container Service
  - Custom Deployments: ACS-Engine
  - CoreOS Tectonic for Azure

Azure Container Service

The Azure Container Service offers simple deployments of one of three open source orchestrators: DC/OS, Swarm, and Kubernetes clusters.

For an example of deploying a Kubernetes cluster onto Azure via the Azure Container Service:

Microsoft Azure Container Service - Kubernetes Walkthrough

Custom Deployments: ACS-Engine

The core of the Azure Container Service is **open source** and available on GitHub for the community to use and contribute to: **ACS-Engine**.

ACS-Engine is a good choice if you need to make customizations to the deployment beyond what the Azure Container Service officially supports. These customizations include deploying into existing virtual networks, utilizing multiple agent pools, and more. Some community contributions to ACS-Engine may even become features of the Azure Container Service.

The input to ACS-Engine is similar to the ARM template syntax used to deploy a cluster directly with the Azure Container Service. The resulting output is an Azure Resource Manager Template that can then be checked into source control and can then be used to deploy Kubernetes clusters into Azure.

You can get started quickly by following the **ACS-Engine Kubernetes Walkthrough**.

CoreOS Tectonic for Azure

The CoreOS Tectonic Installer for Azure is **open source** and available on GitHub for the community to use and contribute to: **Tectonic Installer**.

Tectonic Installer is a good choice when you need to make cluster customizations as it is built on Hashicorp's Terraform Azure Resource Manager (ARM) provider. This enables users to customize or integrate using familiar Terraform tooling.

You can get started using the Tectonic Installer for Azure Guide.

Create an Issue Edit this Page

Edit This Page

Running Kubernetes on CenturyLink Cloud

{: toc}

These scripts handle the creation, deletion and expansion of Kubernetes clusters on CenturyLink Cloud.

You can accomplish all these tasks with a single command. We have made the Ansible playbooks used to perform these tasks available here.

Find Help

If you run into any problems or want help with anything, we are here to help. Reach out to use via any of the following ways:

- Submit a github issue
- Send an email to Kubernetes AT ctl DOT io
- Visit http://info.ctl.io/kubernetes

Clusters of VMs or Physical Servers, your choice.

- We support Kubernetes clusters on both Virtual Machines or Physical Servers. If you want to use physical servers for the worker nodes (minions), simple use the –minion_type=bareMetal flag.
- For more information on physical servers, visit: https://www.ctl.io/bare-metal/
- Physical serves are only available in the VA1 and GB3 data centers.
- VMs are available in all 13 of our public cloud locations

Requirements

The requirements to run this script are:

- A linux administrative host (tested on ubuntu and OSX)
- python 2 (tested on 2.7.11)
  - pip (installed with python as of 2.7.9)
- git
- A CenturyLink Cloud account with rights to create new hosts
- An active VPN connection to the CenturyLink Cloud from your linux host

Script Installation

After you have all the requirements met, please follow these instructions to install this script.

1) Clone this repository and cd into it.

```
git clone https://github.com/CenturyLinkCloud/adm-kubernetes-on-clc
```

2) Install all requirements, including

- Ansible
- CenturyLink Cloud SDK
- Ansible Modules

```
sudo pip install -r ansible/requirements.txt
```

3) Create the credentials file from the template and use it to set your ENV variables

```
cp ansible/credentials.sh.template ansible/credentials.sh
vi ansible/credentials.sh
source ansible/credentials.sh
```

4) Grant your machine access to the CenturyLink Cloud network by using a VM inside the network or configuring a VPN connection to the CenturyLink Cloud network.

Script Installation Example: Ubuntu 14 Walkthrough

If you use an ubuntu 14, for your convenience we have provided a step by step guide to install the requirements and install the script.

```
# system
apt-get update
apt-get install -y git python python-crypto
curl -O https://bootstrap.pypa.io/get-pip.py
python get-pip.py

# installing this repository
mkdir -p ~home/k8s-on-clc
cd ~home/k8s-on-clc
git clone https://github.com/CenturyLinkCloud/adm-kubernetes-on-clc.git
cd adm-kubernetes-on-clc/
pip install -r requirements.txt

# getting started
cd ansible
cp credentials.sh.template credentials.sh; vi credentials.sh
source credentials.sh
```

Cluster Creation

To create a new Kubernetes cluster, simply run the `kube-up.sh` script. A complete list of script options and some examples are listed below.

```
CLC_CLUSTER_NAME=[name of kubernetes cluster]
cd ./adm-kubernetes-on-clc
bash kube-up.sh -c="$CLC_CLUSTER_NAME"
```

It takes about 15 minutes to create the cluster. Once the script completes, it will output some commands that will help you setup kubectl on your machine to point to the new cluster.

When the cluster creation is complete, the configuration files for it are stored locally on your administrative host, in the following directory

```
> CLC_CLUSTER_HOME=$HOME/.clc_kube/$CLC_CLUSTER_NAME/
```

Cluster Creation: Script Options

```
Usage: kube-up.sh [OPTIONS]
Create servers in the CenturyLinkCloud environment and initialize a Kubernetes cluster
Environment variables CLC_V2_API_USERNAME and CLC_V2_API_PASSWD must be set in
order to access the CenturyLinkCloud API
```

```
All options (both short and long form) require arguments, and must include "="
between option name and option value.

     -h (--help)                   display this help and exit
     -c= (--clc_cluster_name=)     set the name of the cluster, as used in CLC group names
     -t= (--minion_type=)          standard -> VM (default), bareMetal -> physical]
     -d= (--datacenter=)           VA1 (default)
     -m= (--minion_count=)         number of kubernetes minion nodes
     -mem= (--vm_memory=)          number of GB ram for each minion
     -cpu= (--vm_cpu=)             number of virtual cps for each minion node
     -phyid= (--server_conf_id=)   physical server configuration id, one of
                                      physical_server_20_core_conf_id
                                      physical_server_12_core_conf_id
                                      physical_server_4_core_conf_id (default)
     -etcd_separate_cluster=yes    create a separate cluster of three etcd nodes,
                                   otherwise run etcd on the master node
```

Cluster Expansion

To expand an existing Kubernetes cluster, run the `add-kube-node.sh` script. A complete list of script options and some examples are listed below. This script must be run from the same host that created the cluster (or a host that has the cluster artifact files stored in `~/.clc_kube/$cluster_name`).

```
cd ./adm-kubernetes-on-clc
bash add-kube-node.sh -c="name_of_kubernetes_cluster" -m=2
```

Cluster Expansion: Script Options

```
Usage: add-kube-node.sh [OPTIONS]
Create servers in the CenturyLinkCloud environment and add to an
existing CLC kubernetes cluster

Environment variables CLC_V2_API_USERNAME and CLC_V2_API_PASSWD must be set in
order to access the CenturyLinkCloud API

     -h (--help)                   display this help and exit
     -c= (--clc_cluster_name=)     set the name of the cluster, as used in CLC group names
     -m= (--minion_count=)         number of kubernetes minion nodes to add
```

Cluster Deletion

There are two ways to delete an existing cluster:

1) Use our python script:

```
python delete_cluster.py --cluster=clc_cluster_name --datacenter=DC1
```

2) Use the CenturyLink Cloud UI. To delete a cluster, log into the CenturyLink Cloud control portal and delete the parent server group that contains the Kubernetes Cluster. We hope to add a scripted option to do this soon.

Examples

Create a cluster with name of k8s_1, 1 master node and 3 worker minions (on physical machines), in VA1

```
bash kube-up.sh --clc_cluster_name=k8s_1 --minion_type=bareMetal --minion_count=3 --datacent
```

Create a cluster with name of k8s_2, an ha etcd cluster on 3 VMs and 6 worker minions (on VMs), in VA1

```
bash kube-up.sh --clc_cluster_name=k8s_2 --minion_type=standard --minion_count=6 --datacente
```

Create a cluster with name of k8s_3, 1 master node, and 10 worker minions (on VMs) with higher mem/cpu, in UC1:

```
bash kube-up.sh --clc_cluster_name=k8s_3 --minion_type=standard --minion_count=10 --datacent
```

Cluster Features and Architecture

We configure the Kubernetes cluster with the following features:

- KubeDNS: DNS resolution and service discovery
- Heapster/InfluxDB: For metric collection. Needed for Grafana and auto-scaling.
- Grafana: Kubernetes/Docker metric dashboard
- KubeUI: Simple web interface to view Kubernetes state
- Kube Dashboard: New web interface to interact with your cluster

We use the following to create the Kubernetes cluster:

- Kubernetes 1.1.7
- Ubuntu 14.04
- Flannel 0.5.4
- Docker 1.9.1-0~trusty
- Etcd 2.2.2

Optional add-ons

- Logging: We offer an integrated centralized logging ELK platform so that all Kubernetes and docker logs get sent to the ELK stack. To install the ELK stack and configure Kubernetes to send logs to it, follow the log aggregation documentation. Note: We don't install this by default as the footprint isn't trivial.

Cluster management

The most widely used tool for managing a Kubernetes cluster is the command-line utility `kubectl`. If you do not already have a copy of this binary on your administrative machine, you may run the script `install_kubectl.sh` which will download it and install it in `/usr/bin/local`.

The script requires that the environment variable `CLC_CLUSTER_NAME` be defined

`install_kubectl.sh` also writes a configuration file which will embed the necessary authentication certificates for the particular cluster. The configuration file is written to the `${CLC_CLUSTER_HOME}/kube` directory

```
export KUBECONFIG=${CLC_CLUSTER_HOME}/kube/config
kubectl version
kubectl cluster-info
```

Accessing the cluster programmatically

It's possible to use the locally stored client certificates to access the apiserver. For example, you may want to use any of the Kubernetes API client libraries to program against your Kubernetes cluster in the programming language of your choice.

To demonstrate how to use these locally stored certificates, we provide the following example of using `curl` to communicate to the master apiserver via https:

```
curl \
   --cacert ${CLC_CLUSTER_HOME}/pki/ca.crt  \
   --key ${CLC_CLUSTER_HOME}/pki/kubecfg.key \
   --cert ${CLC_CLUSTER_HOME}/pki/kubecfg.crt  https://${MASTER_IP}:6443
```

But please note, this *does not* work out of the box with the `curl` binary distributed with OSX.

Accessing the cluster with a browser

We install the kubernetes dashboard. When you create a cluster, the script should output URLs for these interfaces like this:

kubernetes-dashboard is running at `https://${MASTER_IP}:6443/api/v1/namespaces/kube-system/servic`

Note on Authentication to the UIs: The cluster is set up to use basic authentication for the user *admin*. Hitting the url at `https://${MASTER_IP}:6443` will require accepting the self-signed certificate from the apiserver, and then presenting the admin password written to file at:

```
> _${CLC_CLUSTER_HOME}/kube/admin_password.txt_
```

Configuration files

Various configuration files are written into the home directory *CLC_CLUSTER_HOME* under `.clc_kube/${CLC_CLUSTER_NAME}` in several subdirectories. You can use these files to access the cluster from machines other than where you created the cluster from.

- `config/`: Ansible variable files containing parameters describing the master and minion hosts
- `hosts/`: hosts files listing access information for the ansible playbooks
- `kube/`: `kubectl` configuration files, and the basic-authentication password for admin access to the Kubernetes API
- `pki/`: public key infrastructure files enabling TLS communication in the cluster
- `ssh/`: SSH keys for root access to the hosts

`kubectl` usage examples

There are a great many features of *kubectl*. Here are a few examples

List existing nodes, pods, services and more, in all namespaces, or in just one:

```
kubectl get nodes
kubectl get --all-namespaces services
kubectl get --namespace=kube-system replicationcontrollers
```

The Kubernetes API server exposes services on web URLs, which are protected by requiring client certificates. If you run a kubectl proxy locally, `kubectl` will provide the necessary certificates and serve locally over http.

```
kubectl proxy -p 8001
```

Then, you can access urls like `http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/kubern` without the need for client certificates in your browser.

What Kubernetes features do not work on CenturyLink Cloud

These are the known items that don't work on CenturyLink cloud but do work on other cloud providers:

- At this time, there is no support services of the type LoadBalancer. We are actively working on this and hope to publish the changes sometime around April 2016.
- At this time, there is no support for persistent storage volumes provided by CenturyLink Cloud. However, customers can bring their own persistent storage offering. We ourselves use Gluster.

Ansible Files

If you want more information about our Ansible files, please read this file

Further reading

Please see the Kubernetes docs for more details on administering and using a Kubernetes cluster.

Create an Issue Edit this Page

Edit This Page

Running Kubernetes on Google Compute Engine

The example below creates a Kubernetes cluster with 4 worker node Virtual Machines and a master Virtual Machine (i.e. 5 VMs in your cluster). This cluster is set up and controlled from your workstation (or wherever you find convenient).

- Support Level
- Further reading

Before you start

If you want a simplified getting started experience and GUI for managing clusters, please consider trying Google Kubernetes Engine for hosted cluster installation and management.

For an easy way to experiment with the Kubernetes development environment, click the button below to open a Google Cloud Shell with an auto-cloned copy of the Kubernetes source repo.

If you want to use custom binaries or pure open source Kubernetes, please continue with the instructions below.

Prerequisites

1. You need a Google Cloud Platform account with billing enabled. Visit the Google Developers Console for more details.
2. Install `gcloud` as necessary. `gcloud` can be installed as a part of the Google Cloud SDK.
3. Enable the Compute Engine Instance Group Manager API in the Google Cloud developers console.
4. Make sure that gcloud is set to use the Google Cloud Platform project you want. You can check the current project using `gcloud config list project` and change it via `gcloud config set project <project-id>`.
5. Make sure you have credentials for GCloud by running `gcloud auth login`.
6. (Optional) In order to make API calls against GCE, you must also run `gcloud auth application-default login`.
7. Make sure you can start up a GCE VM from the command line. At least make sure you can do the Create an instance part of the GCE Quickstart.
8. Make sure you can SSH into the VM without interactive prompts. See the Log in to the instance part of the GCE Quickstart.

Starting a cluster

You can install a client and start a cluster with either one of these commands (we list both in case only one is installed on your machine):

```
curl -sS https://get.k8s.io | bash
```

or

```
wget -q -O - https://get.k8s.io | bash
```

Once this command completes, you will have a master VM and four worker VMs, running as a Kubernetes cluster.

By default, some containers will already be running on your cluster. Containers like `fluentd` provide logging, while `heapster` provides monitoring services.

The script run by the commands above creates a cluster with the name/prefix "kubernetes". It defines one specific cluster config, so you can't run it more than once.

Alternately, you can download and install the latest Kubernetes release from this page, then run the `<kubernetes>/cluster/kube-up.sh` script to start the cluster:

```
cd kubernetes
cluster/kube-up.sh
```

If you want more than one cluster running in your project, want to use a different name, or want a different number of worker nodes, see the `<kubernetes>/cluster/gce/config-default.sh` file for more fine-grained configuration before you start up your cluster.

If you run into trouble, please see the section on troubleshooting, post to the kubernetes-users group, or come ask questions on Slack.

The next few steps will show you:

1. How to set up the command line client on your workstation to manage the cluster
2. Examples of how to use the cluster
3. How to delete the cluster
4. How to start clusters with non-default options (like larger clusters)

Installing the Kubernetes command line tools on your workstation

The cluster startup script will leave you with a running cluster and a `kubernetes` directory on your workstation.

The kubectl tool controls the Kubernetes cluster manager. It lets you inspect your cluster resources, create, delete, and update components, and much more. You will use it to look at your new cluster and bring up example apps.

You can use `gcloud` to install the `kubectl` command-line tool on your workstation:

```
gcloud components install kubectl
```

Note: The kubectl version bundled with `gcloud` may be older than the one downloaded by the get.k8s.io install script. See Installing kubectl document to see how you can set up the latest `kubectl` on your workstation.

Getting started with your cluster

Inspect your cluster

Once `kubectl` is in your path, you can use it to look at your cluster. E.g., running:

```
$ kubectl get --all-namespaces services
```

should show a set of services that look something like this:

```
NAMESPACE     NAME                  CLUSTER_IP       EXTERNAL_IP       PORT(S)        AGE
default       kubernetes            10.0.0.1         <none>            443/TCP        1d
kube-system   kube-dns              10.0.0.2         <none>            53/TCP,53/UDP  1d
kube-system   kube-ui               10.0.0.3         <none>            80/TCP         1d
...
```

Similarly, you can take a look at the set of pods that were created during cluster startup. You can do this via the

```
$ kubectl get --all-namespaces pods
```

command.

You'll see a list of pods that looks something like this (the name specifics will be different):

```
NAMESPACE     NAME                                         READY     STATUS    RESTARTS
kube-system   fluentd-cloud-logging-kubernetes-minion-63uo   1/1       Running   0
kube-system   fluentd-cloud-logging-kubernetes-minion-c1n9   1/1       Running   0
kube-system   fluentd-cloud-logging-kubernetes-minion-c4og   1/1       Running   0
kube-system   fluentd-cloud-logging-kubernetes-minion-ngua   1/1       Running   0
kube-system   kube-dns-v5-7ztia                            3/3       Running   0
kube-system   kube-ui-v1-curt1                             1/1       Running   0
kube-system   monitoring-heapster-v5-ex4u3                 1/1       Running   1
kube-system   monitoring-influx-grafana-v1-piled           2/2       Running   0
```

Some of the pods may take a few seconds to start up (during this time they'll show `Pending`), but check that they all show as `Running` after a short period.

Run some examples

Then, see a simple nginx example to try out your new cluster.

For more complete applications, please look in the examples directory. The guestbook example is a good "getting started" walkthrough.

Tearing down the cluster

To remove/delete/teardown the cluster, use the `kube-down.sh` script.

```
cd kubernetes
cluster/kube-down.sh
```

Likewise, the `kube-up.sh` in the same directory will bring it back up. You do not need to rerun the `curl` or `wget` command: everything needed to setup the Kubernetes cluster is now on your workstation.

Customizing

The script above relies on Google Storage to stage the Kubernetes release. It then will start (by default) a single master VM along with 4 worker VMs. You can tweak some of these parameters by editing `kubernetes/cluster/gce/config-default.sh` You can view a transcript of a successful cluster creation here.

Troubleshooting

Project settings

You need to have the Google Cloud Storage API, and the Google Cloud Storage JSON API enabled. It is activated by default for new projects. Otherwise, it can be done in the Google Cloud Console. See the Google Cloud Storage JSON API Overview for more details.

Also ensure that– as listed in the Prerequisites section– you've enabled the `Compute Engine Instance Group Manager API`, and can start up a GCE VM from the command line as in the GCE Quickstart instructions.

Cluster initialization hang

If the Kubernetes startup script hangs waiting for the API to be reachable, you can troubleshoot by SSHing into the master and node VMs and looking at logs such as `/var/log/startupscript.log`.

Once you fix the issue, you should run `kube-down.sh` to cleanup after the partial cluster creation, before running `kube-up.sh` to try again.

SSH

If you're having trouble SSHing into your instances, ensure the GCE firewall isn't blocking port 22 to your VMs. By default, this should work but if you have edited firewall rules or created a new non-default network, you'll need to expose it: `gcloud compute firewall-rules create default-ssh --network=<network-name> --description "SSH allowed from anywhere" --allow tcp:22`

Additionally, your GCE SSH key must either have no passcode or you need to be using `ssh-agent`.

Networking

The instances must be able to connect to each other using their private IP. The script uses the "default" network which should have a firewall rule called "default-allow-internal" which allows traffic on any port on the private IPs. If this rule is missing from the default network or if you change the network being used in `cluster/config-default.sh` create a new rule with the following field values:

- Source Ranges: `10.0.0.0/8`
- Allowed Protocols and Port: `tcp:1-65535;udp:1-65535;icmp`

Support Level

IaaS Provider	Config. Mgmt	OS	Networking	Docs	Conforms	Support Level
GCE	Saltstack	Debian	GCE	docs		Project

For support level information on all solutions, see the Table of solutions chart.

Further reading

Please see the Kubernetes docs for more details on administering and using a Kubernetes cluster.

Create an Issue Edit this Page

Edit This Page

Running Kubernetes on Multiple Clouds with Stackpoint.io

Introduction

StackPointCloud is the universal control plane for Kubernetes Anywhere. StackPointCloud allows you to deploy and manage a Kubernetes cluster to the cloud provider of your choice in 3 steps using a web-based interface.

AWS

To create a Kubernetes cluster on AWS, you will need an Access Key ID and a Secret Access Key from AWS.

Choose a Provider

Log in to stackpoint.io with a GitHub, Google, or Twitter account.

Click **+ADD A CLUSTER NOW**.

Click to select Amazon Web Services (AWS).

Configure Your Provider

Add your Access Key ID and a Secret Access Key from AWS. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

Click **SUBMIT** to submit the authorization information.

Configure Your Cluster

Choose any extra options you may want to include with your cluster, then click **SUBMIT** to create the cluster.

Running the Cluster

You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

For information on using and managing a Kubernetes cluster on AWS, consult the Kubernetes documentation.

GCE

To create a Kubernetes cluster on GCE, you will need the Service Account JSON Data from Google.

Choose a Provider

Log in to stackpoint.io with a GitHub, Google, or Twitter account.

Click **+ADD A CLUSTER NOW**.

Click to select Google Compute Engine (GCE).

Configure Your Provider

Add your Service Account JSON Data from Google. Select your default Stack-PointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

Click **SUBMIT** to submit the authorization information.

Configure Your Cluster

Choose any extra options you may want to include with your cluster, then click **SUBMIT** to create the cluster.

Running the Cluster

You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

For information on using and managing a Kubernetes cluster on GCE, consult the Kubernetes documentation.

Google Kubernetes Engine

To create a Kubernetes cluster on Google Kubernetes Engine, you will need the Service Account JSON Data from Google.

Choose a Provider

Log in to stackpoint.io with a GitHub, Google, or Twitter account.

Click **+ADD A CLUSTER NOW**.

Click to select Google Kubernetes Engine.

Configure Your Provider

Add your Service Account JSON Data from Google. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

Click **SUBMIT** to submit the authorization information.

Configure Your Cluster

Choose any extra options you may want to include with your cluster, then click **SUBMIT** to create the cluster.

Running the Cluster

You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

For information on using and managing a Kubernetes cluster on Google Kubernetes Engine, consult the official documentation.

DigitalOcean

To create a Kubernetes cluster on DigitalOcean, you will need a DigitalOcean API Token.

Choose a Provider

Log in to stackpoint.io with a GitHub, Google, or Twitter account.

Click **+ADD A CLUSTER NOW**.

Click to select DigitalOcean.

Configure Your Provider

Add your DigitalOcean API Token. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

Click **SUBMIT** to submit the authorization information.

Configure Your Cluster

Choose any extra options you may want to include with your cluster, then click **SUBMIT** to create the cluster.

Running the Cluster

You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

For information on using and managing a Kubernetes cluster on DigitalOcean, consult the official documentation.

Microsoft Azure

To create a Kubernetes cluster on Microsoft Azure, you will need an Azure Subscription ID, Username/Email, and Password.

Choose a Provider

Log in to stackpoint.io with a GitHub, Google, or Twitter account.

Click **+ADD A CLUSTER NOW**.

Click to select Microsoft Azure.

Configure Your Provider

Add your Azure Subscription ID, Username/Email, and Password. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

Click **SUBMIT** to submit the authorization information.

Configure Your Cluster

Choose any extra options you may want to include with your cluster, then click **SUBMIT** to create the cluster.

Running the Cluster

You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

For information on using and managing a Kubernetes cluster on Azure, consult the Kubernetes documentation.

Packet

To create a Kubernetes cluster on Packet, you will need a Packet API Key.

Choose a Provider

Log in to stackpoint.io with a GitHub, Google, or Twitter account.

Click **+ADD A CLUSTER NOW**.

Click to select Packet.

Configure Your Provider

Add your Packet API Key. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

Click **SUBMIT** to submit the authorization information.

Configure Your Cluster

Choose any extra options you may want to include with your cluster, then click **SUBMIT** to create the cluster.

Running the Cluster

You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

For information on using and managing a Kubernetes cluster on Packet, consult the official documentation.

Create an Issue Edit this Page

Edit This Page

Kubernetes on DCOS

- – Official Mesosphere Guide

Mesosphere provides an easy option to provision Kubernetes onto DC/OS, offering:

- Pure upstream Kubernetes

- Single-click cluster provisioning
- Highly available and secure by default
- Kubernetes running alongside fast-data platforms (e.g. Akka, Cassandra, Kafka, Spark)

Official Mesosphere Guide

The canonical source of getting started on DC/OS is located in the quickstart repo.

Create an Issue Edit this Page

Edit This Page

Cloudstack

CloudStack is a software to build public and private clouds based on hardware virtualization principles (traditional IaaS). To deploy Kubernetes on CloudStack there are several possibilities depending on the Cloud being used and what images are made available. CloudStack also has a vagrant plugin available, hence Vagrant could be used to deploy Kubernetes either using the existing shell provisioner or using new Salt based recipes.

CoreOS templates for CloudStack are built nightly. CloudStack operators need to register this template in their cloud before proceeding with these Kubernetes deployment instructions.

This guide uses a single Ansible playbook, which is completely automated and can deploy Kubernetes on a CloudStack based Cloud using CoreOS images. The playbook, creates an ssh key pair, creates a security group and associated rules and finally starts coreOS instances configured via cloud-init.

- - Prerequisites
    - Clone the playbook
    - Create a Kubernetes cluster
  - Support Level

Prerequisites

```
$ sudo apt-get install -y python-pip libssl-dev
$ sudo pip install cs
$ sudo pip install sshpubkeys
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

On CloudStack server you also have to install libselinux-python :

```
yum install libselinux-python
```

cs is a python module for the CloudStack API.

Set your CloudStack endpoint, API keys and HTTP method used.

You can define them as environment variables: `CLOUDSTACK_ENDPOINT`, `CLOUDSTACK_KEY`, `CLOUDSTACK_SECRET` and `CLOUDSTACK_METHOD`.

Or create a `~/.cloudstack.ini` file:

```
[cloudstack]
endpoint = <your cloudstack api endpoint>
key = <your api access key>
secret = <your api secret key>
method = post
```

We need to use the http POST method to pass the *large* userdata to the coreOS instances.

Clone the playbook

```
$ git clone https://github.com/apachecloudstack/k8s
$ cd kubernetes-cloudstack
```

Create a Kubernetes cluster

You simply need to run the playbook.

```
$ ansible-playbook k8s.yml
```

Some variables can be edited in the `k8s.yml` file.

```
vars:
  ssh_key: k8s
  k8s_num_nodes: 2
  k8s_security_group_name: k8s
  k8s_node_prefix: k8s2
  k8s_template: <templatename>
  k8s_instance_type: <serviceofferingname>
```

This will start a Kubernetes master node and a number of compute nodes (by default 2). The `instance_type` and `template` are specific, edit them to specify your CloudStack cloud specific template and instance type (i.e. service offering).

Check the tasks and templates in `roles/k8s` if you want to modify anything.

Once the playbook as finished, it will print out the IP of the Kubernetes master:

```
TASK: [k8s | debug msg='k8s master IP is {{ k8s_master.default_ip }}'] ********
```

SSH to it using the key that was created and using the *core* user and you can list the machines in your cluster:

```
$ ssh -i ~/.ssh/id_rsa_k8s core@<master IP>
$ fleetctl list-machines
MACHINE         IP               METADATA
a017c422...     <node #1 IP>   role=node
ad13bf84...     <master IP>       role=master
e9af8293...     <node #2 IP>   role=node
```

Support Level

IaaS Provider	Config. Mgmt	OS	Networking	Docs	Conforms	Support Level
CloudStack	Ansible	CoreOS	flannel	docs		Community (@Guiques)

For support level information on all solutions, see the Table of solutions chart.

Create an Issue Edit this Page

Edit This Page

Kubernetes on DCOS

- – Official Mesosphere Guide

Mesosphere provides an easy option to provision Kubernetes onto DC/OS, offering:

- Pure upstream Kubernetes
- Single-click cluster provisioning
- Highly available and secure by default
- Kubernetes running alongside fast-data platforms (e.g. Akka, Cassandra, Kafka, Spark)

Official Mesosphere Guide

The canonical source of getting started on DC/OS is located in the quickstart repo.

Create an Issue Edit this Page

Edit This Page

oVirt

- - What is oVirt
  - oVirt Cloud Provider Deployment
  - Using the oVirt Cloud Provider
  - oVirt Cloud Provider Screencast
  - Support Level

What is oVirt

oVirt is a virtual datacenter manager that delivers powerful management of multiple virtual machines on multiple hosts. Using KVM and libvirt, oVirt can be installed on Fedora, CentOS, or Red Hat Enterprise Linux hosts to set up and manage your virtual data center.

oVirt Cloud Provider Deployment

The oVirt cloud provider allows to easily discover and automatically add new VM instances as nodes to your Kubernetes cluster. At the moment there are no community-supported or pre-loaded VM images including Kubernetes but it is possible to import or install Project Atomic (or Fedora) in a VM to generate a template. Any other distribution that includes Kubernetes may work as well.

It is mandatory to install the ovirt-guest-agent in the guests for the VM ip address and hostname to be reported to ovirt-engine and ultimately to Kubernetes.

Once the Kubernetes template is available it is possible to start instantiating VMs that can be discovered by the cloud provider.

Using the oVirt Cloud Provider

The oVirt Cloud Provider requires access to the oVirt REST-API to gather the proper information, the required credential should be specified in the `ovirt-cloud.conf` file:

```
[connection]
uri = https://localhost:8443/ovirt-engine/api
username = admin@internal
password = admin
```

In the same file it is possible to specify (using the `filters` section) what search query to use to identify the VMs to be reported to Kubernetes:

```
[filters]
# Search query used to find nodes
vms = tag=kubernetes
```

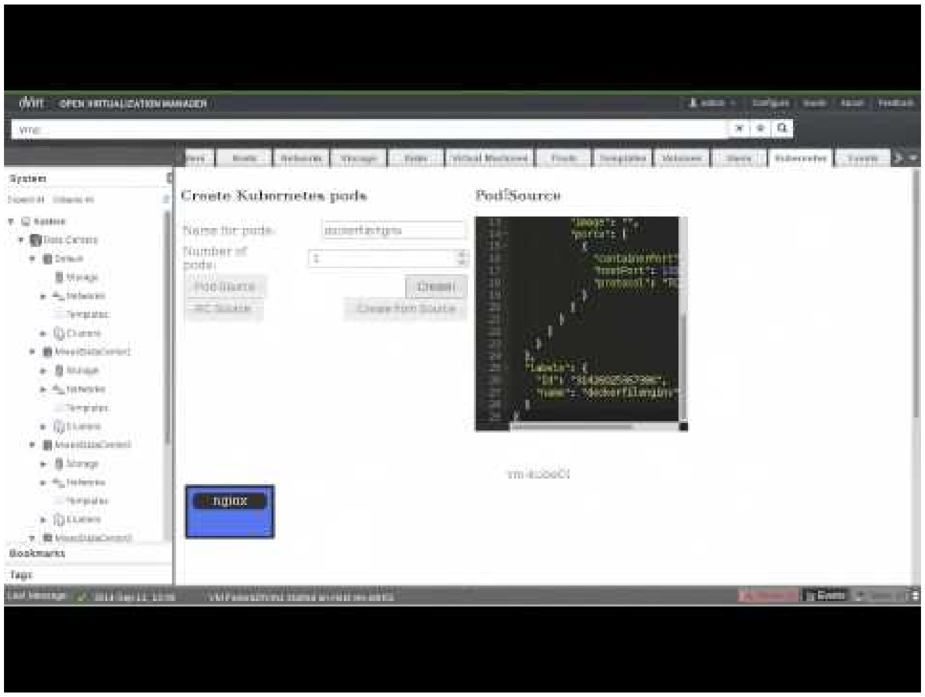
In the above example all the VMs tagged with the `kubernetes` label will be reported as nodes to Kubernetes.

The `ovirt-cloud.conf` file then must be specified in kube-controller-manager:

```
kube-controller-manager ... --cloud-provider=ovirt --cloud-config=/path/to/ovirt-cloud.conf
```

oVirt Cloud Provider Screencast

This short screencast demonstrates how the oVirt Cloud Provider can be used to dynamically add VMs to your Kubernetes cluster.



Support Level

IaaS Provider	Config. Mgmt	OS	Networking	Docs	Conforms	Support Level
oVirt				docs		Community (@simon3z)

For support level information on all solutions, see the Table of solutions chart.

Create an Issue Edit this Page

Edit This Page

Installing Kubernetes on AWS with kops

Overview

This quickstart shows you how to easily install a Kubernetes cluster on AWS. It uses a tool called `kops`.

kops is an opinionated provisioning system:

- Fully automated installation
- Uses DNS to identify clusters
- Self-healing: everything runs in Auto-Scaling Groups
- Limited OS support (Debian preferred, Ubuntu 16.04 supported, early support for CentOS & RHEL)
- High-Availability support
- Can directly provision, or generate terraform manifests

If your opinions differ from these you may prefer to build your own cluster using kubeadm as a building block. kops builds on the kubeadm work.

Creating a cluster

(1/5) Install kops

Requirements

You must have kubectl installed in order for kops to work.

Installation

Download kops from the releases page (it is also easy to build from source):

On MacOS:

```
curl -OL https://github.com/kubernetes/kops/releases/download/1.8.0/kops-darwin-amd64
chmod +x kops-darwin-amd64
mv kops-darwin-amd64 /usr/local/bin/kops
# you can also install using Homebrew
brew update && brew install kops
```

On Linux:

```
wget https://github.com/kubernetes/kops/releases/download/1.8.0/kops-linux-amd64
chmod +x kops-linux-amd64
mv kops-linux-amd64 /usr/local/bin/kops
```

(2/5) Create a route53 domain for your cluster

kops uses DNS for discovery, both inside the cluster and so that you can reach the kubernetes API server from clients.

kops has a strong opinion on the cluster name: it should be a valid DNS name. By doing so you will no longer get your clusters confused, you can share clusters with your colleagues unambiguously, and you can reach them without relying on remembering an IP address.

You can, and probably should, use subdomains to divide your clusters. As our example we will use `useast1.dev.example.com`. The API server endpoint will then be `api.useast1.dev.example.com`.

A Route53 hosted zone can serve subdomains. Your hosted zone could be `useast1.dev.example.com`, but also `dev.example.com` or even `example.com`. kops works with any of these, so typically you choose for organization reasons (e.g. you are allowed to create records under `dev.example.com`, but not under `example.com`).

Let's assume you're using `dev.example.com` as your hosted zone. You create that hosted zone using the normal process, or with a command such as `aws route53 create-hosted-zone --name dev.example.com --caller-reference 1`.

You must then set up your NS records in the parent domain, so that records in the domain will resolve. Here, you would create NS records in `example.com` for `dev`. If it is a root domain name you would configure the NS records at your domain registrar (e.g. `example.com` would need to be configured where you bought `example.com`).

This step is easy to mess up (it is the #1 cause of problems!) You can double-check that your cluster is configured correctly if you have the dig tool by running:

```
dig NS dev.example.com
```

You should see the 4 NS records that Route53 assigned your hosted zone.

(3/5) Create an S3 bucket to store your clusters state

kops lets you manage your clusters even after installation. To do this, it must keep track of the clusters that you have created, along with their configuration, the keys they are using etc. This information is stored in an S3 bucket. S3 permissions are used to control access to the bucket.

Multiple clusters can use the same S3 bucket, and you can share an S3 bucket between your colleagues that administer the same clusters - this is much easier than passing around kubecfg files. But anyone with access to the S3 bucket will have administrative access to all your clusters, so you don't want to share it beyond the operations team.

So typically you have one S3 bucket for each ops team (and often the name will correspond to the name of the hosted zone above!)

In our example, we chose `dev.example.com` as our hosted zone, so let's pick `clusters.dev.example.com` as the S3 bucket name.

- Export `AWS_PROFILE` (if you need to select a profile for the AWS CLI to work)
- Create the S3 bucket using `aws s3 mb s3://clusters.dev.example.com`
- You can `export KOPS_STATE_STORE=s3://clusters.dev.example.com` and then kops will use this location by default. We suggest putting this in your bash profile or similar.

(4/5) Build your cluster configuration

Run "kops create cluster" to create your cluster configuration:

```
kops create cluster --zones=us-east-1c useast1.dev.example.com
```

kops will create the configuration for your cluster. Note that it *only* creates the configuration, it does not actually create the cloud resources - you'll do that in the next step with a `kops update cluster`. This give you an opportunity to review the configuration or change it.

It prints commands you can use to explore further:

- List your clusters with: `kops get cluster`
- Edit this cluster with: `kops edit cluster useast1.dev.example.com`
- Edit your node instance group: `kops edit ig --name=useast1.dev.example.com nodes`
- Edit your master instance group: `kops edit ig --name=useast1.dev.example.com master-us-east-1c`

If this is your first time using kops, do spend a few minutes to try those out! An instance group is a set of instances, which will be registered as kubernetes nodes. On AWS this is implemented via auto-scaling-groups. You can have several instance groups, for example if you wanted nodes that are a mix of spot and on-demand instances, or GPU and non-GPU instances.

(5/5) Create the cluster in AWS

Run “kops update cluster” to create your cluster in AWS:

```
kops update cluster useast1.dev.example.com --yes
```

That takes a few seconds to run, but then your cluster will likely take a few minutes to actually be ready. `kops update cluster` will be the tool you’ll use whenever you change the configuration of your cluster; it applies the changes you have made to the configuration to your cluster - reconfiguring AWS or kubernetes as needed.

For example, after you `kops edit ig nodes`, then `kops update cluster --yes` to apply your configuration, and sometimes you will also have to `kops rolling-update cluster` to roll out the configuration immediately.

Without `--yes`, `kops update cluster` will show you a preview of what it is going to do. This is handy for production clusters!

Explore other add-ons

See the list of add-ons to explore other add-ons, including tools for logging, monitoring, network policy, visualization & control of your Kubernetes cluster.

What’s next

- Learn more about Kubernetes concepts and `kubectl`.
- Learn about `kops` advanced usage

Cleanup

- To delete your cluster: `kops delete cluster useast1.dev.example.com --yes`

Feedback

- Slack Channel: #sig-aws has a lot of kops users
- GitHub Issues

Create an Issue Edit this Page

Edit This Page

CoreOS on AWS or GCE

-
  -
    - Official CoreOS Guides
    - Community Guides
  - Support Level

There are multiple guides on running Kubernetes with CoreOS:

Official CoreOS Guides

These guides are maintained by CoreOS and deploy Kubernetes the "CoreOS Way" with full TLS, the DNS add-on, and more. These guides pass Kubernetes conformance testing and we encourage you to test this yourself.

AWS Multi-Node

Guide and CLI tool for setting up a multi-node cluster on AWS. CloudFormation is used to set up a master and multiple workers in auto-scaling groups.

Bare Metal Multi-Node

Guide and HTTP/API service for PXE booting and provisioning a multi-node cluster on bare metal. Ignition is used to provision a master and multiple workers on the first boot from disk.

Vagrant Multi-Node

Guide to setting up a multi-node cluster on Vagrant. The deployer can independently configure the number of etcd nodes, master nodes, and worker nodes to bring up a fully HA control plane.

Vagrant Single-Node

The quickest way to set up a Kubernetes development environment locally. As easy as `git clone`, `vagrant up` and configuring `kubectl`.

Full Step by Step Guide

A generic guide to setting up an HA cluster on any cloud or bare metal, with full TLS. Repeat the master or worker steps to configure more machines of that role.

Community Guides

These guides are maintained by community members, cover specific platforms and use cases, and experiment with different ways of configuring Kubernetes on CoreOS.

Easy Multi-node Cluster on Google Compute Engine

Scripted installation of a single master, multi-worker cluster on GCE. Kubernetes components are managed by fleet.

Multi-node cluster using cloud-config and Weave on Vagrant

Configure a Vagrant-based cluster of 3 machines with networking provided by Weave.

Multi-node cluster using cloud-config and Vagrant

Configure a single master, multi-worker cluster locally, running on your choice of hypervisor: VirtualBox, Parallels, or VMware

---

Single-node cluster using a small OS X App

Guide to running a solo cluster (master + worker) controlled by an OS X menubar application. Uses xhyve + CoreOS under the hood.

Multi-node cluster with Vagrant and fleet units using a small OS X App

Guide to running a single master, multi-worker cluster controlled by an OS X menubar application. Uses Vagrant under the hood.

Multi-node cluster using cloud-config, CoreOS and VMware ESXi

Configure a single master, single worker cluster on VMware ESXi.

Single/Multi-node cluster using cloud-config, CoreOS and Foreman

Configure a standalone Kubernetes or a Kubernetes cluster with Foreman.

Support Level

IaaS Provider	Config. Mgmt	OS	Networking	Docs	Conforms	Support Level
GCE	CoreOS	CoreOS	flannel	docs		Community (@pires)
Vagrant	CoreOS	CoreOS	flannel	docs		Community (@pires, @Antonio

For support level information on all solutions, see the Table of solutions chart.

Create an Issue Edit this Page

Edit This Page

Installing Kubernetes On-premises/Cloud Providers with Kubespray

Overview

This quickstart helps to install a Kubernetes cluster hosted on GCE, Azure, OpenStack, AWS, or Baremetal with Kubespray.

Kubespray is a composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks. Kubespray provides:

- a highly available cluster
- composable attributes
- support for most popular Linux distributions (CoreOS, Debian Jessie, Ubuntu 16.04, CentOS/RHEL 7, Fedora/CentOS Atomic)
- continuous integration tests

To choose a tool which best fits your use case, read this comparison to kubeadm and kops.

Creating a cluster

(1/5) Meet the underlay requirements

Provision servers with the following requirements:

- `Ansible v2.4` (or newer)
- `Jinja 2.9` (or newer)
- `python-netaddr` installed on the machine that running Ansible commands
- Target servers must have access to the Internet in order to pull docker images
- Target servers are configured to allow IPv4 forwarding
- Target servers have SSH connectivity (tcp/22) directly to your nodes or through a bastion host/ssh jump box
- Target servers have a privileged user
- Your SSH key must be copied to all the servers that are part of your inventory
- Firewall rules configured properly to allow Ansible and Kubernetes components to communicate
- If using a cloud provider, you must have the appropriate credentials available and exported as environment variables

Kubespray provides the following utilities to help provision your environment:

- Terraform scripts for the following cloud providers:
  - AWS
  - OpenStack

(2/5) Compose an inventory file

After you provision your servers, create an inventory file for Ansible. You can do this manually or via a dynamic inventory script. For more information, see "Building your own inventory".

(3/5) Plan your cluster deployment

Kubespray provides the ability to customize many aspects of the deployment:

- CNI (networking) plugins
- DNS configuration
- Choice of control plane: native/binary or containerized with docker or rkt)

- Component versions
- Calico route reflectors
- Component runtime options
- Certificate generation methods

Kubespray customizations can be made to a variable file. If you are just getting started with Kubespray, consider using the Kubespray defaults to deploy your cluster and explore Kubernetes.

(4/5) Deploy a Cluster

Next, deploy your cluster:

Cluster deployment using ansible-playbook.

```
ansible-playbook -i your/inventory/hosts.ini cluster.yml -b -v \
  --private-key=~/.ssh/private_key
```

Large deployments (100+ nodes) may require specific adjustments for best results.

(5/5) Verify the deployment

Kubespray provides a way to verify inter-pod connectivity and DNS resolve with Netchecker. Netchecker ensures the netchecker-agents pods can resolve DNS requests and ping each over within the default namespace. Those pods mimic similar behavior of the rest of the workloads and serve as cluster health indicators.

Cluster operations

Kubespray provides additional playbooks to manage your cluster: *scale* and *upgrade*.

Scale your cluster

You can add worker nodes from your cluster by running the scale playbook. For more information, see “Adding nodes”. You can remove worker nodes from your cluster by running the remove-node playbook. For more information, see “Remove nodes”.

Upgrade your cluster

You can upgrade your cluster by running the upgrade-cluster playbook. For more information, see "Upgrades".

What's next

Check out planned work on Kubespray's roadmap.

Cleanup

You can reset your nodes and wipe out all components installed with Kubespray via the reset playbook.

> **Caution:** When running the reset playbook, be sure not to accidentally target your production cluster!

Feedback

- Slack Channel: #kubespray
- GitHub Issues

Create an Issue Edit this Page

Edit This Page

Installing Kubernetes on AWS with kops

Overview

This quickstart shows you how to easily install a Kubernetes cluster on AWS. It uses a tool called `kops`.

kops is an opinionated provisioning system:

- Fully automated installation
- Uses DNS to identify clusters
- Self-healing: everything runs in Auto-Scaling Groups
- Limited OS support (Debian preferred, Ubuntu 16.04 supported, early support for CentOS & RHEL)
- High-Availability support
- Can directly provision, or generate terraform manifests

If your opinions differ from these you may prefer to build your own cluster using kubeadm as a building block. kops builds on the kubeadm work.

Creating a cluster

(1/5) Install kops

Requirements

You must have kubectl installed in order for kops to work.

Installation

Download kops from the releases page (it is also easy to build from source):

On MacOS:

```
curl -OL https://github.com/kubernetes/kops/releases/download/1.8.0/kops-darwin-amd64
chmod +x kops-darwin-amd64
mv kops-darwin-amd64 /usr/local/bin/kops
# you can also install using Homebrew
brew update && brew install kops
```

On Linux:

```
wget https://github.com/kubernetes/kops/releases/download/1.8.0/kops-linux-amd64
chmod +x kops-linux-amd64
mv kops-linux-amd64 /usr/local/bin/kops
```

(2/5) Create a route53 domain for your cluster

kops uses DNS for discovery, both inside the cluster and so that you can reach the kubernetes API server from clients.

kops has a strong opinion on the cluster name: it should be a valid DNS name. By doing so you will no longer get your clusters confused, you can share clusters with your colleagues unambiguously, and you can reach them without relying on remembering an IP address.

You can, and probably should, use subdomains to divide your clusters. As our example we will use `useast1.dev.example.com`. The API server endpoint will then be `api.useast1.dev.example.com`.

A Route53 hosted zone can serve subdomains. Your hosted zone could be `useast1.dev.example.com`, but also `dev.example.com` or even `example.com`. kops works with any of these, so typically you choose for organization reasons (e.g. you are allowed to create records under `dev.example.com`, but not under `example.com`).

Let's assume you're using `dev.example.com` as your hosted zone. You create that hosted zone using the normal process, or with a command such as `aws route53 create-hosted-zone --name dev.example.com --caller-reference 1`.

You must then set up your NS records in the parent domain, so that records in the domain will resolve. Here, you would create NS records in `example.com` for `dev`. If it is a root domain name you would configure the NS records at your domain registrar (e.g. `example.com` would need to be configured where you bought `example.com`).

This step is easy to mess up (it is the #1 cause of problems!) You can double-check that your cluster is configured correctly if you have the dig tool by running:

```
dig NS dev.example.com
```

You should see the 4 NS records that Route53 assigned your hosted zone.

(3/5) Create an S3 bucket to store your clusters state

kops lets you manage your clusters even after installation. To do this, it must keep track of the clusters that you have created, along with their configuration, the keys they are using etc. This information is stored in an S3 bucket. S3 permissions are used to control access to the bucket.

Multiple clusters can use the same S3 bucket, and you can share an S3 bucket between your colleagues that administer the same clusters - this is much easier than passing around kubecfg files. But anyone with access to the S3 bucket will have administrative access to all your clusters, so you don't want to share it beyond the operations team.

So typically you have one S3 bucket for each ops team (and often the name will correspond to the name of the hosted zone above!)

In our example, we chose `dev.example.com` as our hosted zone, so let's pick `clusters.dev.example.com` as the S3 bucket name.

- Export `AWS_PROFILE` (if you need to select a profile for the AWS CLI to work)
- Create the S3 bucket using `aws s3 mb s3://clusters.dev.example.com`
- You can `export KOPS_STATE_STORE=s3://clusters.dev.example.com` and then kops will use this location by default. We suggest putting this in your bash profile or similar.

(4/5) Build your cluster configuration

Run "kops create cluster" to create your cluster configuration:

```
kops create cluster --zones=us-east-1c useast1.dev.example.com
```

kops will create the configuration for your cluster. Note that it *only* creates the configuration, it does not actually create the cloud resources - you'll do that in the next step with a `kops update cluster`. This give you an opportunity to review the configuration or change it.

It prints commands you can use to explore further:

- List your clusters with: `kops get cluster`
- Edit this cluster with: `kops edit cluster useast1.dev.example.com`
- Edit your node instance group: `kops edit ig --name=useast1.dev.example.com nodes`
- Edit your master instance group: `kops edit ig --name=useast1.dev.example.com master-us-east-1c`

If this is your first time using kops, do spend a few minutes to try those out! An instance group is a set of instances, which will be registered as kubernetes nodes. On AWS this is implemented via auto-scaling-groups. You can have several instance groups, for example if you wanted nodes that are a mix of spot and on-demand instances, or GPU and non-GPU instances.

(5/5) Create the cluster in AWS

Run "kops update cluster" to create your cluster in AWS:

```
kops update cluster useast1.dev.example.com --yes
```

That takes a few seconds to run, but then your cluster will likely take a few minutes to actually be ready. `kops update cluster` will be the tool you'll use whenever you change the configuration of your cluster; it applies the changes you have made to the configuration to your cluster - reconfiguring AWS or kubernetes as needed.

For example, after you `kops edit ig nodes`, then `kops update cluster --yes` to apply your configuration, and sometimes you will also have to `kops rolling-update cluster` to roll out the configuration immediately.

Without `--yes`, `kops update cluster` will show you a preview of what it is going to do. This is handy for production clusters!

Explore other add-ons

See the list of add-ons to explore other add-ons, including tools for logging, monitoring, network policy, visualization & control of your Kubernetes cluster.

What's next

- Learn more about Kubernetes concepts and `kubectl`.
- Learn about `kops` advanced usage

Cleanup

- To delete your cluster: `kops delete cluster useast1.dev.example.com --yes`

Feedback

- Slack Channel: #sig-aws has a lot of kops users
- GitHub Issues

Create an Issue Edit this Page

Edit This Page

Building Large Clusters

Support

At v1.10, Kubernetes supports clusters with up to 5000 nodes. More specifically, we support configurations that meet *all* of the following criteria:

- No more than 5000 nodes
- No more than 150000 total pods
- No more than 300000 total containers

- No more than 100 pods per node
-
  - Support
  - Setup
    - Quota Issues
    - Etcd storage
    - Size of master and master components
    - Addon Resources
    - Allowing minor node failure at startup

Setup

A cluster is a set of nodes (physical or virtual machines) running Kubernetes agents, managed by a "master" (the cluster-level control plane).

Normally the number of nodes in a cluster is controlled by the value `NUM_NODES` in the platform-specific `config-default.sh` file (for example, see GCE's `config-default.sh`).

Simply changing that value to something very large, however, may cause the setup script to fail for many cloud providers. A GCE deployment, for example, will run in to quota issues and fail to bring the cluster up.

When setting up a large Kubernetes cluster, the following issues must be considered.

Quota Issues

To avoid running into cloud provider quota issues, when creating a cluster with many nodes, consider:

- Increase the quota for things like CPU, IPs, etc.
  - In GCE, for example, you'll want to increase the quota for:
  - CPUs
  - VM instances
  - Total persistent disk reserved
  - In-use IP addresses
  - Firewall Rules
  - Forwarding rules
  - Routes
  - Target pools
- Gating the setup script so that it brings up new node VMs in smaller batches with waits in between, because some cloud providers rate limit the creation of VMs.

Etcd storage

To improve performance of large clusters, we store events in a separate dedicated etcd instance.

When creating a cluster, existing salt scripts:

- start and configure additional etcd instance
- configure api-server to use it for storing events

Size of master and master components

On GCE/Google Kubernetes Engine, and AWS, `kube-up` automatically configures the proper VM size for your master depending on the number of nodes in your cluster. On other providers, you will need to configure it manually. For reference, the sizes we use on GCE are

- 1-5 nodes: n1-standard-1
- 6-10 nodes: n1-standard-2
- 11-100 nodes: n1-standard-4
- 101-250 nodes: n1-standard-8
- 251-500 nodes: n1-standard-16
- more than 500 nodes: n1-standard-32

And the sizes we use on AWS are

- 1-5 nodes: m3.medium
- 6-10 nodes: m3.large
- 11-100 nodes: m3.xlarge
- 101-250 nodes: m3.2xlarge
- 251-500 nodes: c4.4xlarge
- more than 500 nodes: c4.8xlarge

Note that these master node sizes are currently only set at cluster startup time, and are not adjusted if you later scale your cluster up or down (e.g. manually removing or adding nodes, or using a cluster autoscaler).

Addon Resources

To prevent memory leaks or other resource issues in cluster addons from consuming all the resources available on a node, Kubernetes sets resource limits on addon containers to limit the CPU and Memory resources they can consume (See PR #10653 and #10778).

For example:

```
containers:
- name: fluentd-cloud-logging
```

```
    image: k8s.gcr.io/fluentd-gcp:1.16
    resources:
      limits:
        cpu: 100m
        memory: 200Mi
```

Except for Heapster, these limits are static and are based on data we collected from addons running on 4-node clusters (see #10335). The addons consume a lot more resources when running on large deployment clusters (see #5880). So, if a large cluster is deployed without adjusting these values, the addons may continuously get killed because they keep hitting the limits.

To avoid running into cluster addon resource issues, when creating a cluster with many nodes, consider the following:

- Scale memory and CPU limits for each of the following addons, if used, as you scale up the size of cluster (there is one replica of each handling the entire cluster so memory and CPU usage tends to grow proportionally with size/load on cluster):
  - InfluxDB and Grafana
  - kubedns, dnsmasq, and sidecar
  - Kibana
- Scale number of replicas for the following addons, if used, along with the size of cluster (there are multiple replicas of each so increasing replicas should help handle increased load, but, since load per replica also increases slightly, also consider increasing CPU/memory limits):
  - elasticsearch
- Increase memory and CPU limits slightly for each of the following addons, if used, along with the size of cluster (there is one replica per node but CPU/memory usage increases slightly along with cluster load/size as well):
  - FluentD with ElasticSearch Plugin
  - FluentD with GCP Plugin

Heapster's resource limits are set dynamically based on the initial size of your cluster (see #16185 and #22940). If you find that Heapster is running out of resources, you should adjust the formulas that compute heapster memory request (see those PRs for details).

For directions on how to detect if addon containers are hitting resource limits, see the Troubleshooting section of Compute Resources.

In the future, we anticipate to set all cluster addon resource limits based on cluster size, and to dynamically adjust them if you grow or shrink your cluster. We welcome PRs that implement those features.

Allowing minor node failure at startup

For various reasons (see #18969 for more details) running `kube-up.sh` with a very large `NUM_NODES` may fail due to a very small number of nodes not coming up properly. Currently you have two choices: restart the cluster (`kube-down.sh` and then `kube-up.sh` again), or before running `kube-up.sh` set the environment variable `ALLOWED_NOTREADY_NODES` to whatever value you feel comfortable with. This will allow `kube-up.sh` to succeed with fewer than `NUM_NODES` coming up. Depending on the reason for the failure, those additional nodes may join later or the cluster may remain at a size of `NUM_NODES - ALLOWED_NOTREADY_NODES`.

Create an Issue Edit this Page

Edit This Page

Building from Source

- – Building from source

You can either build a release from source or download a pre-built release. If you do not plan on developing Kubernetes itself, we suggest using a pre-built version of the current release, which can be found in the Release Notes.

The Kubernetes source code can be downloaded from the kubernetes/kubernetes repo.

Building from source

If you are simply building a release from source there is no need to set up a full golang environment as all building happens in a Docker container.

Building a release is simple.

```
git clone https://github.com/kubernetes/kubernetes.git
cd kubernetes
make release
```

For more details on the release process see the kubernetes/kubernetes `build` directory.

Create an Issue Edit this Page

Edit This Page

Configuring Kubernetes with Salt

- - Salt cluster setup
  - Standalone Salt Configuration on GCE and others
  - Salt security
  - Salt minion configuration
  - Best Practices
  - Future enhancements (Networking)

The Kubernetes cluster can be configured using Salt.

The Salt scripts are shared across multiple hosting providers and depending on where you host your Kubernetes cluster, you may be using different operating systems and different networking configurations. As a result, it's important to understand some background information before making Salt changes in order to minimize introducing failures for other hosting providers.

Salt cluster setup

The **salt-master** service runs on the kubernetes-master (except on the default GCE and OpenStack-Heat setup).

The **salt-minion** service runs on the kubernetes-master and each kubernetes-node in the cluster.

Each salt-minion service is configured to interact with the **salt-master** service hosted on the kubernetes-master via the **master.conf** file (except on GCE and OpenStack-Heat).

```
[root@kubernetes-master] $ cat /etc/salt/minion.d/master.conf
master: kubernetes-master
```

The salt-master is contacted by each salt-minion and depending upon the machine information presented, the salt-master will provision the machine as either a kubernetes-master or kubernetes-node with all the required capabilities needed to run Kubernetes.

If you are running the Vagrant based environment, the **salt-api** service is running on the kubernetes-master. It is configured to enable the vagrant user to introspect the salt cluster in order to find out about machines in the Vagrant environment via a REST API.

Standalone Salt Configuration on GCE and others

On GCE and OpenStack, using the Openstack-Heat provider, the master and nodes are all configured as standalone minions. The configuration for each VM is derived from the VM's instance metadata and then stored in Salt grains (`/etc/salt/minion.d/grains.conf`) and pillars (`/srv/salt-overlay/pillar/cluster-params.sls`) that local Salt uses to enforce state.

All remaining sections that refer to master/minion setups should be ignored for GCE and OpenStack. One fallout of this setup is that the Salt mine doesn't exist - there is no sharing of configuration amongst nodes.

Salt security

(Not applicable on default GCE and OpenStack-Heat setup.)

Security is not enabled on the salt-master, and the salt-master is configured to auto-accept incoming requests from minions. It is not recommended to use this security configuration in production environments without deeper study. (In some environments this isn't as bad as it might sound if the salt master port isn't externally accessible and you trust everyone on your network.)

```
[root@kubernetes-master] $ cat /etc/salt/master.d/auto-accept.conf
open_mode: True
auto_accept: True
```

Salt minion configuration

Each minion in the salt cluster has an associated configuration that instructs the salt-master how to provision the required resources on the machine.

An example file is presented below using the Vagrant based environment.

```
[root@kubernetes-master] $ cat /etc/salt/minion.d/grains.conf
grains:
  etcd_servers: $MASTER_IP
  cloud: vagrant
  roles:
    - kubernetes-master
```

Each hosting environment has a slightly different grains.conf file that is used to build conditional logic where required in the Salt files.

The following enumerates the set of defined key/value pairs that are supported today. If you add new ones, please make sure to update this list.

Key	Value
`api_servers`	(Optional) The IP address / host name where a kubelet can get read-only access
`cbr-cidr`	(Optional) The minion IP address range used for the docker container bridge.
`cloud`	(Optional) Which IaaS platform is used to host Kubernetes, *gce*, *azure*, *aws*, *vagr*
`etcd_servers`	(Optional) Comma-delimited list of IP addresses the kube-apiserver and kubelet u

Key	Value
`hostnamef`	(Optional) The full host name of the machine, i.e. uname -n
`node_ip`	(Optional) The IP address to use to address this node
`hostname_override`	(Optional) Mapped to the kubelet hostname-override
`network_mode`	(Optional) Networking model to use among nodes: *openvswitch*
`networkInterfaceName`	(Optional) Networking interface to use to bind addresses, default value *eth0*
`publicAddressOverride`	(Optional) The IP address the kube-apiserver should use to bind against for exter
`roles`	(Required) 1. `kubernetes-master` means this machine is the master in the Kube

These keys may be leveraged by the Salt sls files to branch behavior.

In addition, a cluster may be running a Debian based operating system or Red Hat based operating system (Centos, Fedora, RHEL, etc.). As a result, it's important to sometimes distinguish behavior based on operating system using if branches like the following.

```
{% if grains['os_family'] == 'RedHat' %}
// something specific to a RedHat environment (Centos, Fedora, RHEL) where you may use yum,
{% else %}
// something specific to Debian environment (apt-get, initd)
{% endif %}
```

Best Practices

When configuring default arguments for processes, it's best to avoid the use of EnvironmentFiles (Systemd in Red Hat environments) or init.d files (Debian distributions) to hold default values that should be common across operating system environments. This helps keep our Salt template files easy to understand for editors who may not be familiar with the particulars of each distribution.

Future enhancements (Networking)

Per pod IP configuration is provider-specific, so when making networking changes, it's important to sandbox these as all providers may not use the same mechanisms (iptables, openvswitch, etc.)

We should define a grains.conf key that captures more specifically what network configuration environment is being used to avoid future confusion across providers.

Create an Issue Edit this Page

Edit This Page

Picking the Right Solution

Kubernetes can run on various platforms: from your laptop, to VMs on a cloud provider, to a rack of bare metal servers. The effort required to set up a cluster varies from running a single command to crafting your own customized cluster. Use this guide to choose a solution that fits your needs.

If you just want to "kick the tires" on Kubernetes, use the local Docker-based solutions.

When you are ready to scale up to more machines and higher availability, a hosted solution is the easiest to create and maintain.

Turnkey cloud solutions require only a few commands to create and cover a wide range of cloud providers. On-Premises turnkey cloud solutions have the simplicity of the turnkey cloud solution combined with the security of your own private network.

If you already have a way to configure hosting resources, use kubeadm to easily bring up a cluster with a single command per machine.

Custom solutions vary from step-by-step instructions to general advice for setting up a Kubernetes cluster from scratch.

- Local-machine Solutions
- Hosted Solutions
- Turnkey Cloud Solutions
- On-Premises turnkey cloud solutions
- Custom Solutions
  - Universal
  - Cloud
  - On-Premises VMs
  - Bare Metal
  - Integrations
- Table of Solutions
  - Definition of columns

Local-machine Solutions

- Minikube is the recommended method for creating a local, single-node Kubernetes cluster for development and testing. Setup is completely automated and doesn't require a cloud provider account.

- Kubeadm-dind is a multi-node (while minikube is single-node) Kubernetes cluster which only requires a docker daemon. It uses docker-in-docker technique to spawn the Kubernetes cluster.

- Ubuntu on LXD supports a nine-instance deployment on localhost.

- IBM Cloud Private-CE (Community Edition) can use VirtualBox on your machine to deploy Kubernetes to one or more VMs for development and test scenarios. Scales to full multi-node cluster.
- IBM Cloud Private-CE (Community Edition) on Linux Containers is a Terraform/Packer/BASH based Infrastructure as Code (IaC) scripts to create a seven node (1 Boot, 1 Master, 1 Management, 1 Proxy and 3 Workers) LXD cluster on Linux Host.

Hosted Solutions

- Google Kubernetes Engine offers managed Kubernetes clusters.
- Amazon Elastic Container Service for Kubernetes offers managed Kubernetes service.
- Azure Container Service offers managed Kubernetes clusters.
- Stackpoint.io provides Kubernetes infrastructure automation and management for multiple public clouds.
- AppsCode.com provides managed Kubernetes clusters for various public clouds, including AWS and Google Cloud Platform.
- Madcore.Ai is devops-focused CLI tool for deploying Kubernetes infrastructure in AWS. Master, auto-scaling group nodes with spot-instances, ingress-ssl-lego, Heapster, and Grafana.
- Platform9 offers managed Kubernetes on-premises or on any public cloud, and provides 24/7 health monitoring and alerting. (Kube2go, a web-UI driven Kubernetes cluster deployment service Platform9 released, has been integrated to Platform9 Sandbox.)
- OpenShift Dedicated offers managed Kubernetes clusters powered by OpenShift.
- OpenShift Online provides free hosted access for Kubernetes applications.
- IBM Cloud Container Service offers managed Kubernetes clusters with isolation choice, operational tools, integrated security insight into images and containers, and integration with Watson, IoT, and data.
- Giant Swarm offers managed Kubernetes clusters in their own datacenter, on-premises, or on public clouds.
- Kubermatic provides managed Kubernetes clusters for various public clouds, including AWS and Digital Ocean, as well as on-premises with OpenStack integration.

- Pivotal Container Service provides enterprise-grade Kubernetes for both on-premises and public clouds. PKS enables on-demand provisioning of Kubernetes clusters, multi-tenancy and fully automated day-2 operations.
- Oracle Container Engine for Kubernetes is a fully-managed, scalable, and highly available service that you can use to deploy your containerized applications to the cloud.

Turnkey Cloud Solutions

These solutions allow you to create Kubernetes clusters on a range of Cloud IaaS providers with only a few commands. These solutions are actively developed and have active community support.

- Conjure-up Kubernetes with Ubuntu on AWS, Azure, Google Cloud, Oracle Cloud
- Google Compute Engine (GCE)
- AWS
- Azure
- Tectonic by CoreOS
- CenturyLink Cloud
- IBM Cloud
- Stackpoint.io
- Madcore.Ai
- Kubermatic
- Rancher 2.0
- Oracle Container Engine for K8s
- Gardener

On-Premises turnkey cloud solutions

These solutions allow you to create Kubernetes clusters on your internal, secure, cloud network with only a few commands.

- IBM Cloud Private
- Kubermatic
- SUSE CaaS Platform
- SUSE Cloud Application Platform
- Rancher 2.0

Custom Solutions

Kubernetes can run on a wide range of Cloud providers and bare-metal environments, and with many base operating systems.

If you can find a guide below that matches your needs, use it. It may be a little out of date, but it will be easier than starting from scratch. If you do want to start from scratch, either because you have special requirements, or just because you want to understand what is underneath a Kubernetes cluster, try the Getting Started from Scratch guide.

If you are interested in supporting Kubernetes on a new platform, see Writing a Getting Started Guide.

Universal

If you already have a way to configure hosting resources, use kubeadm to easily bring up a cluster with a single command per machine.

Cloud

These solutions are combinations of cloud providers and operating systems not covered by the above solutions.

- CoreOS on AWS or GCE
- Kubernetes on Ubuntu
- Kubespray
- Rancher Kubernetes Engine (RKE)
- Gardener

On-Premises VMs

- Vagrant (uses CoreOS and flannel)
- CloudStack (uses Ansible, CoreOS and flannel)
- VMware vSphere
- VMware vSphere, OpenStack, or Bare Metal (uses Juju, Ubuntu and flannel)
- VMware (uses CoreOS and flannel)
- oVirt
- Fedora (Multi Node) (uses Fedora and flannel)

Bare Metal

- Fedora (Single Node)
- Fedora (Multi Node)
- Kubernetes on Ubuntu
- CoreOS

Integrations

These solutions provide integration with third-party schedulers, resource managers, and/or lower level platforms.

- DCOS
  - Community Edition DCOS uses AWS
  - Enterprise Edition DCOS supports cloud hosting, on-premises VMs, and bare metal

Table of Solutions

Below is a table of all of the solutions listed above.

IaaS Provider	Config. Mgmt.	OS	Networking	Docs	S
any	any	multi-support	any CNI	docs	F
Google Kubernetes Engine			GCE	docs	C
Stackpoint.io		multi-support	multi-support	docs	C
AppsCode.com	Saltstack	Debian	multi-support	docs	C
Madcore.Ai	Jenkins DSL	Ubuntu	flannel	docs	C
Platform9		multi-support	multi-support	docs	C
Kubermatic		multi-support	multi-support	docs	C
Giant Swarm		CoreOS	flannel and/or Calico	docs	C
GCE	Saltstack	Debian	GCE	docs	F
Azure Container Service		Ubuntu	Azure	docs	C
Azure (IaaS)		Ubuntu	Azure	docs	C
Bare-metal	custom	Fedora	*none*	docs	F
Bare-metal	custom	Fedora	flannel	docs	C
libvirt	custom	Fedora	flannel	docs	C
KVM	custom	Fedora	flannel	docs	C
DCOS	Marathon	CoreOS/Alpine	custom	docs	C
AWS	CoreOS	CoreOS	flannel	docs	C
GCE	CoreOS	CoreOS	flannel	docs	C
Vagrant	CoreOS	CoreOS	flannel	docs	C
CloudStack	Ansible	CoreOS	flannel	docs	C
VMware vSphere	any	multi-support	multi-support	docs	C

IaaS Provider	Config. Mgmt.	OS	Networking	Docs	S
Bare-metal	custom	CentOS	flannel	docs	C
lxd	Juju	Ubuntu	flannel/canal	docs	C
AWS	Juju	Ubuntu	flannel/calico/canal	docs	C
Azure	Juju	Ubuntu	flannel/calico/canal	docs	C
GCE	Juju	Ubuntu	flannel/calico/canal	docs	C
Oracle Cloud	Juju	Ubuntu	flannel/calico/canal	docs	C
Rackspace	Juju	Ubuntu	flannel/calico/canal	docs	C
VMware vSphere	Juju	Ubuntu	flannel/calico/canal	docs	C
Bare Metal	Juju	Ubuntu	flannel/calico/canal	docs	C
AWS	Saltstack	Debian	AWS	docs	C
AWS	kops	Debian	AWS	docs	C
Bare-metal	custom	Ubuntu	flannel	docs	C
oVirt				docs	C
any	any	any	any	docs	C
any	any	any	any	docs	C
any	RKE	multi-support	flannel or canal	docs	C
any	Gardener Cluster-Operator	multi-support	multi-support	docs	I

Note: The above table is ordered by version test/used in nodes, followed by support level.

Definition of columns

- **IaaS Provider** is the product or organization which provides the virtual or physical machines (nodes) that Kubernetes runs on.
- **OS** is the base operating system of the nodes.
- **Config. Mgmt.** is the configuration management system that helps install and maintain Kubernetes on the nodes.
- **Networking** is what implements the networking model. Those with networking type *none* may not support more than a single node, or may support multiple VM nodes in a single physical node.
- **Conformance** indicates whether a cluster created with this configuration has passed the project's conformance tests for supporting the API and base features of Kubernetes v1.0.0.
- **Support Levels**
  - **Project**: Kubernetes committers regularly use this configuration, so it usually works with the latest release of Kubernetes.
  - **Commercial**: A commercial offering with its own support arrangements.
  - **Community**: Actively supported by community contributions. May not work with recent releases of Kubernetes.

  - **Inactive**: Not actively maintained. Not recommended for first-time Kubernetes users, and may be removed.
- **Notes** has other relevant information, such as the version of Kubernetes used.

Create an Issue Edit this Page

Edit This Page

Running Kubernetes Locally via Minikube

Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

- Add-ons
- Using Minikube with an HTTP Proxy
- Known Issues
- Design
- Additional Links:
- Community

Minikube Features

- Minikube supports Kubernetes features such as:
  - DNS
  - NodePorts
  - ConfigMaps and Secrets
  - Dashboards
  - Container Runtime: Docker, rkt and CRI-O
  - Enabling CNI (Container Network Interface)
  - Ingress

Installation

See Installing Minikube.

Quickstart

Here's a brief demo of minikube usage. If you want to change the VM driver add the appropriate `--vm-driver=xxx` flag to `minikube start`. Minikube supports the following drivers:

- virtualbox
- vmwarefusion
- kvm (driver installation)
- hyperkit (driver installation)
- xhyve (driver installation) (deprecated)

Note that the IP below is dynamic and can change. It can be retrieved with `minikube ip`.

```
$ minikube start
Starting local Kubernetes cluster...
Running pre-create checks...
Creating machine...
Starting local Kubernetes cluster...
```

```
$ kubectl run hello-minikube --image=k8s.gcr.io/echoserver:1.10 --port=8080
deployment "hello-minikube" created
$ kubectl expose deployment hello-minikube --type=NodePort
service "hello-minikube" exposed

# We have now launched an echoserver pod but we have to wait until the pod is up before cur
# via the exposed service.
# To check whether the pod is up and running we can use the following:
$ kubectl get pod
NAME                              READY     STATUS              RESTARTS   AGE
hello-minikube-3383150820-vctvh   0/1       ContainerCreating   0          3s
# We can see that the pod is still being created from the ContainerCreating status
$ kubectl get pod
NAME                              READY     STATUS    RESTARTS   AGE
hello-minikube-3383150820-vctvh   1/1       Running   0          13s
# We can see that the pod is now Running and we will now be able to curl it:
$ curl $(minikube service hello-minikube --url)
CLIENT VALUES:
client_address=192.168.99.1
command=GET
real path=/
...
$ kubectl delete services hello-minikube
service "hello-minikube" deleted
$ kubectl delete deployment hello-minikube
deployment "hello-minikube" deleted
$ minikube stop
Stopping local Kubernetes cluster...
Stopping "minikube"...
```

Alternative Container Runtimes

CRI-O

To use CRI-O as the container runtime, run:

```
$ minikube start \
    --network-plugin=cni \
    --container-runtime=cri-o \
    --bootstrapper=kubeadm
```

Or you can use the extended version:

```
$ minikube start \
    --network-plugin=cni \
    --extra-config=kubelet.container-runtime=remote \
    --extra-config=kubelet.container-runtime-endpoint=/var/run/crio.sock \
    --extra-config=kubelet.image-service-endpoint=/var/run/crio.sock \
    --bootstrapper=kubeadm
```

rkt container engine

To use rkt as the container runtime run:

```
$ minikube start \
    --network-plugin=cni \
    --container-runtime=rkt
```

This will use an alternative minikube ISO image containing both rkt, and Docker, and enable CNI networking.

Driver plugins

See DRIVERS for details on supported drivers and how to install plugins, if required.

Reusing the Docker daemon

When using a single VM of Kubernetes, it's really handy to reuse the minikube's built-in Docker daemon; as this means you don't have to build a docker registry on your host machine and push the image into it - you can just build inside the same docker daemon as minikube which speeds up local experiments. Just make sure you tag your Docker image with something other than 'latest' and use that tag while you pull the image. Otherwise, if you do not specify version of your image, it will be assumed as `:latest`, with pull image policy of `Always` correspondingly, which may eventually result in `ErrImagePull` as you may not have any versions of your Docker image out there in the default docker registry (usually DockerHub) yet.

To be able to work with the docker daemon on your mac/linux host use the `docker-env command` in your shell:

```
eval $(minikube docker-env)
```

You should now be able to use docker on the command line on your host mac/linux machine talking to the docker daemon inside the minikube VM:

```
docker ps
```

On Centos 7, docker may report the following error:

```
Could not read CA certificate "/etc/docker/ca.pem": open /etc/docker/ca.pem: no such file o
```

The fix is to update /etc/sysconfig/docker to ensure that minikube's environment changes are respected:

```
< DOCKER_CERT_PATH=/etc/docker
---
> if [ -z "${DOCKER_CERT_PATH}" ]; then
>   DOCKER_CERT_PATH=/etc/docker
> fi
```

Remember to turn off the imagePullPolicy:Always, as otherwise Kubernetes won't use images you built locally.

Managing your Cluster

Starting a Cluster

The `minikube start` command can be used to start your cluster. This command creates and configures a virtual machine that runs a single-node Kubernetes cluster. This command also configures your kubectl installation to communicate with this cluster.

If you are behind a web proxy, you will need to pass this information in e.g. via

```
https_proxy=<my proxy> minikube start --docker-env http_proxy=<my proxy> --docker-env https_
```

Unfortunately just setting the environment variables will not work.

Minikube will also create a "minikube" context, and set it to default in kubectl. To switch back to this context later, run this command: `kubectl config use-context minikube`.

Specifying the Kubernetes version

Minikube supports running multiple different versions of Kubernetes. You can access a list of all available versions via

```
minikube get-k8s-versions
```

You can specify the specific version of Kubernetes for Minikube to use by adding the `--kubernetes-version` string to the `minikube start` command. For example, to run version `v1.7.3`, you would run the following:

```
minikube start --kubernetes-version v1.7.3
```

Configuring Kubernetes

Minikube has a "configurator" feature that allows users to configure the Kubernetes components with arbitrary values. To use this feature, you can use the `--extra-config` flag on the `minikube start` command.

This flag is repeated, so you can pass it several times with several different values to set multiple options.

This flag takes a string of the form `component.key=value`, where `component` is one of the strings from the below list, `key` is a value on the configuration struct and `value` is the value to set.

Valid keys can be found by examining the documentation for the Kubernetes `componentconfigs` for each component. Here is the documentation for each supported configuration:

- kubelet
- apiserver
- proxy
- controller-manager
- etcd
- scheduler

Examples

To change the `MaxPods` setting to 5 on the Kubelet, pass this flag: `--extra-config=kubelet.MaxPods=5`.

This feature also supports nested structs. To change the `LeaderElection.LeaderElect` setting to `true` on the scheduler, pass this flag: `--extra-config=scheduler.LeaderElection.LeaderElect=t`

To set the `AuthorizationMode` on the `apiserver` to `RBAC`, you can use: `--extra-config=apiserver.Authorization.Mode=RBAC`.

Stopping a Cluster

The `minikube stop` command can be used to stop your cluster. This command shuts down the minikube virtual machine, but preserves all cluster state and data. Starting the cluster again will restore it to it's previous state.

Deleting a Cluster

The `minikube delete` command can be used to delete your cluster. This command shuts down and deletes the minikube virtual machine. No data or state is preserved.

Interacting With your Cluster

Kubectl

The `minikube start` command creates a “kubectl context” called “minikube”. This context contains the configuration to communicate with your minikube cluster.

Minikube sets this context to default automatically, but if you need to switch back to it in the future, run:

`kubectl config use-context minikube`,

Or pass the context on each command like this: `kubectl get pods --context=minikube`.

Dashboard

To access the Kubernetes Dashboard, run this command in a shell after starting minikube to get the address:

```
minikube dashboard
```

Services

To access a service exposed via a node port, run this command in a shell after starting minikube to get the address:

```
minikube service [-n NAMESPACE] [--url] NAME
```

Networking

The minikube VM is exposed to the host system via a host-only IP address, that can be obtained with the `minikube ip` command. Any services of type `NodePort` can be accessed over that IP address, on the NodePort.

To determine the NodePort for your service, you can use a `kubectl` command like this:

```
kubectl get service $SERVICE --output='jsonpath="{.spec.ports[0].nodePort}"'
```

Persistent Volumes

Minikube supports PersistentVolumes of type `hostPath`. These PersistentVolumes are mapped to a directory inside the minikube VM.

The Minikube VM boots into a tmpfs, so most directories will not be persisted across reboots (`minikube stop`). However, Minikube is configured to persist files stored under the following host directories:

- `/data`
- `/var/lib/localkube`
- `/var/lib/docker`

Here is an example PersistentVolume config to persist data in the `/data` directory:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 5Gi
  hostPath:
    path: /data/pv0001/
```

Mounted Host Folders

Some drivers will mount a host folder within the VM so that you can easily share files between the VM and host. These are not configurable at the moment and different for the driver and OS you are using.

Note: Host folder sharing is not implemented in the KVM driver yet.

Driver	OS	HostFolder	VM
VirtualBox	Linux	/home	/hosthome
VirtualBox	OSX	/Users	/Users
VirtualBox	Windows	C://Users	/c/Users
VMware Fusion	OSX	/Users	/Users
Xhyve	OSX	/Users	/Users

Private Container Registries

To access a private container registry, follow the steps on this page.

We recommend you use `ImagePullSecrets`, but if you would like to configure access on the minikube VM you can place the `.dockercfg` in the `/home/docker` directory or the `config.json` in the `/home/docker/.docker` directory.

Add-ons

In order to have minikube properly start or restart custom addons, place the addons you wish to be launched with minikube in the `~/.minikube/addons` directory. Addons in this folder will be moved to the minikube VM and launched each time minikube is started or restarted.

Using Minikube with an HTTP Proxy

Minikube creates a Virtual Machine that includes Kubernetes and a Docker daemon. When Kubernetes attempts to schedule containers using Docker, the Docker daemon may require external network access to pull containers.

If you are behind an HTTP proxy, you may need to supply Docker with the proxy settings. To do this, pass the required environment variables as flags during `minikube start`.

For example:

```
$ minikube start --docker-env http_proxy=http://$YOURPROXY:PORT \
                 --docker-env https_proxy=https://$YOURPROXY:PORT
```

If your Virtual Machine address is 192.168.99.100, then chances are your proxy settings will prevent kubectl from directly reaching it. To by-pass proxy configuration for this IP address, you should modify your no_proxy settings. You can do so with:

```
$ export no_proxy=$no_proxy,$(minikube ip)
```

Known Issues

- Features that require a Cloud Provider will not work in Minikube. These include:
  - LoadBalancers
- Features that require multiple nodes. These include:
  - Advanced scheduling policies

Design

Minikube uses libmachine for provisioning VMs, and localkube (originally written and donated to this project by RedSpread) for running the cluster.

For more information about minikube, see the proposal.

Additional Links:

- **Goals and Non-Goals**: For the goals and non-goals of the minikube project, please see our roadmap.
- **Development Guide**: See CONTRIBUTING.md for an overview of how to send pull requests.
- **Building Minikube**: For instructions on how to build/test minikube from source, see the build guide
- **Adding a New Dependency**: For instructions on how to add a new dependency to minikube see the adding dependencies guide
- **Adding a New Addon**: For instruction on how to add a new addon for minikube see the adding an addon guide
- **Updating Kubernetes**: For instructions on how to update kubernetes see the updating Kubernetes guide

Community

Contributions, questions, and comments are all welcomed and encouraged! minikube developers hang out on Slack in the #minikube channel (get an invitation here). We also have the kubernetes-dev Google Groups mailing list. If you are posting to the list please prefix your subject with "minikube: ".

Create an Issue Edit this Page

Edit This Page

Running in Multiple Zones

Introduction

Kubernetes 1.2 adds support for running a single cluster in multiple failure zones (GCE calls them simply "zones", AWS calls them "availability zones", here we'll refer to them as "zones"). This is a lightweight version of a broader Cluster Federation feature (previously referred to by the affectionate nickname "Ubernetes"). Full Cluster Federation allows combining separate Kubernetes clusters running in different regions or cloud providers (or on-premises data centers). However, many users simply want to run a more available Kubernetes cluster in multiple zones of their single cloud provider, and this is what the multizone support in 1.2 allows (this previously went by the nickname "Ubernetes Lite").

Multizone support is deliberately limited: a single Kubernetes cluster can run in multiple zones, but only within the same region (and cloud provider). Only

GCE and AWS are currently supported automatically (though it is easy to add similar support for other clouds or even bare metal, by simply arranging for the appropriate labels to be added to nodes and volumes).

Functionality

When nodes are started, the kubelet automatically adds labels to them with zone information.

Kubernetes will automatically spread the pods in a replication controller or service across nodes in a single-zone cluster (to reduce the impact of failures.) With multiple-zone clusters, this spreading behavior is extended across zones (to reduce the impact of zone failures.) (This is achieved via `SelectorSpreadPriority`). This is a best-effort placement, and so if the zones in your cluster are heterogeneous (e.g. different numbers of nodes, different types of nodes, or different pod resource requirements), this might prevent perfectly even spreading of your pods across zones. If desired, you can use homogeneous zones (same number and types of nodes) to reduce the probability of unequal spreading.

When persistent volumes are created, the `PersistentVolumeLabel` admission controller automatically adds zone labels to them. The scheduler (via the `VolumeZonePredicate` predicate) will then ensure that pods that claim a given volume are only placed into the same zone as that volume, as volumes cannot be attached across zones.

Limitations

There are some important limitations of the multizone support:

- We assume that the different zones are located close to each other in the network, so we don't perform any zone-aware routing. In particular, traffic that goes via services might cross zones (even if pods in some pods backing that service exist in the same zone as the client), and this may incur additional latency and cost.

- Volume zone-affinity will only work with a `PersistentVolume`, and will not work if you directly specify an EBS volume in the pod spec (for example).
- Clusters cannot span clouds or regions (this functionality will require full federation support).
- Although your nodes are in multiple zones, kube-up currently builds a single master node by default. While services are highly available and can tolerate the loss of a zone, the control plane is located in a single zone. Users that want a highly available control plane should follow the high availability instructions.
- StatefulSet volume zone spreading when using dynamic provisioning is currently not compatible with pod affinity or anti-affinity policies.
- If the name of the StatefulSet contains dashes ("-"), volume zone spreading may not provide a uniform distribution of storage across zones.
- When specifying multiple PVCs in a Deployment or Pod spec, the StorageClass needs to be configured for a specific, single zone, or the PVs need to be statically provisioned in a specific zone. Another workaround is to use a StatefulSet, which will ensure that all the volumes for a replica are provisioned in the same zone.

Walkthrough

We're now going to walk through setting up and using a multi-zone cluster on both GCE & AWS. To do so, you bring up a full cluster (specifying `MULTIZONE=true`), and then you add nodes in additional zones by running `kube-up` again (specifying `KUBE_USE_EXISTING_MASTER=true`).

Bringing up your cluster

Create the cluster as normal, but pass MULTIZONE to tell the cluster to manage multiple zones; creating nodes in us-central1-a.

GCE:

```
curl -sS https://get.k8s.io | MULTIZONE=true KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-centra
```

AWS:

```
curl -sS https://get.k8s.io | MULTIZONE=true KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2
```

This step brings up a cluster as normal, still running in a single zone (but `MULTIZONE=true` has enabled multi-zone capabilities).

Nodes are labeled

View the nodes; you can see that they are labeled with zone information. They are all in `us-central1-a` (GCE) or `us-west-2a` (AWS) so far. The labels are `failure-domain.beta.kubernetes.io/region` for the region, and `failure-domain.beta.kubernetes.io/zone` for the zone:

```
> kubectl get nodes --show-labels
```

```
NAME                     STATUS                     AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   6m    v1.6.0+fff5156   beta.kubernetes.i
kubernetes-minion-87j9   Ready                      6m    v1.6.0+fff5156   beta.kubernetes.i
kubernetes-minion-9vlv   Ready                      6m    v1.6.0+fff5156   beta.kubernetes.i
kubernetes-minion-a12q   Ready                      6m    v1.6.0+fff5156   beta.kubernetes.i
```

Add more nodes in a second zone

Let's add another set of nodes to the existing cluster, reusing the existing master, running in a different zone (us-central1-b or us-west-2b). We run kube-up again, but by specifying `KUBE_USE_EXISTING_MASTER=true` kube-up will not create a new master, but will reuse one that was previously created instead.

GCE:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-centra
```

On AWS we also need to specify the network CIDR for the additional subnet, along with the master internal IP address:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2
```

View the nodes again; 3 more nodes should have launched and be tagged in us-central1-b:

```
> kubectl get nodes --show-labels
```

```
NAME                     STATUS                     AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   16m   v1.6.0+fff5156   beta.kubernetes.
kubernetes-minion-281d   Ready                      2m    v1.6.0+fff5156   beta.kubernetes.
kubernetes-minion-87j9   Ready                      16m   v1.6.0+fff5156   beta.kubernetes.
kubernetes-minion-9vlv   Ready                      16m   v1.6.0+fff5156   beta.kubernetes.
kubernetes-minion-a12q   Ready                      17m   v1.6.0+fff5156   beta.kubernetes.
kubernetes-minion-pp2f   Ready                      2m    v1.6.0+fff5156   beta.kubernetes.
kubernetes-minion-wf8i   Ready                      2m    v1.6.0+fff5156   beta.kubernetes.
```

Volume affinity

Create a volume using the dynamic volume creation (only PersistentVolumes are supported for zone affinity):

```
kubectl create -f - <<EOF
{
  "kind": "PersistentVolumeClaim",
  "apiVersion": "v1",
  "metadata": {
    "name": "claim1",
    "annotations": {
        "volume.alpha.kubernetes.io/storage-class": "foo"
    }
  },
  "spec": {
    "accessModes": [
      "ReadWriteOnce"
    ],
    "resources": {
      "requests": {
        "storage": "5Gi"
      }
    }
  }
}
EOF
```

NOTE: For version 1.3+ Kubernetes will distribute dynamic PV claims across the configured zones. For version 1.2, dynamic persistent volumes were always created in the zone of the cluster master (here us-central1-a / us-west-2a); that issue (#23330) was addressed in 1.3+.

Now lets validate that Kubernetes automatically labeled the zone & region the PV was created in.

```
> kubectl get pv --show-labels
NAME           CAPACITY   ACCESSMODES   STATUS    CLAIM            REASON    AGE       LABEI
pv-gce-mj4gm   5Gi        RWO           Bound     default/claim1             46s       failu
```

So now we will create a pod that uses the persistent volume claim. Because GCE PDs / AWS EBS volumes cannot be attached across zones, this means that this pod can only be created in the same zone as the volume:

```
kubectl create -f - <<EOF
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: claim1
EOF
```

Note that the pod was automatically created in the same zone as the volume, as cross-zone attachments are not generally permitted by cloud providers:

```
> kubectl describe pod mypod | grep Node
Node:        kubernetes-minion-9vlv/10.240.0.5
> kubectl get node kubernetes-minion-9vlv --show-labels
NAME                     STATUS    AGE    VERSION          LABELS
kubernetes-minion-9vlv   Ready     22m    v1.6.0+fff5156   beta.kubernetes.io/instance-type=
```

Pods are spread across zones

Pods in a replication controller or service are automatically spread across zones. First, let's launch more nodes in a third zone:

GCE:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-centra
```

AWS:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2
```

Verify that you now have nodes in 3 zones:

```
kubectl get nodes --show-labels
```

Create the guestbook-go example, which includes an RC of size 3, running a simple web app:

```
find kubernetes/examples/guestbook-go/ -name '*.json' | xargs -I {} kubectl create -f {}
```

The pods should be spread across all 3 zones:

```
>  kubectl describe pod -l app=guestbook | grep Node
Node:        kubernetes-minion-9vlv/10.240.0.5
Node:        kubernetes-minion-281d/10.240.0.8
Node:        kubernetes-minion-olsh/10.240.0.11
```

```
 > kubectl get node kubernetes-minion-9vlv kubernetes-minion-281d kubernetes-minion-olsh --s
NAME                     STATUS    AGE    VERSION          LABELS
kubernetes-minion-9vlv   Ready     34m    v1.6.0+fff5156   beta.kubernetes.io/instance-type=
kubernetes-minion-281d   Ready     20m    v1.6.0+fff5156   beta.kubernetes.io/instance-type=
kubernetes-minion-olsh   Ready     3m     v1.6.0+fff5156   beta.kubernetes.io/instance-type=
```

Load-balancers span all zones in a cluster; the guestbook-go example includes an example load-balanced service:

```
> kubectl describe service guestbook | grep LoadBalancer.Ingress
LoadBalancer Ingress:   130.211.126.21

> ip=130.211.126.21

> curl -s http://${ip}:3000/env | grep HOSTNAME
  "HOSTNAME": "guestbook-44sep",

> (for i in `seq 20`; do curl -s http://${ip}:3000/env | grep HOSTNAME; done)  | sort | unic
  "HOSTNAME": "guestbook-44sep",
  "HOSTNAME": "guestbook-hum5n",
  "HOSTNAME": "guestbook-ppm40",
```

The load balancer correctly targets all the pods, even though they are in multiple zones.

Shutting down the cluster

When you're done, clean up:

GCE:

```
KUBERNETES_PROVIDER=gce KUBE_USE_EXISTING_MASTER=true KUBE_GCE_ZONE=us-central1-f kubernetes
KUBERNETES_PROVIDER=gce KUBE_USE_EXISTING_MASTER=true KUBE_GCE_ZONE=us-central1-b kubernetes
KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-central1-a kubernetes/cluster/kube-down.sh
```

AWS:

```
KUBERNETES_PROVIDER=aws KUBE_USE_EXISTING_MASTER=true KUBE_AWS_ZONE=us-west-2c kubernetes/cl
KUBERNETES_PROVIDER=aws KUBE_USE_EXISTING_MASTER=true KUBE_AWS_ZONE=us-west-2b kubernetes/cl
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2a kubernetes/cluster/kube-down.sh
```

Create an Issue Edit this Page

Edit This Page

Validate Node Setup

Node Conformance Test

Node conformance test is a containerized test framework that provides a system verification and functionality test for a node. The test validates whether the node meets the minimum requirements for Kubernetes; a node that passes the test is qualified to join a Kubernetes cluster.

Limitations

In Kubernetes version 1.5, node conformance test has the following limitations:

- Node conformance test only supports Docker as the container runtime.

Node Prerequisite

To run node conformance test, a node must satisfy the same prerequisites as a standard Kubernetes node. At a minimum, the node should have the following daemons installed:

- Container Runtime (Docker)
- Kubelet

Running Node Conformance Test

To run the node conformance test, perform the following steps:

1. Point your Kubelet to localhost `--api-servers="http://localhost:8080"`, because the test framework starts a local master to test Kubelet. There are some other Kubelet flags you may care:

   - `--pod-cidr`: If you are using `kubenet`, you should specify an arbitrary CIDR to Kubelet, for example `--pod-cidr=10.180.0.0/24`.
   - `--cloud-provider`: If you are using `--cloud-provider=gce`, you should remove the flag to run the test.

2. Run the node conformance test with command:

```
# $CONFIG_DIR is the pod manifest path of your Kubelet.
# $LOG_DIR is the test output path.
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/result \
  k8s.gcr.io/node-test:0.2
```

Running Node Conformance Test for Other Architectures

Kubernetes also provides node conformance test docker images for other architectures:

Arch	Image
amd64	node-test-amd64
arm	node-test-arm
arm64	node-test-arm64

Running Selected Test

To run specific tests, overwrite the environment variable `FOCUS` with the regular expression of tests you want to run.

```
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs:ro -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/result \
  -e FOCUS=MirrorPod \ # Only run MirrorPod test
  k8s.gcr.io/node-test:0.2
```

To skip specific tests, overwrite the environment variable `SKIP` with the regular expression of tests you want to skip.

```
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs:ro -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/result \
  -e SKIP=MirrorPod \ # Run all conformance tests but skip MirrorPod test
  k8s.gcr.io/node-test:0.2
```

Node conformance test is a containerized version of node e2e test. By default, it runs all conformance tests.

Theoretically, you can run any node e2e test if you configure the container and mount required volumes properly. But **it is strongly recommended to only run conformance test**, because it requires much more complex configuration to run non-conformance test.

Caveats

- The test leaves some docker images on the node, including the node conformance test image and images of containers used in the functionality test.
- The test leaves dead containers on the node. These containers are created during the functionality test.

Create an Issue Edit this Page