Edit This Page

Setup

Use this page to find the type of solution that best fits your needs.

Deciding where to run Kubernetes depends on what resources you have available and how much flexibility you need. You can run Kubernetes almost anywhere, from your laptop to VMs on a cloud provider to a rack of bare metal servers. You can also set up a fully-managed cluster by running a single command or craft your own customized cluster on your bare metal servers.

- Local-machine Solutions
- Hosted Solutions
- Turnkey – Cloud Solutions
- Turnkey – On-Premises Solutions
- Custom Solutions
- What's next

Local-machine Solutions

A local-machine solution is an easy way to get started with Kubernetes. You can create and test Kubernetes clusters without worrying about consuming cloud resources and quotas.

You should pick a local solution if you want to:

- Try or start learning about Kubernetes
- Develop and test clusters locally

Pick a local-machine solution.

Hosted Solutions

Hosted solutions are a convenient way to create and maintain Kubernetes clusters. They manage and operate your clusters so you don't have to.

You should pick a hosted solution if you:

- Want a fully-managed solution
- Want to focus on developing your apps or services

- Don't have dedicated site reliability engineering (SRE) team but want high availability
- Don't have resources to host and monitor your clusters

Pick a hosted solution.

Turnkey – Cloud Solutions

These solutions allow you to create Kubernetes clusters with only a few commands and are actively developed and have active community support. They can also be hosted on a range of Cloud IaaS providers, but they offer more freedom and flexibility in exchange for effort.

You should pick a turnkey cloud solution if you:

- Want more control over your clusters than the hosted solutions allow
- Want to take on more operations ownership

Pick a turnkey cloud solution

Turnkey – On-Premises Solutions

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

You should pick a on-prem turnkey cloud solution if you:

- Want to deploy clusters on your private cloud network
- Have a dedicated SRE team
- Have the resources to host and monitor your clusters

Pick an on-prem turnkey cloud solution.

Custom Solutions

Custom solutions give you the most freedom over your clusters but require the most expertise. These solutions range from bare-metal to cloud providers on different operating systems.

Pick a custom solution.

What's next

Go to Picking the Right Solution for a complete list of solutions.

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
- Table of Solutions

Local-machine Solutions

- Minikube is a 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.

- microk8s provides a single command installation of the latest Kubernetes release on a local machine for development and testing. Setup is quick, fast (~30 sec) and supports many plugins including Istio with a single command.

- 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.

- 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.

Hosted Solutions

- AppsCode.com provides managed Kubernetes clusters for various public clouds, including AWS and Google Cloud Platform.
- APPUiO runs an OpenShift public cloud platform, supporting any Kubernetes workload. Additionally APPUiO offers Private Managed OpenShift Clusters, running on any public or private cloud.
- Amazon Elastic Container Service for Kubernetes offers managed Kubernetes service.
- Azure Kubernetes Service offers managed Kubernetes clusters.
- Giant Swarm offers managed Kubernetes clusters in their own datacenter, on-premises, or on public clouds.
- Google Kubernetes Engine offers managed Kubernetes clusters.
- IBM Cloud Kubernetes Service offers managed Kubernetes clusters with isolation choice, operational tools, integrated security insight into images and containers, and integration with Watson, IoT, and data.
- Kubermatic provides managed Kubernetes clusters for various public clouds, including AWS and Digital Ocean, as well as on-premises with OpenStack integration.
- Kublr offers enterprise-grade secure, scalable, highly reliable Kubernetes clusters on AWS, Azure, GCP, and on-premise. It includes out-of-the-box backup and disaster recovery, multi-cluster centralized logging and monitoring, and built-in alerting.
- 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.
- OpenShift Dedicated offers managed Kubernetes clusters powered by OpenShift.
- OpenShift Online provides free hosted access for Kubernetes applications.

- 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.
- 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.)
- Stackpoint.io provides Kubernetes infrastructure automation and management for multiple public clouds.
- VMware Cloud PKS is an enterprise Kubernetes-as-a-Service offering in the VMware Cloud Services portfolio that provides easy to use, secure by default, cost effective, SaaS-based Kubernetes clusters.

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.

- Agile Stacks
- Alibaba Cloud
- APPUiO
- AWS
- Azure
- CenturyLink Cloud
- Conjure-up Kubernetes with Ubuntu on AWS, Azure, Google Cloud, Oracle Cloud
- Gardener
- Google Compute Engine (GCE)
- IBM Cloud
- Kontena Pharos
- Kubermatic
- Kublr
- Madcore.Ai
- Oracle Container Engine for K8s
- Pivotal Container Service
- Giant Swarm
- Rancher 2.0
- Stackpoint.io
- Tectonic by CoreOS

On-Premises turnkey cloud solutions

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

- Agile Stacks
- APPUiO
- GKE On-Prem | Google Cloud
- IBM Cloud Private
- Kontena Pharos
- Kubermatic
- Kublr
- Pivotal Container Service
- Giant Swarm
- Rancher 2.0
- SUSE CaaS Platform
- SUSE Cloud Application Platform

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
- Gardener
- Kublr

- Kubernetes on Ubuntu
- Kubespray
- Rancher Kubernetes Engine (RKE)

On-Premises VMs

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

Bare Metal

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

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	Networkin
any	any	multi-support	any CNI
Google Kubernetes Engine			GCE
Stackpoint.io		multi-support	multi-sup
AppsCode.com	Saltstack	Debian	multi-sup
Madcore.Ai	Jenkins DSL	Ubuntu	flannel

IaaS Provider	Config. Mgmt.	OS	Networkin
Platform9		multi-support	multi-sup
Kublr	custom	multi-support	multi-sup
Kubermatic		multi-support	multi-sup
IBM Cloud Kubernetes Service		Ubuntu	IBM Clou
Giant Swarm		CoreOS	flannel an
GCE	Saltstack	Debian	GCE
Azure Kubernetes Service		Ubuntu	Azure
Azure (IaaS)		Ubuntu	Azure
Bare-metal	custom	Fedora	*none*
Bare-metal	custom	Fedora	flannel
libvirt	custom	Fedora	flannel
KVM	custom	Fedora	flannel
DCOS	Marathon	CoreOS/Alpine	custom
AWS	CoreOS	CoreOS	flannel
GCE	CoreOS	CoreOS	flannel
Vagrant	CoreOS	CoreOS	flannel
CloudStack	Ansible	CoreOS	flannel
VMware vSphere	any	multi-support	multi-sup
Bare-metal	custom	CentOS	flannel
lxd	Juju	Ubuntu	flannel/ca
AWS	Juju	Ubuntu	flannel/ca
Azure	Juju	Ubuntu	flannel/ca
GCE	Juju	Ubuntu	flannel/ca
Oracle Cloud	Juju	Ubuntu	flannel/ca
Rackspace	Juju	Ubuntu	flannel/ca
VMware vSphere	Juju	Ubuntu	flannel/ca
Bare Metal	Juju	Ubuntu	flannel/ca
AWS	Saltstack	Debian	AWS
AWS	kops	Debian	AWS
Bare-metal	custom	Ubuntu	flannel
oVirt			
any	any	any	any
any	any	any	any
any	RKE	multi-support	flannel or
any	Gardener Cluster-Operator	multi-support	multi-sup
Alibaba Cloud Container Service For Kubernetes	ROS	CentOS	flannel/Te
Agile Stacks	Terraform	CoreOS	multi-sup
IBM Cloud Kubernetes Service		Ubuntu	calico
Digital Rebar	kubeadm	any	metal
VMware Cloud PKS		Photon OS	Canal

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.

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.

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.

Edit This Page

v1.12 Release Notes

- v1.12.0-rc.2
  - Downloads for v1.12.0-rc.2
    - Client Binaries
    - Server Binaries
    - Node Binaries
  - Changelog since v1.12.0-rc.1
    - Other notable changes
- v1.12.0-rc.1

v1.12.0-rc.2

Documentation & Examples

Downloads for v1.12.0-rc.2

filename	sha256 hash
kubernetes.tar.gz	`184ea437bc72d0e6a4c96b964de53181273e919a1d4785515da3406c7e982bf5`
kubernetes-src.tar.gz	`aee82938827ef05ab0ee81bac42f4f79fff126294469868d02efb3426717d71e`

Client Binaries

filename	sha256 hash
kubernetes-client-darwin-386.tar.gz	`40ed3ef9bbc4fad7787dd14eae952edf06d40e1094604bc6d10209b8`
kubernetes-client-darwin-amd64.tar.gz	`a317fe3801ea5387ce474b9759a7e28ede8324587f79935a7a945da4`

filename	sha256 hash
kubernetes-client-linux-386.tar.gz	`cd61b4b71d6b739582c02b5be1d87d928507bc59f64ee72629a920cc`
kubernetes-client-linux-amd64.tar.gz	`306af04fc18ca2588e16fd831358df50a2cb02219687b543073836f8`
kubernetes-client-linux-arm.tar.gz	`497584f2686339cce857cff1ebf4ed10dcd63f4684a03c242b0828fc`
kubernetes-client-linux-arm64.tar.gz	`1dfbb8c299f5af15239ef39135a6c8a52ee4c234764ee0437d8f707e`
kubernetes-client-linux-ppc64le.tar.gz	`668d6f35c5f6adcd25584d9ef74c549db13ffca9d93b4bc8d25609a8`
kubernetes-client-linux-s390x.tar.gz	`8a8e205c38858bd9d161115e5e2870c6cfc9c82e189d156e7062e6fa`
kubernetes-client-windows-386.tar.gz	`cdef48279c22cc8c764e43a4b9c2a86f02f21c80abbbcd48041fb1e8`
kubernetes-client-windows-amd64.tar.gz	`50621a3d2b1550c69325422c6dce78f5690574b35d3778dd3afcf698`

Server Binaries

filename	sha256 hash
kubernetes-server-linux-amd64.tar.gz	`87a8438887a2daa199508aae591b158025860b8381c64cbe9b1d0c06c4`
kubernetes-server-linux-arm.tar.gz	`f65be73870a0e564ef8ce1b6bb2b75ff7021a6807de84b5750e4fa7863`
kubernetes-server-linux-arm64.tar.gz	`171f15aa8b7c365f4fee70ce025c882a921d0075bd726a99b5534cadd0`
kubernetes-server-linux-ppc64le.tar.gz	`abc2003d58bd1aca517415c582ed1e8bb1ed596bf04197f4fc7c0c5186`
kubernetes-server-linux-s390x.tar.gz	`e2ce834abb4d45d91fd7a8d774e47f0f8092eb4edcf556605c2ef6e2b1`

Node Binaries

filename	sha256 hash
kubernetes-node-linux-amd64.tar.gz	`6016c3a1e14c42dcc88caed6497de1b2c56a02bb52d836b19e2ff5209`
kubernetes-node-linux-arm.tar.gz	`e712e38c8037159ea074ad93c2f2905cf279f3f119e5fdbf9b9739103`
kubernetes-node-linux-arm64.tar.gz	`7f4095f12d8ad9438919fa447360113799f88bb9435369b9307a41dd9`
kubernetes-node-linux-ppc64le.tar.gz	`4aeb5dbb0c68e54570542eb5a1d7506d73c81b57eba3c2080ee73bb53`
kubernetes-node-linux-s390x.tar.gz	`a160599598167208286db6dc73b415952836218d967fa964fc432b213`
kubernetes-node-windows-amd64.tar.gz	`174bedf62b7959d4cb1b1595666f607cd6377c7a2e2208fef5bd55460`

Changelog since v1.12.0-rc.1

Other notable changes

- Update to use manifest list for etcd image (#68896, @ixdy)
- Fix Azure nodes power state for InstanceShutdownByProviderID() (#68921, @feiskyer)
- Bump kube-dns to 1.14.13 (#68900, @MrHohn)
  - - Update Alpine base image to 3.8.1.
  - - Build multi-arch images correctly.
- kubelet: fix grpc timeout in the CRI client (#67793, @fisherxu)

- Update to golang 1.10.4 (#68802, @ixdy)
- kubeadm now uses fat manifests for the kube-dns images (#68830, @rosti)
- Update Cluster Autoscaler version to 1.12.0. (#68739, @losipiuk)
  - See https://github.com/kubernetes/autoscaler/releases/tag/1.12.0 for CA release notes.
- kube-proxy restores the *filter table when running in ipvs mode. (#68786, @alexjx)
- New kubeDNS image fixes an issue where SRV records were incorrectly being compressed. Added manifest file for multiple arch images. (#68430, @prameshj)
- Drain should delete terminal pods. (#68767, @ravisantoshgudimetla)

v1.12.0-rc.1

Documentation & Examples

Downloads for v1.12.0-rc.1

filename	sha256 hash
kubernetes.tar.gz	`ac65cf9571c3a03105f373db23c8d7f4d01fe1c9ee09b06615bb02d0b81d572c`
kubernetes-src.tar.gz	`28518e1d9c7fe5c54aa3b57235ac8d1a7dae02aec04177c38ca157fc2d16edb6`

Client Binaries

filename	sha256 hash
kubernetes-client-darwin-386.tar.gz	`7b6f6f264464d40b7975baecdd796d4f75c5a305999b4ae1f4513646`
kubernetes-client-darwin-amd64.tar.gz	`5feabe3e616125a36ce4c8021d6bdccdec0f3d82f151b80af7cac145`
kubernetes-client-linux-386.tar.gz	`40524a1a09dd24081b3494593a02a461227727f8706077542f2b8603`
kubernetes-client-linux-amd64.tar.gz	`ac2c9757d7df761bdf8ffc259fff07448c300dd110c7dbe2ae383019`
kubernetes-client-linux-arm.tar.gz	`02f27ae16e8ebb12b3cb66391fe85f64de08a99450d726e9defd2c5b`
kubernetes-client-linux-arm64.tar.gz	`1286af2cad3f8e2ee8e2dc18a738935779631b58e7ef3da8794bbead`
kubernetes-client-linux-ppc64le.tar.gz	`9c04419b159fb0fe501d6e0c8122d6a80b5d6961070ebc5e759f4327`
kubernetes-client-linux-s390x.tar.gz	`104d5c695826971c64cb0cec26cf791d609d3e831edb33574e9af2c4`
kubernetes-client-windows-386.tar.gz	`0096f8126eb04eafa9decd258f6d09977d24eee91b83781347a34ebb`
kubernetes-client-windows-amd64.tar.gz	`a641a1a421795279a6213163d7becab9dc6014362e6566f13d660ef1`

Server Binaries

filename	sha256 hash
kubernetes-server-linux-amd64.tar.gz	`202958d3cfb774fd065ad1ec2477dc9c92ce7f0ff355807c9a2a3a61e8`
kubernetes-server-linux-arm.tar.gz	`474de8f6a58d51eb01f6cc73b41897351528a839f818d5c4f828a484f8`
kubernetes-server-linux-arm64.tar.gz	`dbd5affd244815bf45ac0c7a56265800864db623a6a37e7ce9ebe5e589`
kubernetes-server-linux-ppc64le.tar.gz	`a62fefa8ad7b3fbfeb7702dac7d4d6f37823b6c3e4edae3356bf0781b4`
kubernetes-server-linux-s390x.tar.gz	`0f77690f87503c8ee7ccb473c9d2b9d26420292defd82249509cf50d8b`

Node Binaries

filename	sha256 hash
kubernetes-node-linux-amd64.tar.gz	`2191845147d5aab08f14312867f86078b513b6aff8685bb8ce84a06b7`
kubernetes-node-linux-arm.tar.gz	`54de98d7d2a71b78bc7a45e70a2005144d210401663f5a9daadedd05f`
kubernetes-node-linux-arm64.tar.gz	`a765514e0c4865bb20ceb476af83b9d9356c9b565cfe12615ecf7ad3c`
kubernetes-node-linux-ppc64le.tar.gz	`b7ae7d159602d0b933614071f11216ede4df3fc2b28a30d0018e06b3b`
kubernetes-node-linux-s390x.tar.gz	`7d4f502eda6aa70b7a18420344abfaec740d74a1edffcb9869e4305c2`
kubernetes-node-windows-amd64.tar.gz	`ed5516b1f66a39592a101bec135022b3905a66ae526b8ed3e2e9dff5e`

Changelog since v1.12.0-beta.2

Action Required

- Service events are now added in azure-cloud-provider for easily identify the underground errors of Azure API. (#68212, @feiskyer)
  - Action required: The following clusterrole and clusterrolebinding should be applied:
  - kind: List
  - apiVersion: v1
  - items:
  - - apiVersion: rbac.authorization.k8s.io/v1
  - kind: ClusterRole
  - metadata:
  - labels:
  - kubernetes.io/cluster-service: "true"
  - name: system:azure-cloud-provider
  - rules:
  - - apiGroups: [""]
  - resources: ["events"]
  - verbs:
  - - create
  - - patch
  - - update
  - - apiVersion: rbac.authorization.k8s.io/v1

– kind: ClusterRoleBinding
– metadata:
– labels:
– kubernetes.io/cluster-service: “true”
– name: system:azure-cloud-provider
– roleRef:
– apiGroup: rbac.authorization.k8s.io
– kind: ClusterRole
– name: system:azure-cloud-provider
– subjects:
– - kind: ServiceAccount
– name: azure-cloud-provider
– namespace: kube-system
– If the clusterrole with same has already been provisioned (e.g. for accessing azurefile secrets), then the above yaml should be merged togather, e.g.
– kind: List
– apiVersion: v1
– items:
– - apiVersion: rbac.authorization.k8s.io/v1
– kind: ClusterRole
– metadata:
– labels:
– kubernetes.io/cluster-service: “true”
– name: system:azure-cloud-provider
– rules:
– - apiGroups: [“”]
– resources: [“events”]
– verbs:
– - create
– - patch
– - update
– - apiGroups: [“”]
– resources: [“secrets”]
– verbs:
– - get
– - create
– - apiVersion: rbac.authorization.k8s.io/v1
– kind: ClusterRoleBinding
– metadata:
– labels:
– kubernetes.io/cluster-service: “true”
– name: system:azure-cloud-provider
– roleRef:
– apiGroup: rbac.authorization.k8s.io
– kind: ClusterRole

  - name: system:azure-cloud-provider
  - subjects:
  - - kind: ServiceAccount
  - name: azure-cloud-provider
  - namespace: kube-system
  - - kind: ServiceAccount
  - name: persistent-volume-binder
  - namespace: kube-system

Other notable changes

- Update metrics-server to v0.3.1 (#68746, @DirectXMan12)
- Upgrade kubeadm's version of docker support (#68495, @yuansisi)
- fix a bug that overwhelming number of prometheus metrics are generated because $NAMESPACE is not replaced by string "{namespace}" (#68530, @wenjiaswe)
- The feature gates `ReadOnlyAPIDataVolumes` and `ServiceProxyAllowExternalIPs`, deprecated since 1.10, have been removed and any references must be removed from command-line invocations. (#67951, @liggitt)
- Verify invalid secret/configmap/projected volumes before calling setup (#68691, @gnufied)
- Fix bug that caused `kubectl` commands to sometimes fail to refresh access token when running against GKE clusters. (#66314, @jlowdermilk)
- Use KubeDNS by default in GCE setups, as CoreDNS has significantly higher memory usage in large clusters. (#68629, @shyamjvs)
- Fix PodAntiAffinity issues in case of multiple affinityTerms. (#68173, @Huang-Wei)
- Make APIGroup field in TypedLocalObjectReference optional. (#68419, @xing-yang)
- Fix potential panic when getting azure load balancer status (#68609, @feiskyer)
- Fix kubelet panics when RuntimeClass is enabled. (#68521, @yujuhong)
- - cAdvisor: Fix NVML initialization race condition (#68431, @dashpole)
  - - cAdvisor: Fix brtfs filesystem discovery
  - - cAdvisor: Fix race condition with AllDockerContainers
  - - cAdvisor: Don't watch .mount cgroups
  - - cAdvisor: Reduce lock contention during list containers
- Promote ScheduleDaemonSetPods by default scheduler to beta (#67899, @ravisantoshgudimetla)

v1.12.0-beta.2

Documentation & Examples

Downloads for v1.12.0-beta.2

filename	sha256 hash
kubernetes.tar.gz	`7163d18b9c1bd98ce804b17469ed67b399deb7b574dd12a86609fc647c5c773b`
kubernetes-src.tar.gz	`6225b71b2dec0f29afb713e64d2b6b82bd0e122274c31310c0de19ef023cb1d0`

Client Binaries

filename	sha256 hash
kubernetes-client-darwin-386.tar.gz	`f2ec9799e47c28fce336bc90a6e9b4e47def7081fd73b8e2164940f0`
kubernetes-client-darwin-amd64.tar.gz	`0e8cfcbe5ec862423ced97da1d9740d4cc4904a0d5cd11a60616aee5`
kubernetes-client-linux-386.tar.gz	`1cbd6e8dd892cfc2555d37e733b66aaf85df9950466c7295875d312a`
kubernetes-client-linux-amd64.tar.gz	`47337b58a26a4953e5c061d28e3ec89b3d4354bce40f9b51fbe26959`
kubernetes-client-linux-arm.tar.gz	`eaaed82f428fb7ddbb10b4e39a2f287817c33ae24ff16008159f437a`
kubernetes-client-linux-arm64.tar.gz	`3249d1c7d5d5500793546eb144fe537d1984a01c7a79c1382eb2e26a`
kubernetes-client-linux-ppc64le.tar.gz	`67afd34f2199deff901b0872a177dc448ba700dc4ced9ede6f3187a0`
kubernetes-client-linux-s390x.tar.gz	`e8faa6e45c6e2aeb67ac65737e09be87c190e3c89782ec87a9a205d4`
kubernetes-client-windows-386.tar.gz	`2395051c8cbd0a995b5f3689c0f8c0447bcc1c46440d8cdeffd7c7fc`
kubernetes-client-windows-amd64.tar.gz	`c6a38ee6eda20656b391ecfcc1f24505eb8a3a5a3200d4bddede3182`

Server Binaries

filename	sha256 hash
kubernetes-server-linux-amd64.tar.gz	`795c713a91118218f5952e1bd4cf0933f36476aa3d9d60a9ee43c9bae8`
kubernetes-server-linux-arm.tar.gz	`1798d48a37b8f06878e0ecb8d9b67d0fb5c8ee721608412add57725eb5`
kubernetes-server-linux-arm64.tar.gz	`da2459b5e811daaa2fc04a072773e81dc220400f3aeb6e29bb9594c306`
kubernetes-server-linux-ppc64le.tar.gz	`7fd1c2ba0c2c9da5db54f8d0aed28261f03e9953ce01fa367e4ce3d84b`
kubernetes-server-linux-s390x.tar.gz	`c9fafb009d7e5da74f588aaa935244c452de52b9488863b90e8b477b1b`

Node Binaries

filename	sha256 hash
kubernetes-node-linux-amd64.tar.gz	`ab901137b499829b20b868492d04c1f69d738620b96eb349c642d6d77`
kubernetes-node-linux-arm.tar.gz	`116dd82721f200f3f37df0e47aebb611fdd7856f94d4c2ebb1d51db21`
kubernetes-node-linux-arm64.tar.gz	`56d8316eb95f7f54c154625063617b86ffb8e2cc80b8225cce4f5c91c`
kubernetes-node-linux-ppc64le.tar.gz	`66535b16ad588ba3bfcb40728a0497c6821360ab7be9c3ced2072bfa1`
kubernetes-node-linux-s390x.tar.gz	`688e09becc9327e50c68b33161eac63a8ba018c02fb298cbd0de82d6e`
kubernetes-node-windows-amd64.tar.gz	`b72582f67d19c06f605ca9b02c08b7227796c15c639e3c09b06a8b667`

Changelog since v1.12.0-beta.1

Action Required

- Action required: The –storage-versions flag of kube-apiserver is deprecated. Please omit this flag to ensure the default storage versions are used. Otherwise the cluster is not safe to upgrade to a version newer than 1.12. This flag will be removed in 1.13. (#68080, @caesarxuchao)

Other notable changes

- kubeadm: add mandatory "–config" flag to "kubeadm alpha phase preflight" (#68446, @neolit123)
- Apply user configurations for local etcd (#68334, @SataQiu)
- kubeadm: added phase command "alpha phase kubelet config annotate-cri" (#68449, @fabriziopandini)
- If `TaintNodesByCondition` is enabled, add `node.kubernetes.io/unschedulable` and (#64954, @k82cn)
  - `node.kubernetes.io/network-unavailable` automatically to DaemonSet pods.
- Deprecate cloudstack and ovirt controllers (#68199, @dims)
- add missing LastTransitionTime of ContainerReady condition (#64867, @dixudx)
- kube-controller-manager: use informer cache instead of active pod gets in HPA controller (#68241, @krzysztof-jastrzebski)
- Support NodeShutdown taint for azure (#68033, @yastij)
- Registers volume topology information reported by a node-level Container Storage Interface (CSI) driver. This enables Kubernetes support of CSI topology mechanisms. (#67684, @verult)
- Update default etcd server to 3.2.24 for kubernetes 1.12 (#68318, @timothysc)
- External CAs can now be used for kubeadm with only a certificate, as long as all required certificates already exist. (#68296, @liztio)
- Bump addon-manager to v8.7 (#68299, @MrHohn)
  - - Support extra `--prune-whitelist` resources in kube-addon-manager.
  - - Update kubectl to v1.10.7.
- Let service controller retry creating load balancer when persistUpdate failed due to conflict. (#68087, @grayluck)
- Kubelet now only sync iptables on Linux. (#67690, @feiskyer)
- CSI NodePublish call can optionally contain information about the pod that requested the CSI volume. (#67945, @jsafrane)
- [e2e] verifying LimitRange update is effective before creating new pod (#68171, @dixudx)

- cluster/gce: generate consistent key sizes in config-default.sh using /dev/urandom instead of /dev/random (#67139, @yogi-sagar)
- Add support for volume attach limits for CSI volumes (#67731, @gnufied)
- CSI volume plugin does not need external attacher for non-attachable CSI volumes. (#67955, @jsafrane)
- KubeletPluginsWatcher feature graduates to beta. (#68200, @RenaudWasTaken)
- Update etcd client to 3.2.24 for latest release (#68147, @timothysc)
- [fluentd-gcp-scaler addon] Bump fluentd-gcp-scaler to 0.4 to pick up security fixes. (#67691, @loburm)
  - [prometheus-to-sd addon] Bump prometheus-to-sd to 0.3.1 to pick up security fixes, bug fixes and new features.
  - [event-exporter addon] Bump event-exporter to 0.2.3 to pick up security fixes.
- Fixes issue where pod scheduling may fail when using local PVs and pod affinity and anti-affinity without the default StatefulSet OrderedReady pod management policy (#67556, @msau42)
- Kubelet only applies default hard evictions of nodefs.inodesFree on Linux (#67709, @feiskyer)
- Add kubelet stats for windows system container "pods" (#66427, @feiskyer)
- Add a TTL machenism to clean up Jobs after they finish. (#66840, @janetkuo)

v1.12.0-beta.1

Documentation & Examples

Downloads for v1.12.0-beta.1

filename	sha256 hash
kubernetes.tar.gz	`caa332b14a6ea9d24710e3b015a91b62c04cab14bed14c49077e08bd82b8f4c1`
kubernetes-src.tar.gz	`821bdea3a52a348306fa8226bcfffa67b375cf1dd80e4be343ce0b38dd20a9a0`

Client Binaries

filename	sha256 hash
kubernetes-client-darwin-386.tar.gz	`58323c0a81afe53dd0dda1c6eb513caa4c82514fb6c7f0a327242e57`
kubernetes-client-darwin-amd64.tar.gz	`28e9344ede16890ea7848c261e461ded89c3bb2dd5b08446da04b071`
kubernetes-client-linux-386.tar.gz	`a9eece5e0994d2ad5e07152d88787a8b5e9efcdf78983a5bafe3699e`
kubernetes-client-linux-amd64.tar.gz	`9a67750cc4243335f0c2eb89db1c4b54b0a8af08c59e2041636d0a3e`

filename	sha256 hash
kubernetes-client-linux-arm.tar.gz	bbd2644f843917a3de517a53c90b327502b577fe533a9ad3da4fe6bc
kubernetes-client-linux-arm64.tar.gz	630946f49ef18dd43c004d99dccd9ae76390281f54740d7335c042f6
kubernetes-client-linux-ppc64le.tar.gz	1d4e5cd83faf4cae8e16667576492fcd48a72f69e8fd89d599a8b555
kubernetes-client-linux-s390x.tar.gz	9cefdcf21a62075b5238fda8ef2db08f81b0541ebce0e67353af1dde
kubernetes-client-windows-386.tar.gz	8b0085606ff38bded362bbe4826b5c8ee5199a33d5cbbc1b9b58f133
kubernetes-client-windows-amd64.tar.gz	f44a3ec55dc7d926e681c33b5f7830c6d1cb165e24e349e426c1089b

Server Binaries

filename	sha256 hash
kubernetes-server-linux-amd64.tar.gz	1bf7364aa168fc251768bc850d66fef1d93f324f0ec85f6dce7408062
kubernetes-server-linux-arm.tar.gz	dadc94fc0564cfa98add5287763bbe9c33bf8ba3eebad95fb2258c33fe
kubernetes-server-linux-arm64.tar.gz	2e6c8a7810705594f191b33476bf4c8fca8cebb364f0855dfea577b01f
kubernetes-server-linux-ppc64le.tar.gz	ced4a0a4e03639378eff0d3b8bfb832f5fb96be8df3e0befbdbd71373a
kubernetes-server-linux-s390x.tar.gz	7e1a3fac2115c15b5baa0db04c7f319fbaaca92aa4c4588ecf62fb1981

Node Binaries

filename	sha256 hash
kubernetes-node-linux-amd64.tar.gz	81d2e2f4cd3254dd345c1e921b12bff62eb96e7551336c44fb0da540
kubernetes-node-linux-arm.tar.gz	b14734a20190aca2b2af9cee59549d285be4f0c38faf89c5308c9453
kubernetes-node-linux-arm64.tar.gz	ad0a81ecf6ef8346b7aa98a8d02a4f3853d0a5439d149a14b1ac2307
kubernetes-node-linux-ppc64le.tar.gz	8e6d72837fe19afd055786c8731bd555fe082e107195c956c6985e56
kubernetes-node-linux-s390x.tar.gz	0fc7d55fb2750b29c0bbc36da050c8bf14508b1aa40e38e3b7f6cf31
kubernetes-node-windows-amd64.tar.gz	09bf133156b9bc474d272bf16e765b143439959a1f007283c477e799

Changelog since v1.12.0-alpha.1

Action Required

- Move volume dynamic provisioning scheduling to beta (ACTION REQUIRED: The DynamicProvisioningScheduling alpha feature gate has been removed. The VolumeScheduling beta feature gate is still required for this feature) (#67432, @lichuqiang)

Other notable changes

- Not split nodes when searching for nodes but doing it all at once. (#67555, @wgliang)
- Deprecate kubectl run generators, except for run-pod/v1 (#68132, @soltysh)
- Using the Horizontal Pod Autoscaler with metrics from Heapster is now deprecated. (#68089, @DirectXMan12)
- Support both directory and block device for local volume plugin FileSystem VolumeMode (#63011, @NickrenREN)
- Add CSI volume attributes for kubectl describe pv. (#65074, @wgliang)
- `kubectl rollout status` now works for unlimited timeouts. (#67817, @tnozicka)
- Fix panic when processing Azure HTTP response. (#68210, @feiskyer)
- add mixed protocol support for azure load balancer (#67986, @andyzhangx)
- Replace scale down forbidden window with scale down stabilization window. Rather than waiting a fixed period of time between scale downs HPA now scales down to the highest recommendation it during the scale down stabilization window. (#68122, @krzysztof-jastrzebski)
- Adding validation to kube-scheduler at the API level (#66799, @noqcks)
- Improve performance of Pod affinity/anti-affinity in the scheduler (#67788, @ahmad-diaa)
- kubeadm: fix air-gapped support and also allow some kubeadm commands to work without an available networking interface (#67397, @neolit123)
- Increase Horizontal Pod Autoscaler default update interval (30s -> 15s). It will improve HPA reaction time for metric changes. (#68021, @krzysztof-jastrzebski)
- Increase scrape frequency of metrics-server to 30s (#68127, @serathius)
- Add new `--server-dry-run` flag to `kubectl apply` so that the request will be sent to the server with the dry-run flag (alpha), which means that changes won't be persisted. (#68069, @apelisse)
- kubelet v1beta1 external ComponentConfig types are now available in the `k8s.io/kubelet` repo (#67263, @luxas)
- Adds a kubelet parameter and config option to change CFS quota period from the default 100ms to some other value between 1µs and 1s. This was done to improve response latencies for workloads running in clusters with guaranteed and burstable QoS classes. (#63437, @szuecs)
- Enable secure serving on port 10258 to cloud-controller-manager (configurable via `--secure-port`). Delegated authentication and authorization have to be configured like for aggregated API servers. (#67069, @sttts)
- Support extra `--prune-whitelist` resources in kube-addon-manager. (#67743, @Random-Liu)
- Upon receiving a LIST request with expired continue token, the apiserver now returns a continue token together with the 410 "the from parameter

is too old " error. If the client does not care about getting a list from a consistent snapshot, the client can use this token to continue listing from the next key, but the returned chunk will be from the latest snapshot. (#67284, @caesarxuchao)

- Role, ClusterRole and their bindings for cloud-provider is put under system namespace. Their addonmanager mode switches to EnsureExists. (#67224, @grayluck)
- Mount propagation has promoted to GA. The `MountPropagation` feature gate is deprecated and will be removed in 1.13. (#67255, @bertinatto)
- Introduce CSI Cluster Registration mechanism to ease CSI plugin discovery and allow CSI drivers to customize Kubernetes' interaction with them. (#67803, @saad-ali)
- Adds the commands `kubeadm alpha phases renew <cert-name>` (#67910, @liztio)
- ProcMount added to SecurityContext and AllowedProcMounts added to PodSecurityPolicy to allow paths in the container's /proc to not be masked. (#64283, @jessfraz)
- support cross resource group for azure file (#68117, @andyzhangx)
- Port 31337 will be used by fluentd (#68051, @Szetty)
- Improve CPU sample sanitization in HPA by taking metric's freshness into account. (#68068, @krzysztof-jastrzebski)
- CoreDNS is now v1.2.2 for Kubernetes 1.12 (#68076, @rajansandeep)
- Enable secure serving on port 10257 to kube-controller-manager (configurable via `--secure-port`). Delegated authentication and authorization have to be configured like for aggregated API servers. (#64149, @sttts)
- Update metrics-server to v0.3.0. (#68077, @DirectXMan12)
- TokenRequest and TokenRequestProjection are now beta features. To enable these feature, the API server needs to be started with the following flags: (#67349, @mikedanese) * –service-account-issuer * –service-account-signing-key-file * –service-account-api-audiences
- Don't let aggregated apiservers fail to launch if the external-apiserver-authentication configmap is not found in the cluster. (#67836, @sttts)
- Promote AdvancedAuditing to GA, replacing the previous (legacy) audit logging mechanisms. (#65862, @loburm)
- Azure cloud provider now supports unmanaged nodes (such as on-prem) that are labeled with `kubernetes.azure.com/managed=false` and `alpha.service-controller.kubernetes.io/exclude-balancer=true` (#67984, @feiskyer)
- `kubectl get apiservice` now shows the target service and whether the service is available (#67747, @smarterclayton)
- Openstack supports now node shutdown taint. Taint is added when instance is shutdown in openstack. (#67982, @zetaab)
- Return apiserver panics as 500 errors instead terminating the apiserver process. (#68001, @sttts)
- Fix VMWare VM freezing bug by reverting #51066 (#67825, @nikopen)
- Make CoreDNS be the default DNS server in kube-up (instead of kube-dns

formerly). (#67569, @fturib)
  - It is still possible to deploy kube-dns by setting CLUSTER_DNS_CORE_DNS=false.
- Added support to restore a volume from a volume snapshot data source. (#67087, @xing-yang)
- fixes the errors/warnings in fluentd configuration (#67947, @saravanan30erd)
- Stop counting soft-deleted pods for scaling purposes in HPA controller to avoid soft-deleted pods incorrectly affecting scale up replica count calculation. (#67067, @moonek)
- delegated authn/z: optionally opt-out of mandatory authn/authz kubeconfig (#67545, @sttts)
- kubeadm: Control plane images (etcd, kube-apiserver, kube-proxy, etc.) don't use arch suffixes. Arch suffixes are kept for kube-dns only. (#66960, @rosti)
- Adds sample-cli-plugin staging repository (#67938, @soltysh)
- adjusted http/2 buffer sizes for apiservers to prevent starvation issues between concurrent streams (#67902, @liggitt)
- SCTP is now supported as additional protocol (alpha) alongside TCP and UDP in Pod, Service, Endpoint, and NetworkPolicy. (#64973, @janosi)
- Always create configmaps/extensions-apiserver-authentication from kube-apiserver. (#67694, @sttts)
- kube-proxy v1beta1 external ComponentConfig types are now available in the `k8s.io/kube-proxy` repo (#67688, @Lion-Wei)
- Apply unreachable taint to a node when it lost network connection. (#67734, @Huang-Wei)
- Allow ImageReview backend to return annotations to be added to the created pod. (#64597, @wteiken)
- Bump ip-masq-agent to v2.1.1 (#67916, @MrHohn)
  - - Update debian-iptables image for CVEs.
  - - Change chain name to IP-MASQ to be compatible with the
  - pre-injected masquerade rules.
- AllowedTopologies field inside StorageClass is now validated against set and map semantics. Specifically, there cannot be duplicate TopologySelectorTerms, MatchLabelExpressions keys, and TopologySelectorLabelRequirement Values. (#66843, @verult)
- Introduces autoscaling/v2beta2 and custom_metrics/v1beta2, which implement metric selectors for Object and Pods metrics, as well as allowing AverageValue targets on Objects, similar to External metrics. (#64097, @damemi)
- The cloudstack cloud provider now reports a `Hostname` address type for nodes based on the `local-hostname` metadata key. (#67719, @liggitt)
- kubeadm: –cri-socket now defaults to tcp://localhost:2375 when running on Windows (#67447, @benmoss)
- kubeadm: The kubeadm configuration now support definition of more than one control plane instances with their own APIEndpoint. The APIEndpoint for the "bootstrap" control plane instance should be

defined using `InitConfiguration.APIEndpoint`, while the APIEndpoints for additional control plane instances should be added using `JoinConfiguration.APIEndpoint`. (#67832, @fabriziopandini)

- Enable dynamic azure disk volume limits (#67772, @andyzhangx)
- kubelet: Users can now enable the alpha NodeLease feature gate to have the Kubelet create and periodically renew a Lease in the kube-node-lease namespace. The lease duration defaults to 40s, and can be configured via the kubelet.config.k8s.io/v1beta1.KubeletConfiguration's NodeLeaseDurationSeconds field. (#66257, @mtaufen)
- latent controller caches no longer cause repeating deletion messages for deleted pods (#67826, @deads2k)
- API paging is now enabled for custom resource definitions, custom resources and APIService objects (#67861, @liggitt)
- kubeadm: ControlPlaneEndpoint was moved from the API config struct to ClusterConfiguration (#67830, @fabriziopandini)
- kubeadm - feature-gates HighAvailability, SelfHosting, CertsInSecrets are now deprecated and can't be used anymore for new clusters. Update of cluster using above feature-gates flag is not supported (#67786, @fabriziopandini)
- Replace scale up forbidden window with disregarding CPU samples collected when pod was initializing. (#67252, @jbartosik)
- Moving KubeSchedulerConfiguration from ComponentConfig API types to staging repos (#66916, @dixudx)
- Improved error message when checking the rollout status of StatefulSet with OnDelete strategy type (#66983, @mortent)
- RuntimeClass is a new API resource for defining different classes of runtimes that may be used to run containers in the cluster. Pods can select a RunitmeClass to use via the RuntimeClassName field. This feature is in alpha, and the RuntimeClass feature gate must be enabled in order to use it. (#67737, @tallclair)
- Remove rescheduler since scheduling DS pods by default scheduler is moving to beta. (#67687, @Lion-Wei)
- Turn on PodReadinessGate by default (#67406, @freehan)
- Speed up kubelet start time by executing an immediate runtime and node status update when the Kubelet sees that it has a CIDR. (#67031, @krzysztof-jastrzebski)
- The OpenStack cloud provider now reports a `Hostname` address type for nodes (#67748, @FengyunPan2)
- The aws cloud provider now reports a `Hostname` address type for nodes based on the `local-hostname` metadata key. (#67715, @liggitt)
- Azure cloud provider now supports cross resource group nodes that are labeled with `kubernetes.azure.com/resource-group=<rg-name>` and `alpha.service-controller.kubernetes.io/exclude-balancer=true` (#67604, @feiskyer)
- Reduce API calls for Azure instance metadata. (#67478, @feiskyer)
- `kubectl create secret tls` can now read certificate and key files from

process substitution arguments (#67713, @liggitt)

- change default value of kind for azure disk (#67483, @andyzhangx)
- To address the possibility dry-run requests overwhelming admission webhooks that rely on side effects and a reconciliation mechanism, a new field is being added to admissionregistration.k8s.io/v1beta1.ValidatingWebhookConfiguration and admissionregistration.k8s.io/v1beta1.MutatingWebhookConfiguration so that webhooks can explicitly register as having dry-run support. If a dry-run request is made on a resource that triggers a non dry-run supporting webhook, the request will be completely rejected, with "400: Bad Request". Additionally, a new field is being added to the admission.k8s.io/v1beta1.AdmissionReview API object, exposing to webhooks whether or not the request being reviewed is a dry-run. (#66936, @jennybuckley)
- Kubeadm ha upgrade (#66973, @fabriziopandini)
- kubeadm: InitConfiguration now consists of two structs: InitConfiguration and ClusterConfiguration (#67441, @rosti)
- Updated Cluster Autoscaler version to 1.3.2-beta.2. Release notes: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.3.2-beta.2 (#67697, @aleksandra-malinowska)
- cpumanager: rollback state if updateContainerCPUSet failed (#67430, @choury)
- [CRI] Adds a "runtime_handler" field to RunPodSandboxRequest, for selecting the runtime configuration to run the sandbox with (alpha feature). (#67518, @tallclair)
- Create cli-runtime staging repository (#67658, @soltysh)
- Headless Services with no ports defined will now create Endpoints correctly, and appear in DNS. (#67622, @thockin)
- Kubernetes juju charms will now use CSI for ceph. (#66523, @hyperbolic2346)
- kubeadm: Fix panic when node annotation is nil (#67648, @xlgao-zju)
- Prevent `resourceVersion` updates for custom resources on no-op writes. (#67562, @nikhita)
- Fail container start if its requested device plugin resource hasn't registered after Kubelet restart. (#67145, @jiayingz)
- Use sync.map to scale ecache better (#66862, @resouer)
- DaemonSet: Fix bug- daemonset didn't create pod after node have enough resource (#67337, @linyouchong)
- updates kibana to 6.3.2 (#67582, @monotek)
- fixes json logging in fluentd-elasticsearch image by downgrading fluent-plugin-kubernetes_metadata_filter plugin to version 2.0.0 (#67544, @monotek)
- add –dns-loop-detect option to dnsmasq run by kube-dns (#67302, @dixudx)
- Switched certificate data replacement from "REDACTED" to "DATA+OMITTED" (#66023, @ibrasho)
- improve performance of anti-affinity predicate of default scheduler.

(#66948, @mohamed-mehany)

- Fixed a bug that was blocking extensible error handling when serializing API responses error out. Previously, serialization failures always resulted in the status code of the original response being returned. Now, the following behavior occurs: (#67041, @tristanburgess)
  - - If the serialization type is application/vnd.kubernetes.protobuf, and protobuf marshaling is not implemented for the requested API resource type, a '406 Not Acceptable is returned'.
  - - If the serialization type is 'application/json':
  - - If serialization fails, and the original status code was an failure (e.g. 4xx or 5xx), the original status code will be returned.
  - - If serialization fails, and the original status code was not a failure (e.g. 2xx), the status code of the serialization failure will be returned. By default, this is '500 Internal Server Error', because JSON serialization is our default, and not supposed to be implemented on a type-by-type basis.
- Add a feature to the scheduler to score fewer than all nodes in every scheduling cycle. This can improve performance of the scheduler in large clusters. (#66733, @bsalamat)
- kube-controller-manager can now start the quota controller when discovery results can only be partially determined. (#67433, @deads2k)
- The plugin mechanism functionality now closely follows the git plugin design (#66876, @juanvallejo)
- GCE: decrease cpu requests on master node, to allow more components to fit on one core machine. (#67504, @loburm)
- PVC may not be synced to controller local cache in time if PV is bound by external PV binder (e.g. kube-scheduler), double check if PVC is not found to prevent reclaiming PV wrongly. (#67062, @cofyc)
- add more storage account sku support for azure disk (#67528, @andyzhangx)
- updates es-image to elasticsearch 6.3.2 (#67484, @monotek)
- Bump GLBC version to 1.2.3 (#66793, @freehan)
- kube-apiserver: fixes error creating system priority classes when starting multiple apiservers simultaneously (#67372, @tanshanshan)
- kubectl patch now respects –local (#67399, @deads2k)
- Defaults for file audit logging backend in batch mode changed: (#67223, @tallclair)
  - - Logs are written 1 at a time (no batching)
  - - Only a single writer process (lock contention)
- Forget rate limit when CRD establish controller successfully updated CRD condition (#67370, @yue9944882)
- updates fluentd in fluentd-elasticsearch to version 1.2.4 (#67434, @monotek) * also updates activesupport, fluent-plugin-elasticsearch & oj gems
- The dockershim now sets the "bandwidth" and "ipRanges" CNI capabilities (dynamic parameters). Plugin authors and administrators can now

take advantage of this by updating their CNI configuration file. For more information, see the CNI docs (#64445, @squeed)

- Expose `/debug/flags/v` to allow kubelet dynamically set glog logging level. If want to change glog level to 3, you only have to send a PUT request like `curl -X PUT http://127.0.0.1:8080/debug/flags/v -d "3"`. (#64601, @hzxuzhonghu)
- Fix an issue that pods using hostNetwork keep increasing. (#67456, @Huang-Wei)
- DaemonSet controller is now using backoff algorithm to avoid hot loops fighting with kubelet on pod recreation when a particular DaemonSet is misconfigured. (#65309, @tnozicka)
- Add node affinity for Azure unzoned managed disks (#67229, @feiskyer)
- Attacher/Detacher refactor for local storage (#66884, @NickrenREN)
- Update debian-iptables and hyperkube-base images to include CVE fixes. (#67365, @ixdy)
- Fix an issue where filesystems are not unmounted when a backend is not reachable and returns EIO. (#67097, @chakri-nelluri)
- Update Cluster Autoscaler version to 1.3.2-beta.1. Release notes: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.3.2-beta.1 (#67396, @aleksandra-malinowska)
- Remove unused binary and container image for kube-aggregator. The functionality is already integrated into the kube-apiserver. (#67157, @dims)
- Avoid creating new controller revisions for statefulsets when cache is stale (#67039, @mortent)
- Revert #63905: Setup dns servers and search domains for Windows Pods. DNS for Windows containers will be set by CNI plugins. (#66587, @feiskyer)
- attachdetach controller attaches volumes immediately when Pod's PVCs are bound (#66863, @cofyc)
- The check for unsupported plugins during volume resize has been moved from the admission controller to the two controllers that handle volume resize. (#66780, @kangarlou)
- Fix kubelet to not leak goroutines/intofiy watchers on an inactive connection if it's closed (#67285, @yujuhong)
- fix azure disk create failure due to sdk upgrade (#67236, @andyzhangx)
- Kubeadm join –control-plane main workflow (#66873, @fabriziopandini)
- Dynamic provisions that create iSCSI PVs can ensure that multipath is used by specifying 2 or more target portals in the PV, which will cause kubelet to wait up to 10 seconds for the multipath device. PVs with just one portal continue to work as before, with kubelet not waiting for the multipath device and just using the first disk it finds. (#67140, @bswartz)
- kubectl: recreating resources for immutable fields when force is applied (#66602, @dixudx)
- Remove deprecated –interactive flag from kubectl logs. (#65420, @jsoref)
- kubeadm uses audit policy v1 instead of v1beta1 (#67176, @charrywanganthony)

- kubeadm: make sure pre-pulled kube-proxy image and the one specified in its daemon set manifest are the same (#67131, @rosti)
- Graduate Resource Quota ScopeSelectors to beta, and enable it by default. (#67077, @vikaschoudhary16)
- Decrease the amount of time it takes to modify kubeconfig files with large amounts of contexts (#67093, @juanvallejo)
- Fixes issue when updating a DaemonSet causes a hash collision. (#66476, @mortent)
- fix cluster-info dump error (#66652, @charrywanganthony)
- The PodShareProcessNamespace feature to configure PID namespace sharing within a pod has been promoted to beta. (#66507, @verb)
- `kubectl create {clusterrole,role}`'s `--resources` flag supports asterisk to specify all resources. (#62945, @nak3)
- Bump up version number of debian-base, debian-hyperkube-base and debian-iptables. (#67026, @satyasm)
  - Also updates dependencies of users of debian-base.
  - debian-base version 0.3.1 is already available.
- DynamicProvisioningScheduling and VolumeScheduling is now supported for Azure managed disks. Feature gates DynamicProvisioningScheduling and VolumeScheduling should be enabled before using this feature. (#67121, @feiskyer)
- kube-apiserver now includes all registered API groups in discovery, including registered extension API group/versions for unavailable extension API servers. (#66932, @nilebox)
- Allows extension API server to dynamically discover the requestheader CA certificate when the core API server doesn't use certificate based authentication for it's clients (#66394, @rtripat)
- audit.k8s.io api group is upgraded from v1beta1 to v1. (#65891, @CaoShuFeng)
  - Deprecated element metav1.ObjectMeta and Timestamp are removed from audit Events in v1 version.
  - Default value of option –audit-webhook-version and –audit-log-version will be changed from `audit.k8s.io/v1beta1` to `audit.k8s.io/v1` in release 1.13
- scope AWS LoadBalancer security group ICMP rules to spec.loadBalancerSourceRanges (#63572, @haz-mat)
- Add NoSchedule/NoExecute tolerations to ip-masq-agent, ensuring it to be scheduled in all nodes except master. (#66260, @tanshanshan)
- The flag `--skip-preflight-checks` of kubeadm has been removed. Please use `--ignore-preflight-errors` instead. (#62727, @xiangpengzhao)
- The watch API endpoints prefixed with `/watch` are deprecated and will be removed in a future release. These standard method for watching resources (supported since v1.0) is to use the list API endpoints with a `?watch=true` parameter. All client-go clients have used the parameter method since v1.6.0. (#65147, @liggitt)

- Bump Heapster to v1.6.0-beta.1 (#67074, @kawych)
- kube-apiserver: setting a `dryRun` query parameter on a CONNECT request will now cause the request to be rejected, consistent with behavior of other mutating API requests. Examples of CONNECT APIs are the `nodes/proxy`, `services/proxy`, `pods/proxy`, `pods/exec`, and `pods/attach` subresources. Note that this prevents sending a `dryRun` parameter to backends via `{nodes,services,pods}/proxy` subresources. (#66083, @jennybuckley)
- In clusters where the DryRun feature is enabled, dry-run requests will go through the normal admission chain. Because of this, ImagePolicyWebhook authors should especially make sure that their webhooks do not rely on side effects. (#66391, @jennybuckley)
- Metadata Agent Improvements (#66485, @bmoyles0117)
  - Bump metadata agent version to 0.2-0.0.21-1.
  - Expand the metadata agent's access to all API groups.
  - Remove metadata agent config maps in favor of command line flags.
  - Update the metadata agent's liveness probe to a new /healthz handler.
  - Logging Agent Improvements
  - Bump logging agent version to 0.2-1.5.33-1-k8s-1.
  - Appropriately set log severity for k8s_container.
  - Fix detect exceptions plugin to analyze message field instead of log field.
  - Fix detect exceptions plugin to analyze streams based on local resource id.
  - Disable the metadata agent for monitored resource construction in logging.
  - Disable timestamp adjustment in logs to optimize performance.
  - Reduce logging agent buffer chunk limit to 512k to optimize performance.
- kubectl: the wait command now prints an error message and exits with the code 1, if there is no resources matching selectors (#66692, @m1kola)
- Quota admission configuration api graduated to v1beta1 (#66156, @vikaschoudhary16)
- Unit tests for scopes and scope selectors in the quota spec (#66351, @vikaschoudhary16)
- Print kube-apiserver –help flag help in sections. (#64517, @sttts)
- Azure managed disks now support availability zones and new parameters `zoned`, `zone` and `zones` are added for AzureDisk storage class. (#66553, @feiskyer)
- nodes: improve handling of erroneous host names (#64815, @dixudx)
- remove deprecated shorthand flag `-c` from `kubectl version (--client)` (#66817, @charrywanganthony)
- Added etcd_object_count metrics for CustomResources. (#65983, @sttts)
- Handle newlines for `command`, `args`, `env`, and `annotations` in `kubectl`

`describe` wrapping (#66841, @smarterclayton)
- Fix pod launch by kubelet when –cgroups-per-qos=false and –cgroup-driver="systemd" (#66617, @pravisankar)
- kubelet: fix nil pointer dereference while enforce-node-allocatable flag is not config properly (#66190, @linyouchong)
- Fix a bug on GCE that /etc/crictl.yaml is not generated when crictl is preloaded. (#66877, @Random-Liu)
- This fix prevents a GCE PD volume from being mounted if the udev device link is stale and tries to correct the link. (#66832, @msau42)

v1.12.0-alpha.1

Documentation & Examples

Downloads for v1.12.0-alpha.1

filename	sha256 hash
kubernetes.tar.gz	`603345769f5e2306e5c22db928aa1cbedc6af63f387ab7a8818cb0111292133f`
kubernetes-src.tar.gz	`f8fb4610cee20195381e54bfd163fbaeae228d68986817b685948b8957f324d0`

Client Binaries

filename	sha256 hash
kubernetes-client-darwin-386.tar.gz	`e081c275601bcaa45d906a976d35902256f836bb60caa738a2fd8719`
kubernetes-client-darwin-amd64.tar.gz	`2dd222a267ac247dce4dfc52aff313f20c427b4351f7410aadebe856`
kubernetes-client-linux-386.tar.gz	`46b16d6b0429163da67b06242772c3c6c5ab9da6deda5306e63d21be`
kubernetes-client-linux-amd64.tar.gz	`8b8bf0a8a4568559d3762a72c1095ab37785fc8bbbb290aaff3a3434`
kubernetes-client-linux-arm.tar.gz	`d71dc60e087746b2832e66170053816dc8ed42e95efe0769ed926a6e`
kubernetes-client-linux-arm64.tar.gz	`e9091bbfb997d1603dfd17ba9f145ca7dacf304f04d10230e056f8a1`
kubernetes-client-linux-ppc64le.tar.gz	`fc6c0985ccbd806add497f2557000f7e90f3176427250e019a40e8ac`
kubernetes-client-linux-s390x.tar.gz	`b8c64b318d702f6e8be76330fd5da9b87e2e4e31e904ea7e00c0cd64`
kubernetes-client-windows-386.tar.gz	`cb96e353eb5d400756a93c8d16321d0fac87d6a4f8ad89fda42858f8`
kubernetes-client-windows-amd64.tar.gz	`003284f983cafc6fd0ce1205c03d47e638a999def1ef4e1e77bfb914`

Server Binaries

filename	sha256 hash
kubernetes-server-linux-amd64.tar.gz	`d9c282cd02c8c3fdbeb2f46abd0ddd257a8449e94be3beed2514c6e30a`

filename	sha256 hash
kubernetes-server-linux-arm.tar.gz	`613390ba73f4236feb10bb4f70cbf96e504cf8d598da0180efc887d316`
kubernetes-server-linux-arm64.tar.gz	`1dd417f59d17c3583c6b4a3989d24c57e4989eb7b6ab9f2aa10c4cbf9b`
kubernetes-server-linux-ppc64le.tar.gz	`44e9e6424ed3a5a91f5adefa456b2b71c0c5d3b01be9f60f5c8c0f9588`
kubernetes-server-linux-s390x.tar.gz	`3118d9c955f9a50f86ebba324894f06dbf7c1cb8f9bc5bdf6a95caf2a6`

Node Binaries

filename	sha256 hash
kubernetes-node-linux-amd64.tar.gz	`6b4d363d190e0ce6f4e41d19a0ac350b39cad7859bc442166a1da9124`
kubernetes-node-linux-arm.tar.gz	`c80ac005c228217b871bf3e9de032044659db3aa048cc95b101820e31`
kubernetes-node-linux-arm64.tar.gz	`d8b84e7cc6ff5d0e26b045de37bdd40ca8809c303b601d8604902e595`
kubernetes-node-linux-ppc64le.tar.gz	`b0a667c5c905e6e724fba95d44797fb52afb564aedd1c25cbd4e632e1`
kubernetes-node-linux-s390x.tar.gz	`78e7dbb82543ea6ac70767ed63c92823726adb6257f6b70b5911843d1`
kubernetes-node-windows-amd64.tar.gz	`1a3e11cc3f1a0297de2b894a43eb56ede5fbd5cdc43e4da7e61171f5c`

Changelog since v1.11.0

Action Required

- action required: the API server and client-go libraries have been fixed to support additional non-alpha-numeric characters in UserInfo "extra" data keys. Both should be updated in order to properly support extra data containing "/" characters or other characters disallowed in HTTP headers. (#65799, @dekkagaijin)
- [action required] The `NodeConfiguration` kind in the kubeadm v1alpha2 API has been renamed `JoinConfiguration` in v1alpha3 (#65951, @luxas)
- ACTION REQUIRED: Removes defaulting of CSI file system type to ext4. All the production drivers listed under https://kubernetes-csi.github.io/docs/Drivers.html were inspected and should not be impacted after this change. If you are using a driver not in that list, please test the drivers on an updated test cluster first. "' (#65499, @krunaljain)
- [action required] The `MasterConfiguration` kind in the kubeadm v1alpha2 API has been renamed `InitConfiguration` in v1alpha3 (#65945, @luxas)
- [action required] The formerly publicly-available cAdvisor web UI that the kubelet started using `--cadvisor-port` is now entirely removed in 1.12. The recommended way to run cAdvisor if you still need it, is via a DaemonSet. (#65707, @dims)
- Cluster Autoscaler version updated to 1.3.1-beta.1. Release notes: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.3.1-beta.1 (#65857, @aleksandra-malinowska)

  - Default value for expendable pod priority cutoff in GCP deployment of Cluster Autoscaler changed from 0 to -10.
  - action required: users deploying workloads with priority lower than 0 may want to use priority lower than -10 to avoid triggering scale-up.
- [action required] kubeadm: The `v1alpha1` config API has been removed. (#65628, @luxas)
  - Please convert your `v1alpha1` configuration files to `v1alpha2` using the
  - `kubeadm config migrate` command of kubeadm v1.11.x
- kube-apiserver: the `Priority` admission plugin is now enabled by default when using `--enable-admission-plugins`. If using `--admission-control` to fully specify the set of admission plugins, the `Priority` admission plugin should be added if using the `PodPriority` feature, which is enabled by default in 1.11. (#65739, @liggitt)
- The `system-node-critical` and `system-cluster-critical` priority classes are now limited to the `kube-system` namespace by the `PodPriority` admission plugin. (#65593, @bsalamat)
- kubernetes-worker juju charm: Added support for setting the –enable-ssl-chain-completion option on the ingress proxy. "action required": if your installation relies on supplying incomplete certificate chains and using OCSP to fill them in, you must set "ingress-ssl-chain-completion" to "true" in your juju configuration. (#63845, @paulgear)

Other notable changes

- admin RBAC role now aggregates edit and view. edit RBAC role now aggregates view. (#66684, @deads2k)
- Speed up HPA reaction to metric changes by removing scale up forbidden window. (#66615, @jbartosik)
  - Scale up forbidden window was protecting HPA against making decision to scale up based on metrics gathered during pod initialisation (which may be invalid, for example pod may be using a lot of CPU despite not doing any "actual" work).
  - To avoid that negative effect only use per pod metrics from pods that are:
  - - ready (so metrics about them should be valid), or
  - - unready but creation and last readiness change timestamps are apart more than 10s (pods that have formerly been ready and so metrics are in at least some cases (pod becoming unready because of overload) very useful).
- The `kubectl patch` command no longer exits with exit code 1 when a redundant patch results in a no-op (#66725, @juanvallejo)
- Improved the output of `kubectl get events` to prioritize showing the message, and move some fields to `-o wide`. (#66643, @smarterclayton)
- Added CPU Manager state validation in case of changed CPU topology.

(#66718, @ipuustin)

- Make EBS volume expansion faster (#66728, @gnufied)
- Kubelet serving certificate bootstrapping and rotation has been promoted to beta status. (#66726, @liggitt)
- Flag –pod (-p shorthand) of kubectl exec command marked as deprecated (#66558, @quasoft)
- Fixed an issue which prevented `gcloud` from working on GCE when metadata concealment was enabled. (#66630, @dekkagaijin)
- Azure Go SDK has been upgraded to v19.0.0 and VirtualMachineScaleSetVM now supports availability zones. (#66648, @feiskyer)
- kubeadm now can join the cluster with pre-existing client certificate if provided (#66482, @dixudx)
- If `TaintNodesByCondition` enabled, taint node with `TaintNodeUnschedulable` when (#63955, @k82cn)
  - initializing node to avoid race condition.
- kubeadm: remove misleading error message regarding image pulling (#66658, @dixudx)
- Fix Stackdriver integration based on node annotation container.googleapis.com/instance_id. (#66676, @kawych)
- Fix kubelet startup failure when using ExecPlugin in kubeconfig (#66395, @awly)
- When attaching iSCSI volumes, kubelet now scans only the specific (#63176, @bswartz)
  - LUNs being attached, and also deletes them after detaching. This avoids
  - dangling references to LUNs that no longer exist, which used to be the
  - cause of random I/O errors/timeouts in kernel logs, slowdowns during
  - block-device related operations, and very rare cases of data corruption.
- kubeadm: Pull sidecar and dnsmasq-nanny images when using kube-dns (#66499, @rosti)
- Extender preemption should respect IsInterested() (#66291, @resouer)
- Properly autopopulate OpenAPI version field without needing other OpenAPI fields present in generic API server code. (#66411, @DirectXMan12)
- renamed command line option –cri-socket-path of the kubeadm subcommand "kubeadm config images pull" to –cri-socket to be consistent with the rest of kubeadm subcommands. (#66382, @bart0sh)
- The –docker-disable-shared-pid kubelet flag has been removed. PID namespace sharing can instead be enable per-pod using the ShareProcessNamespace option. (#66506, @verb)
- Add support for using User Assigned MSI (https://docs.microsoft.com/en-us/azure/active-directory/managed-service-identity/overview) with Kubernetes cluster on Azure. (#66180, @kkmsft)
- fix acr could not be listed in sp issue (#66429, @andyzhangx)

- This PR will leverage subtests on the existing table tests for the scheduler units. (#63665, @xchapter7x)
  - Some refactoring of error/status messages and functions to align with new approach.
- Fix volume limit for EBS on m5 and c5 instance types (#66397, @gnufied)
- Extend TLS timeouts to work around slow arm64 math/big (#66264, @joejulian)
- kubeadm: stop setting UID in the kubelet ConfigMap (#66341, @runiq)
- kubectl: fixes a panic displaying pods with nominatedNodeName set (#66406, @liggitt)
- Update crictl to v1.11.1. (#66152, @Random-Liu)
- fixes a panic when using a mutating webhook admission plugin with a DELETE operation (#66425, @liggitt)
- GCE: Fixes loadbalancer creation and deletion issues appearing in 1.10.5. (#66400, @nicksardo)
- Azure nodes with availability zone now will have label `failure-domain.beta.kubernetes.io/zone=<reg` (#66242, @feiskyer)
- Re-design equivalence class cache to two level cache (#65714, @resouer)
- Checks CREATE admission for create-on-update requests instead of UPDATE admission (#65572, @yue9944882)
- This PR will leverage subtests on the existing table tests for the scheduler units. (#63666, @xchapter7x)
  - Some refactoring of error/status messages and functions to align with new approach.
- Fixed a panic in the node status update logic when existing node has nil labels. (#66307, @guoshimin)
- Bump Ingress-gce version to 1.2.0 (#65641, @freehan)
- Bump event-exporter to 0.2.2 to pick up security fixes. (#66157, @loburm)
- Allow ScaleIO volumes to be provisioned without having to first manually create /dev/disk/by-id path on each kubernetes node (if not already present) (#66174, @ddebroy)
- fix rollout status for statefulsets (#62943, @faraazkhan)
- Fix for resourcepool-path configuration in the vsphere.conf file. (#66261, @divyenpatel)
- OpenAPI spec and documentation reflect 202 Accepted response path for delete request (#63418, @roycaihw)
- fixes a validation error that could prevent updates to StatefulSet objects containing non-normalized resource requests (#66165, @liggitt)
- Fix validation for HealthzBindAddress in kube-proxy when –healthz-port is set to 0 (#66138, @wsong)
- kubeadm: use an HTTP request timeout when fetching the latest version of Kubernetes from dl.k8s.io (#65676, @dkoshkin)
- Support configuring the Azure load balancer idle connection timeout for services (#66045, @cpuguy83)
- `kubectl config set-context` can now set attributes of the current context, like the current namespace, by passing `--current` instead of a spe-

cific context name (#66140, @liggitt)

- The alpha `Initializers` admission plugin is no longer enabled by default. This matches the off-by-default behavior of the alpha API which drives initializer behavior. (#66039, @liggitt)
- kubeadm: Default component configs are printable via kubeadm config print-default (#66074, @rosti)
- prevents infinite CLI wait on delete when item is recreated (#66136, @deads2k)
- Preserve vmUUID when renewing nodeinfo in vSphere cloud provider (#66007, @w-leads)
- Cluster Autoscaler version updated to 1.3.1. Release notes: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.3.1 (#66122, @aleksandra-malinowska)
- Expose docker registry config for addons used in Juju deployments (#66092, @kwmonroe)
- kubelets that specify `--cloud-provider` now only report addresses in Node status as determined by the cloud provider (#65594, @liggitt) * kubelet serving certificate rotation now reacts to changes in reported node addresses, and will request certificates for addresses set by an external cloud provider
- Fix the bug where image garbage collection is disabled by mistake. (#66051, @jiaxuanzhou)
- fixes an issue with multi-line annotations injected via downward API files getting scrambled (#65992, @liggitt)
- kubeadm: run kube-proxy on non-master tainted nodes (#65931, @neolit123)
- "kubectl delete" no longer waits for dependent objects to be deleted when removing parent resources (#65908, @juanvallejo)
- Introduce a new flag `--keepalive` for kubectl proxy to allow setting keep-alive period for long-running request. (#63793, @hzxuzhonghu)
- If Openstack LoadBalancer is not defined in cloud config, the loadbalancer is not initialized any more in openstack. All setups must have some setting under that section (#65781, @zetaab)
- Re-adds `pkg/generated/bindata.go` to the repository to allow some parts of k8s.io/kubernetes to be go-vendorable. (#65985, @ixdy)
- Fix a bug that preempting a pod may block forever. (#65987, @Random-Liu)
- Fix flexvolume in containarized kubelets (#65549, @gnufied)
- Add volume mode filed to constructed volume spec for CSI plugin (#65456, @wenlxie)
- Fix an issue with dropped audit logs, when truncating and batch backends enabled at the same time. (#65823, @loburm)
- Support traffic shaping for CNI network driver (#63194, @m1093782566)
- kubeadm: Use separate YAML documents for the kubelet and kube-proxy ComponentConfigs (#65787, @luxas)
- kubeadm: Fix pause image to not use architecture, as it is a manifest list

(#65920, @dims)

- kubeadm: print required flags when running kubeadm upgrade plan (#65802, @xlgao-zju)
- Fix `RunAsGroup` which doesn't work since 1.10. (#65926, @Random-Liu)
- Running `kubectl describe pvc` now shows which pods are mounted to the pvc being described with the `Mounted By` field (#65837, @clandry94)
- fix azure storage account creation failure (#65846, @andyzhangx)
- Allow kube- and cloud-controller-manager to listen on ports up to 65535. (#65860, @sttts)
- Allow kube-scheduler to listen on ports up to 65535. (#65833, @sttts)
- kubeadm: Remove usage of `PersistentVolumeLabel` (#65827, @xlgao-zju)
- kubeadm: Add a `v1alpha3` API. (#65629, @luxas)
- Update to use go1.10.3 (#65726, @ixdy)
- LimitRange and Endpoints resources can be created via an update API call if the object does not already exist. When this occurs, an authorization check is now made to ensure the user making the API call is authorized to create the object. In previous releases, only an update authorization check was performed. (#65150, @jennybuckley)
- Fix 'kubectl cp' with no arguments causes a panic (#65482, @wgliang)
- bazel deb package bugfix: The kubeadm deb package now reloads the kubelet after installation (#65554, @rdodev)
- fix smb mount issue (#65751, @andyzhangx)
- More fields are allowed at the root of the CRD validation schema when the status subresource is enabled. (#65357, @nikhita)
- Reload systemd config files before starting kubelet. (#65702, @mborsz)
- Unix: support ZFS as a valid graph driver for Docker (#65635, @neolit123)
- Fix controller-manager crashes when flex plugin is removed from flex plugin directory (#65536, @gnufied)
- Enable etcdv3 client prometheus metics (#64741, @wgliang)
- skip nodes that have a primary NIC in a 'Failed' provisioningState (#65412, @yastij)
- kubeadm: remove redundant flags settings for kubelet (#64682, @dixudx)
- Fixes the wrong elasticsearch node counter (#65627, @IvanovOleg)
- - Can configure the vsphere cloud provider with a trusted Root-CA (#64758, @mariantalla)
- Add Ubuntu 18.04 (Bionic) series to Juju charms (#65644, @tvansteenburgh)
- Fix local volume directory can't be deleted because of volumeMode error (#65310, @wenlxie)
- kubectl: –use-openapi-print-columns is deprecated in favor of –server-print (#65601, @liggitt)
- Add prometheus scrape port to CoreDNS service (#65589, @rajansandeep)
- fixes an out of range panic in the NoExecuteTaintManager controller when

running a non-64-bit build (#65596, @liggitt)

- kubectl: fixes a regression with –use-openapi-print-columns that would not print object contents (#65600, @liggitt)
- Hostnames are now converted to lowercase before being used for node lookups in the kubernetes-worker charm. (#65487, @dshcherb)
- N/A (#64660, @figo)
- bugfix: Do not print feature gates in the generic apiserver code for glog level 0 (#65584, @neolit123)
- Add metrics for PVC in-use (#64527, @gnufied)
- Fixed exception detection in fluentd-gcp plugin. (#65361, @xperimental)
- api-machinery utility functions `SetTransportDefaults` and `DialerFor` once again respect custom Dial functions set on transports (#65547, @liggitt)
- Improve the display of jobs in `kubectl get` and `kubectl describe` to emphasize progress and duration. (#65463, @smarterclayton)
- kubectl convert previous created a list inside of a list. Now it is only wrapped once. (#65489, @deads2k)
- fix azure disk creation issue when specifying external resource group (#65516, @andyzhangx)
- fixes a regression in kube-scheduler to properly load client connection information from a `--config` file that references a kubeconfig file (#65507, @liggitt)
- Fixed cleanup of CSI metadata files. (#65323, @jsafrane)
- Update Rescheduler's manifest to use version 0.4.0. (#65454, @bsalamat)
- On COS, NPD creates a node condition for frequent occurrences of unregister_netdevice (#65342, @dashpole)
- Properly manage security groups for loadbalancer services on OpenStack. (#65373, @multi-io)
- Add user-agent to audit-logging. (#64812, @hzxuzhonghu)
- kubeadm: notify the user of manifest upgrade timeouts (#65164, @xlgaozju)
- Fixes incompatibility with custom scheduler extender configurations specifying `bindVerb` (#65424, @liggitt)
- Using `kubectl describe` on CRDs that use underscores will be prettier. (#65391, @smarterclayton)
- Improve scheduler's performance by eliminating sorting of nodes by their score. (#65396, @bsalamat)
- Add more conditions to the list of predicate failures that won't be resolved by preemption. (#64995, @bsalamat)
- Allow access to ClusterIP from the host network namespace when kube-proxy is started in IPVS mode without either masqueradeAll or clusterCIDR flags (#65388, @lbernail)
- User can now use `sudo crictl` on GCE cluster. (#65389, @Random-Liu)
- Tolerate missing watch permission when deleting a resource (#65370, @deads2k)
- Prevents a `kubectl delete` hang when deleting controller managed lists

(#65367, @deads2k)
- fixes a memory leak in the kube-controller-manager observed when large numbers of pods with tolerations are created/deleted (#65339, @liggitt)
- checkLimitsForResolvConf for the pod create and update events instead of checking period (#64860, @wgliang)
- Fix concurrent map access panic (#65334, @dashpole)
  - Don't watch .mount cgroups to reduce number of inotify watches
  - Fix NVML initialization race condition
  - Fix brtfs disk metrics when using a subdirectory of a subvolume
- Change Azure ARM Rate limiting error message. (#65292, @wgliang)
- AWS now checks for validity of ecryption key when creating encrypted volumes. Dynamic provisioning of encrypted volume may get slower due to these checks. (#65223, @jsafrane)
- Report accurate status for kubernetes-master and -worker charms. (#65187, @kwmonroe)
- Fixed issue 63608, which is that under rare circumstances the ResourceQuota admission controller could lose track of an request in progress and time out after waiting 10 seconds for a decision to be made. (#64598, @MikeSpreitzer)
- In the vSphere cloud provider the `Global.vm-uuid` configuration option is not deprecated anymore, it can be used to overwrite the VMUUID on the controller-manager (#65152, @alvaroaleman)
- fluentd-gcp grace termination period increased to 60s. (#65084, @x13n)
- Pass cluster_location argument to Heapster (#65176, @kawych)
- Fix a scalability issue where high rates of event writes degraded etcd performance. (#64539, @ccding)
- Corrected a mistake in the documentation for wait.PollImmediate(…) (#65026, @spew)
- Split 'scheduling_latency_seconds' metric into finer steps (predicate, priority, premption) (#65306, @shyamjvs)
- Etcd health checks by the apiserver now ensure the apiserver can connect to and exercise the etcd API (#65027, @liggitt)
- Add e2e regression tests for the kubelet being secure (#64140, @dixudx)
- set EnableHTTPSTrafficOnly in azure storage account creation (#64957, @andyzhangx)
- Fixes an issue where Portworx PVCs remain in pending state when created using a StorageClass with empty parameters (#64895, @harsh-px)
- This PR will leverage subtests on the existing table tests for the scheduler units. (#63662, @xchapter7x)
  - Some refactoring of error/status messages and functions to align with new approach.
- This PR will leverage subtests on the existing table tests for the scheduler units. (#63661, @xchapter7x)
  - Some refactoring of error/status messages and functions to align with new approach.
- This PR will leverage subtests on the existing table tests for the scheduler

units. (#63660, @xchapter7x)
    - Some refactoring of error/status messages and functions to align with new approach.
- Updated default image for nginx ingress in CDK to match current Kubernetes docs. (#64285, @hyperbolic2346)
- Added block volume support to Cinder volume plugin. (#64879, @bertinatto)
- fixed incorrect OpenAPI schema for CustomResourceDefinition objects (#65256, @liggitt)
- ignore not found file error when watching manifests (#64880, @dixudx)
- add port-forward examples for sevice (#64773, @MasayaAoyama)
- Fix issues for block device not mapped to container. (#64555, @wenlxie)
- Update crictl on GCE to v1.11.0. (#65254, @Random-Liu)
- Fixes missing nodes lines when kubectl top nodes (#64389, @yue9944882)
- keep pod state consistent when scheduler cache UpdatePod (#64692, @adohe)
- add external resource group support for azure disk (#64427, @andyzhangx)
- Increase the gRPC max message size to 16MB in the remote container runtime. (#64672, @mcluseau)
- The new default value for the –allow-privileged parameter of the Kubernetes-worker charm has been set to true based on changes which went into the Kubernetes 1.10 release. Before this change the default value was set to false. If you're installing Canonical Kubernetes you should expect this value to now be true by default and you should now look to use PSP (pod security policies). (#64104, @CalvinHartwell)
- The –remove-extra-subjects and –remove-extra-permissions flags have been enabled for kubectl auth reconcile (#64541, @mrogers950)
- Fix kubectl drain –timeout option when eviction is used. (#64378, @wrdls)
- This PR will leverage subtests on the existing table tests for the scheduler units. (#63659, @xchapter7x)
    - Some refactoring of error/status messages and functions to align with new approach.

Edit This Page

Troubleshooting kubeadm

As with any program, you might run into an error installing or running kubeadm. This page lists some common failure scenarios and have provided steps that can help you understand and fix the problem.

If your problem is not listed below, please follow the following steps:

- If you think your problem is a bug with kubeadm:

  - Go to github.com/kubernetes/kubeadm and search for existing issues.
  - If no issue exists, please open one and follow the issue template.

- If you are unsure about how kubeadm works, you can ask on Slack in #kubeadm, or open a question on StackOverflow. Please include relevant tags like `#kubernetes` and `#kubeadm` so folks can help you.

- `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
- `coredns` pods have `CrashLoopBackOff` or `Error` state

`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 and nothing happens to the DNS server, 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.
- If you install a version of Docker older than 1.12.1, remove the `MountFlags=slave` option when booting `dockerd` with `systemd` and restart `docker`. You can see the MountFlags in `/usr/lib/systemd/system/docker.service`. MountFlags can interfere with volumes mounted by Kubernetes, and put the Pods in `CrashLoopBackOff` state. The error happens when Kubernetes does not find `var/run/secrets/kubernetes.io/serviceaccount` files.

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

This is **expected** and part of the design. kubeadm is network provider-agnostic, so the admin should install the pod network solution of choice. You have to install a Pod Network before CoreDNS may deployed fully. Hence the `Pending` state before the network is set up.

`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 kubeadm `NodeRegistrationOptions` structure 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
)"
```

`coredns` pods have `CrashLoopBackOff` or `Error` state

If you have nodes that are running SELinux with an older version of Docker you might experience a scenario where the `coredns` pods are not starting. To solve that you can try one of the following options:

- Upgrade to a newer version of Docker.
- Disable SELinux.
- Modify the `coredns` deployment to set `allowPrivilegeEscalation` to `true`:

```
kubectl -n kube-system get deployment coredns -o yaml | \
  sed 's/allowPrivilegeEscalation: false/allowPrivilegeEscalation: true/g' | \
  kubectl apply -f -
```

Another cause for CoreDNS to have `CrashLoopBackOff` is when a CoreDNS Pod deployed in Kubernetes detects a loop. A number of workarounds are available to avoid Kubernetes trying to restart the CoreDNS Pod every time CoreDNS detects the loop and exits.

> **Warning:** Disabling SELinux or setting `allowPrivilegeEscalation` to `true` can compromise the security of your cluster.

Edit This Page

Installing kubeadm

This page shows how to install the `kubeadm` toolbox. For information how to create a cluster with kubeadm once you have performed this installation process, see the Using kubeadm to Create a Cluster page.

- Before you begin
- Verify the MAC address and product_uuid are unique for every node
- Check network adapters
- Check required ports
- Installing runtime
- Installing kubeadm, kubelet and kubectl
- Configure cgroup driver used by kubelet on Master Node
- Troubleshooting
- What's next

Before you begin

- One or more machines running one of:
  - Ubuntu 16.04+
  - Debian 9
  - CentOS 7
  - RHEL 7
  - Fedora 25/26 (best-effort)
  - HypriotOS v1.0.1+
  - Container Linux (tested with 1800.6.0)
- 2 GB or more of RAM per machine (any less will leave little room for your apps)
- 2 CPUs or more
- Full network connectivity between all machines in the cluster (public or private network is fine)
- Unique hostname, MAC address, and product_uuid for every node. See here for more details.

- Certain ports are open on your machines. See here for more details.
- Swap disabled. You **MUST** disable swap in order for the kubelet to work properly.

Verify the MAC address and product_uuid are unique for every node

- You can get the MAC address of the network interfaces using the command `ip link` or `ifconfig -a`
- The product_uuid can be checked by using the command `sudo cat /sys/class/dmi/id/product_uuid`

It is very likely that hardware devices will have unique addresses, although some virtual machines may have identical values. Kubernetes uses these values to uniquely identify the nodes in the cluster. If these values are not unique to each node, the installation process may fail.

Check network adapters

If you have more than one network adapter, and your Kubernetes components are not reachable on the default route, we recommend you add IP route(s) so Kubernetes cluster addresses go via the appropriate adapter.

Check required ports

Master node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443*	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10251	kube-scheduler	Self
TCP	Inbound	10252	kube-controller-manager	Self

Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	30000-32767	NodePort Services**	All

** Default port range for NodePort Services.

Any port numbers marked with * are overridable, so you will need to ensure any custom ports you provide are also open.

Although etcd ports are included in master nodes, you can also host your own etcd cluster externally or on custom ports.

The pod network plugin you use (see below) may also require certain ports to be open. Since this differs with each pod network plugin, please see the documentation for the plugins about what port(s) those need.

Installing runtime

Since v1.6.0, Kubernetes has enabled the use of CRI, Container Runtime Interface, by default. The container runtime used by default is Docker, which is enabled through the built-in `dockershim` CRI implementation inside of the `kubelet`.

Other CRI-based runtimes include:

- containerd (CRI plugin built into containerd)
- cri-o
- frakti
- rkt

Refer to the CRI installation instructions for more information.

Installing kubeadm, kubelet and kubectl

You will install these packages on all of your machines:

- `kubeadm`: the command to bootstrap the cluster.
- `kubelet`: the component that runs on all of the machines in your cluster and does things like starting pods and containers.
- `kubectl`: the command line util to talk to your cluster.

kubeadm **will not** install or manage `kubelet` or `kubectl` for you, so you will need to ensure they match the version of the Kubernetes control panel you want kubeadm to install for you. If you do not, there is a risk of a version skew occurring that can lead to unexpected, buggy behaviour. However, *one* minor version skew between the kubelet and the control plane is supported, but the kubelet version may never exceed the API server version. For example, kubelets running 1.7.0 should be fully compatible with a 1.8.0 API server, but not vice versa.

> **Warning:** These instructions exclude all Kubernetes packages from any system upgrades. This is because kubeadm and Kubernetes require special attention to upgrade.

For more information on version skews, please read our version skew policy.

- Ubuntu, Debian or HypriotOS
- CentOS, RHEL or Fedora
- Container Linux

```
apt-get update && apt-get install -y apt-transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
apt-get install -y kubelet kubeadm kubectl
apt-mark hold kubelet kubeadm kubectl
```

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.c
exclude=kube*
EOF

# Set SELinux in permissive mode (effectively disabling it)
setenforce 0
sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes

systemctl enable kubelet && systemctl start kubelet
```

Note:

- Setting SELinux in permissive mode by running `setenforce 0` and `sed ...` effectively disables it. This is required to allow containers to access the host filesystem, which is needed by pod networks for example. You have to do this until SELinux support is improved in the kubelet.
- Some users on RHEL/CentOS 7 have reported issues with traffic being routed incorrectly due to iptables being bypassed. You should ensure `net.bridge.bridge-nf-call-iptables` is set to 1 in your `sysctl` config, e.g.

```
     cat <<EOF >  /etc/sysctl.d/k8s.conf
     net.bridge.bridge-nf-call-ip6tables = 1
     net.bridge.bridge-nf-call-iptables = 1
     EOF
     sysctl --system
```

Install CNI plugins (required for most pod network):

```
CNI_VERSION="v0.6.0"
mkdir -p /opt/cni/bin
curl -L "https://github.com/containernetworking/plugins/releases/download/${CNI_VERSION}/cni
```

Install crictl (required for kubeadm / Kubelet Container Runtime Interface (CRI))

```
CRICTL_VERSION="v1.11.1"
mkdir -p /opt/bin
curl -L "https://github.com/kubernetes-incubator/cri-tools/releases/download/${CRICTL_VERSI
```

Install `kubeadm`, `kubelet`, `kubectl` and add a `kubelet` systemd service:

```
RELEASE="$(curl -sSL https://dl.k8s.io/release/stable.txt)"

mkdir -p /opt/bin
cd /opt/bin
curl -L --remote-name-all https://storage.googleapis.com/kubernetes-release/release/${RELEAS
chmod +x {kubeadm,kubelet,kubectl}

curl -sSL "https://raw.githubusercontent.com/kubernetes/kubernetes/${RELEASE}/build/debs/kub
mkdir -p /etc/systemd/system/kubelet.service.d
curl -sSL "https://raw.githubusercontent.com/kubernetes/kubernetes/${RELEASE}/build/debs/10-
```

Enable and start `kubelet`:

```
systemctl enable kubelet && systemctl start kubelet
```

The kubelet is now restarting every few seconds, as it waits in a crashloop for kubeadm to tell it what to do.

Configure cgroup driver used by kubelet on Master Node

When using Docker, kubeadm will automatically detect the cgroup driver for the kubelet and set it in the `/var/lib/kubelet/kubeadm-flags.env` file during runtime.

If you are using a different CRI, you have to modify the file `/etc/default/kubelet` with your `cgroup-driver` value, like so:

```
KUBELET_EXTRA_ARGS=--cgroup-driver=<value>
```

This file will be used by `kubeadm init` and `kubeadm join` to source extra user defined arguments for the kubelet.

Please mind, that you **only** have to do that if the cgroup driver of your CRI is not `cgroupfs`, because that is the default value in the kubelet already.

Restarting the kubelet is required:

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

Troubleshooting

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

What's next

- Using kubeadm to Create a Cluster

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 is designed to be a simple way for new users to start trying Kubernetes out, possibly for the first time, a way for existing users to test their application on and stitch together a cluster easily, and also to be a building block in other ecosystem and/or installer tool with a larger scope.

You can install *kubeadm* very easily on operating systems that support installing deb or rpm packages. The responsible SIG for kubeadm, SIG Cluster Lifecycle, provides these packages pre-built for you, but you may also build them from source for other OSes.

kubeadm Maturity

Area	Maturity Level
Command line UX	beta
Implementation	beta
Config file API	alpha
Self-hosting	alpha
kubeadm alpha subcommands	alpha
CoreDNS	GA
DynamicKubeletConfig	alpha

kubeadm's overall feature state is **Beta** and will soon be graduated to **General Availability (GA)** during 2018. Some sub-features, like self-hosting or the configuration file API are still under active development. The implementation of creating the cluster may change slightly as the tool evolves, but the overall implementation should be pretty stable. Any commands under `kubeadm alpha` are by definition, supported on an alpha level.

Support timeframes

Kubernetes releases are generally supported for nine months, and during that period a patch release may be issued from the release branch if a severe bug or security issue is found. Here are the latest Kubernetes releases and the support timeframe; which also applies to `kubeadm`.

Kubernetes version	Release month	End-of-life-month
v1.6.x	March 2017	December 2017
v1.7.x	June 2017	March 2018
v1.8.x	September 2017	June 2018
v1.9.x	December 2017	September 2018
v1.10.x	March 2018	December 2018
v1.11.x	June 2018	March 2019
v1.12.x	September 2018	June 2019

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`
3. (Optional) Run `kubeadm config images pull` prior to `kubeadm init` to verify connectivity to gcr.io registries.

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
[kubeadm] WARNING: starting in 1.8, tokens expire after 24 hours by default (if you require
[certificates] Generated ca certificate and key.
[certificates] Generated apiserver certificate and key.
[certificates] apiserver serving cert is signed for DNS names [kubeadm-master kubernetes kub
[certificates] Generated apiserver-kubelet-client certificate and key.
[certificates] Generated sa key and public key.
[certificates] Generated front-proxy-ca certificate and key.
[certificates] Generated front-proxy-client certificate and key.
[certificates] Valid certificates and keys now exist in "/etc/kubernetes/pki"
[kubeconfig] Wrote KubeConfig file to disk: "admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "scheduler.conf"
[controlplane] Wrote Static Pod manifest for component kube-apiserver to "/etc/kubernetes/ma
[controlplane] Wrote Static Pod manifest for component kube-controller-manager to "/etc/kube
[controlplane] Wrote Static Pod manifest for component kube-scheduler to "/etc/kubernetes/ma
[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/manifests/etc
[init] Waiting for the kubelet to boot up the control plane as Static Pods from directory "/
[init] This often takes around a minute; or longer if the control plane images have to be pu
[apiclient] All control plane components are healthy after 39.511972 seconds
[uploadconfig] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-sys
[markmaster] Will mark node master as master by adding a label and a taint
[markmaster] Master master tainted and labelled with key/value: node-role.kubernetes.io/mast
[bootstraptoken] Using token: <token>
[bootstraptoken] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order
[bootstraptoken] Configured RBAC rules to allow the csrapprover controller automatically app
[bootstraptoken] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

```
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run (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. Also, CoreDNS will not start up before a network is installed. kubeadm only supports Container Network Interface (CNI) based networks (and 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
- Cilium
- Flannel
- Kube-router
- Romana
- Weave Net
- JuniperContrail/TungstenFabric

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.

For Calico to work correctly, you need to pass `--pod-network-cidr=192.168.0.0/16` to `kubeadm init` or update the `calico.yml` file to match your Pod network. Note that Calico works on `amd64` only.

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installatio
kubectl apply -f https://docs.projectcalico.org/v3.3/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/v3.3/getting-started/kubernetes/installatio
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installatio
```

For more information about using Cilium with Kubernetes, see Quickstart for Cilium on Kubernetes and Kubernetes Install guide for Cilium.

Passing `--pod-network-cidr` option to `kubeadm init` is not required, but highly recommended.

These commands will deploy Cilium with its own etcd managed by etcd operator.

```
# Download required manifests from Cilium repository
wget https://github.com/cilium/cilium/archive/v1.2.0.zip
unzip v1.2.0.zip
cd cilium-1.2.0/examples/kubernetes/addons/etcd-operator

# Generate and deploy etcd certificates
export CLUSTER_DOMAIN=$(kubectl get ConfigMap --namespace kube-system coredns -o yaml | awk
tls/certs/gen-cert.sh $CLUSTER_DOMAIN
tls/deploy-certs.sh

# Label kube-dns with fixed identity label
kubectl label -n kube-system pod $(kubectl -n kube-system get pods -l k8s-app=kube-dns -o js

kubectl create -f ./

# Wait several minutes for Cilium, coredns and etcd pods to converge to a working state
```

For `flannel` to work correctly, you must pass `--pod-network-cidr=10.244.0.0/16` to `kubeadm init`.

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.

Note that `flannel` works on `amd64`, `arm`, `arm64` and `ppc64le`.

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/bc79dd1505b0c8681ece4de4c0
```

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
```

Provides overlay SDN solution, delivering multicloud networking, hybrid cloud networking, simultaneous overlay-underlay support, network policy enforcement, network isolation, service chaining and flexible load balancing.

There are multiple, flexible ways to install JuniperContrail/TungstenFabric CNI.

Kindly refer to this quickstart: TungstenFabric

Once a pod network has been installed, you can confirm that it is working by checking that the CoreDNS pod is Running in the output of `kubectl get pods --all-namespaces`. And once the CoreDNS pod is up and running, you can continue by joining your nodes.

If your network is not working or CoreDNS 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 "node-role.kubernetes.io/master:" not found
taint "node-role.kubernetes.io/master:" 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

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 sha256
```

If you do not have the token, you can get it by running the following command on the master node:

```
kubeadm token list
```

The output is similar to this:

```
TOKEN                    TTL  EXPIRES              USAGES           DESCRIPTION          E
8ewj1p.9r9hcjoqgajrj4gi  23h  2018-06-12T02:51:28Z authentication,  The default bootstrap s
                                                   signing          token generated by   b
                                                                    'kubeadm init'.      k
                                                                                         c
```

By default, tokens expire after 24 hours. If you are joining a node to the cluster after the current token has expired, you can create a new token by running the following command on the master node:

```
kubeadm token create
```

The output is similar to this:

```
5didvk.d09sbcov8ph2amjw
```

If you don't have the value of `--discovery-token-ca-cert-hash`, you can get it by running the following command chain on the master node:

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev
   openssl dgst -sha256 -hex | sed 's/^.* //'
```

The output is similar to this:

```
8cb2de97839780a412b93877f8507ad6c94f73add17d5d7058e91741c9d5ec78
```

> **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.

Edit This Page

Configuring each kubelet in your cluster using kubeadm

FEATURE STATE: `Kubernetes 1.11` stable

This feature is *stable*, meaning:

- The version name is vX where X is an integer.
- Stable versions of features will appear in released software for many subsequent versions.

The lifecycle of the kubeadm CLI tool is decoupled from the kubelet, which is a daemon that runs on each node within the Kubernetes cluster. The kubeadm CLI tool is executed by the user when Kubernetes is initialized or upgraded, whereas the kubelet is always running in the background.

Since the kubelet is a daemon, it needs to be maintained by some kind of a init system or service manager. When the kubelet is installed using DEBs or RPMs, systemd is configured to manage the kubelet. You can use a different service manager instead, but you need to configure it manually.

Some kubelet configuration details need to be the same across all kubelets involved in the cluster, while other configuration aspects need to be set on a per-kubelet basis, to accommodate the different characteristics of a given machine, such as OS, storage, and networking. You can manage the configuration of your kubelets manually, but kubeadm now provides a `KubeletConfiguration` API type for managing your kubelet configurations centrally.

- Kubelet configuration patterns
- Configure kubelets using kubeadm
- The kubelet drop-in file for systemd
- Kubernetes binaries and package contents

Kubelet configuration patterns

The following sections describe patterns to kubelet configuration that are simplified by using kubeadm, rather than managing the kubelet configuration for each Node manually.

Propagating cluster-level configuration to each kubelet

You can provide the kubelet with default values to be used by `kubeadm init` and `kubeadm join` commands. Interesting examples include using a different CRI runtime or setting the default subnet used by services.

If you want your services to use the subnet `10.96.0.0/12` as the default for services, you can pass the `--service-cidr` parameter to kubeadm:

```
kubeadm init --service-cidr 10.96.0.0/12
```

Virtual IPs for services are now allocated from this subnet. You also need to set the DNS address used by the kubelet, using the `--cluster-dns` flag. This setting needs to be the same for every kubelet on every manager and Node in the cluster. The kubelet provides a versioned, structured API object that can configure most parameters in the kubelet and push out this configuration to each running kubelet in the cluster. This object is called **the kubelet's ComponentConfig**. The ComponentConfig allows the user to specify flags such as the cluster DNS IP addresses expressed as a list of values to a camelCased key, illustrated by the following example:

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
clusterDNS:
- 10.96.0.10
```

For more details on the ComponentConfig have a look at this section.

Providing instance-specific configuration details

Some hosts require specific kubelet configurations, due to differences in hardware, operating system, networking, or other host-specific parameters. The following list provides a few examples.

- The path to the DNS resolution file, as specified by the `--resolv-conf` kubelet configuration flag, may differ among operating systems, or depending on whether you are using `systemd-resolved`. If this path is wrong, DNS resolution will fail on the Node whose kubelet is configured incorrectly.
- The Node API object `.metadata.name` is set to the machine's hostname by default, unless you are using a cloud provider. You can use the `--hostname-override` flag to override the default behavior if you need to specify a Node name different from the machine's hostname.
- Currently, the kubelet cannot automatically detects the cgroup driver used by the CRI runtime, but the value of `--cgroup-driver` must match the cgroup driver used by the CRI runtime to ensure the health of the kubelet.

- Depending on the CRI runtime your cluster uses, you may need to specify different flags to the kubelet. For instance, when using Docker, you need to specify flags such as `--network-plugin=cni`, but if you are using an external runtime, you need to specify `--container-runtime=remote` and specify the CRI endpoint using the `--container-runtime-path-endpoint=<path>`.

You can specify these flags by configuring an individual kubelet's configuration in your service manager, such as systemd.

Configure kubelets using kubeadm

It is possible to configure the kubelet that kubeadm will start if a custom `KubeletConfiguration` API object is passed with a configuration file like so `kubeadm ... --config some-config-file.yaml`.

By calling `kubeadm config print-default --api-objects KubeletConfiguration` you can see all the default values for this structure.

Also have a look at the API reference for the kubelet ComponentConfig for more information on the individual fields.

Workflow when using `kubeadm init`

When you call `kubeadm init`, the kubelet configuration is marshalled to disk at `/var/lib/kubelet/config.yaml`, and also uploaded to a ConfigMap in the cluster. The ConfigMap is named `kubelet-config-1.X`, where `.X` is the minor version of the Kubernetes version you are initializing. A kubelet configuration file is also written to `/etc/kubernetes/kubelet.conf` with the baseline cluster-wide configuration for all kubelets in the cluster. This configuration file points to the client certificates that allow the kubelet to communicate with the API server. This addresses the need to propagate cluster-level configuration to each kubelet.

To address the second pattern of providing instance-specific configuration details, kubeadm writes an environment file to `/var/lib/kubelet/kubeadm-flags.env`, which contains a list of flags to pass to the kubelet when it starts. The flags are presented in the file like this:

```
KUBELET_KUBEADM_ARGS="--flag1=value1 --flag2=value2 ..."
```

In addition to the flags used when starting the kubelet, the file also contains dynamic parameters such as the cgroup driver and whether to use a different CRI runtime socket (`--cri-socket`).

After marshalling these two files to disk, kubeadm attempts to run the following two commands, if you are using systemd:

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

If the reload and restart are successful, the normal `kubeadm init` workflow continues.

Workflow when using `kubeadm join`

When you run `kubeadm join`, kubeadm uses the Bootstrap Token credential perform a TLS bootstrap, which fetches the credential needed to download the `kubelet-config-1.X` ConfigMap and writes it to `/var/lib/kubelet/config.yaml`. The dynamic environment file is generated in exactly the same way as `kubeadm init`.

Next, `kubeadm` runs the following two commands to load the new configuration into the kubelet:

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

After the kubelet loads the new configuration, kubeadm writes the `/etc/kubernetes/bootstrap-kubelet.conf` KubeConfig file, which contains a CA certificate and Bootstrap Token. These are used by the kubelet to perform the TLS Bootstrap and obtain a unique credential, which is stored in `/etc/kubernetes/kubelet.conf`. When this file is written, the kubelet has finished performing the TLS Bootstrap.

The kubelet drop-in file for systemd

The configuration file installed by the kubeadm DEB or RPM package is written to `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` and is used by systemd.

```
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubele
--kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating
the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. I
#the user should use the .NodeRegistration.KubeletExtraArgs object in the configuration file
# KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/default/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_AI
```

This file specifies the default locations for all of the files managed by kubeadm for the kubelet.

- The KubeConfig file to use for the TLS Bootstrap is `/etc/kubernetes/bootstrap-kubelet.conf`, but it is only used if `/etc/kubernetes/kubelet.conf` does not exist.
- The KubeConfig file with the unique kubelet identity is `/etc/kubernetes/kubelet.conf`.
- The file containing the kubelet's ComponentConfig is `/var/lib/kubelet/config.yaml`.
- The dynamic environment file that contains `KUBELET_KUBEADM_ARGS` is sourced from `/var/lib/kubelet/kubeadm-flags.env`.
- The file that can contain user-specified flag overrides with `KUBELET_EXTRA_ARGS` is sourced from `/etc/default/kubelet` (for DEBs), or `/etc/systconfig/kubelet` (for RPMs). `KUBELET_EXTRA_ARGS` is last in the flag chain and has the highest priority in the event of conflicting settings.

Kubernetes binaries and package contents

The DEB and RPM packages shipped with the Kubernetes releases are:

Package name	Description
`kubeadm`	Installs the `/usr/bin/kubeadm` CLI tool and [The kubelet drop-in file(#the-kubelet-drop-
`kubelet`	Installs the `/usr/bin/kubelet` binary.
`kubectl`	Installs the `/usr/bin/kubectl` binary.
`kubernetes-cni`	Installs the official CNI binaries into the `/opt/cni/bin` directory.
`cri-tools`	Installs the `/usr/bin/crictl` binary from https://github.com/kubernetes-incubator/cri-t

Edit This Page

Customizing control plane configuration with kubeadm

The kubeadm configuration exposes the following fields that can override the default flags passed to control plane components such as the APIServer, ControllerManager and Scheduler:

- `APIServerExtraArgs`
- `ControllerManagerExtraArgs`
- `SchedulerExtraArgs`

These fields consist of `key: value` pairs. To override a flag for a control plane component:

1. Add the appropriate field to your configuration.
2. Add the flags to override to the field.

For more details on each field in the configuration you can navigate to our API reference pages.

- APIServer flags
- ControllerManager flags
- Scheduler flags

APIServer flags

For details, see the reference documentation for kube-apiserver.

Example usage:

```
apiVersion: kubeadm.k8s.io/v1alpha3
kind: ClusterConfiguration
kubernetesVersion: v1.12.0
metadata:
  name: 1.12-sample
apiServerExtraArgs:
  advertise-address: 192.168.0.103
  anonymous-auth: false
  enable-admission-plugins: AlwaysPullImages,DefaultStorageClass
  audit-log-path: /home/johndoe/audit.log
```

ControllerManager flags

For details, see the reference documentation for kube-controller-manager.

Example usage:

```
apiVersion: kubeadm.k8s.io/v1alpha3
kind: ClusterConfiguration
kubernetesVersion: v1.12.0
metadata:
  name: 1.12-sample
controllerManagerExtraArgs:
  cluster-signing-key-file: /home/johndoe/keys/ca.key
  bind-address: 0.0.0.0
  deployment-controller-sync-period: 50
```

Scheduler flags

For details, see the reference documentation for kube-scheduler.

Example usage:

```
apiVersion: kubeadm.k8s.io/v1alpha3
kind: ClusterConfiguration
kubernetesVersion: v1.12.0
```

```
metadata:
  name: 1.12-sample
schedulerExtraArgs:
  address: 0.0.0.0
  config: /home/johndoe/schedconfig.yaml
  kubeconfig: /home/johndoe/kubeconfig.yaml
```

Edit This Page

Creating Highly Available Clusters with kubeadm

This page explains two different approaches to setting up a highly available Kubernetes cluster using kubeadm:

- With stacked masters. This approach requires less infrastructure. etcd members and control plane nodes are co-located.
- With an external etcd cluster. This approach requires more infrastructure. The control plane nodes and etcd members are separated.

Your clusters must run Kubernetes version 1.12 or later. You should also be aware that setting up HA clusters with kubeadm is still experimental. You might encounter issues with upgrading your clusters, for example. We encourage you to try either approach, and provide feedback.

> **Caution:** This page does not address running your cluster on a cloud provider. In a cloud environment, neither approach documented here works with Service objects of type LoadBalancer, or with dynamic PersistentVolumes.

Before you begin

For both methods you need this infrastructure:

- Three machines that meet kubeadm's minimum requirements for the masters
- Three machines that meet kubeadm's minimum requirements for the workers
- Full network connectivity between all machines in the cluster (public or private network is fine)
- SSH access from one device to all nodes in the system

- sudo privileges on all machines

For the external etcd cluster only, you also need:

- Three additional machines for etcd members

  Note: The following examples run Calico as the Pod networking provider. If you run another networking provider, make sure to replace any default values as needed.

First steps for both methods

Note: All commands in this guide on any control plane or etcd node should be run as root.

- Find your pod CIDR. For details, see the CNI network documentation. The example uses Calico, so the pod CIDR is `192.168.0.0/16`.

Configure SSH

1. Enable ssh-agent on your main device that has access to all other nodes in the system:

   ```
   eval $(ssh-agent)
   ```

2. Add your SSH identity to the session:

   ```
   ssh-add ~/.ssh/path_to_private_key
   ```

3. SSH between nodes to check that the connection is working correctly.

   - When you SSH to any node, make sure to add the `-A` flag:

     ```
     ssh -A 10.0.0.7
     ```

   - When using sudo on any node, make sure to preserve the environment so SSH forwarding works:

     ```
     sudo -E -s
     ```

Create load balancer for kube-apiserver

Note: There are many configurations for load balancers. The following example is only one option. Your cluster requirements may need a different configuration.

1. Create a kube-apiserver load balancer with a name that resolves to DNS.

- In a cloud environment you should place your control plane nodes behind a TCP forwarding load balancer. This load balancer distributes traffic to all healthy control plane nodes in its target list. The health check for an apiserver is a TCP check on the port the kube-apiserver listens on (default value `:6443`).
- It is not recommended to use an IP address directly in a cloud environment.
- The load balancer must be able to communicate with all control plane nodes on the apiserver port. It must also allow incoming traffic on its listening port.

2. Add the first control plane nodes to the load balancer and test the connection:

   ```
   nc -v LOAD_BALANCER_IP PORT
   ```

   - A connection refused error is expected because the apiserver is not yet running. A timeout, however, means the load balancer cannot communicate with the control plane node. If a timeout occurs, reconfigure the load balancer to communicate with the control plane node.

3. Add the remaining control plane nodes to the load balancer target group.

Stacked control plane nodes

Bootstrap the first stacked control plane node

Note: Optionally replace the string `stable` with a different version of Kubernetes, for example `v1.12.0`.

1. Create a `kubeadm-config.yaml` template file:

   ```
   apiVersion: kubeadm.k8s.io/v1alpha3
   kind: ClusterConfiguration
   kubernetesVersion: stable
   apiServerCertSANs:
   - "LOAD_BALANCER_DNS"
   controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
   etcd:
     local:
       extraArgs:
         name: "CP0_HOSTNAME"
         listen-client-urls: "https://127.0.0.1:2379,https://CP0_IP:2379"
         advertise-client-urls: "https://CP0_IP:2379"
         listen-peer-urls: "https://CP0_IP:2380"
         initial-advertise-peer-urls: "https://CP0_IP:2380"
   ```

```
        initial-cluster: "CP0_HOSTNAME=https://CP0_IP:2380"
      serverCertSANs:
        - CP0_HOSTNAME
        - CP0_IP
      peerCertSANs:
        - CP0_HOSTNAME
        - CP0_IP
  networking:
      # This CIDR is a Calico default. Substitute or remove for your CNI provider.
      podSubnet: "192.168.0.0/16"
```

2. Replace the following variables in the template with the appropriate values for your cluster:
   - `LOAD_BALANCER_DNS`
   - `LOAD_BALANCER_PORT`
   - `CP0_HOSTNAME`
   - `CP0_IP`
3. Run `kubeadm init --config kubeadm-config.yaml`

Copy required files to other control plane nodes

The following certificates and other required files were created when you ran `kubeadm init`. Copy these files to your other control plane nodes:

- `/etc/kubernetes/pki/ca.crt`
- `/etc/kubernetes/pki/ca.key`
- `/etc/kubernetes/pki/sa.key`
- `/etc/kubernetes/pki/sa.pub`
- `/etc/kubernetes/pki/front-proxy-ca.crt`
- `/etc/kubernetes/pki/front-proxy-ca.key`
- `/etc/kubernetes/pki/etcd/ca.crt`
- `/etc/kubernetes/pki/etcd/ca.key`

Copy the admin kubeconfig to the other control plane nodes:

- `/etc/kubernetes/admin.conf`

In the following example, replace `CONTROL_PLANE_IPS` with the IP addresses of the other control plane nodes.

```
USER=ubuntu # customizable
CONTROL_PLANE_IPS="10.0.0.7 10.0.0.8"
for host in ${CONTROL_PLANE_IPS}; do
    scp /etc/kubernetes/pki/ca.crt "${USER}"@$host:
    scp /etc/kubernetes/pki/ca.key "${USER}"@$host:
    scp /etc/kubernetes/pki/sa.key "${USER}"@$host:
    scp /etc/kubernetes/pki/sa.pub "${USER}"@$host:
```

```
    scp /etc/kubernetes/pki/front-proxy-ca.crt "${USER}"@$host:
    scp /etc/kubernetes/pki/front-proxy-ca.key "${USER}"@$host:
    scp /etc/kubernetes/pki/etcd/ca.crt "${USER}"@$host:etcd-ca.crt
    scp /etc/kubernetes/pki/etcd/ca.key "${USER}"@$host:etcd-ca.key
    scp /etc/kubernetes/admin.conf "${USER}"@$host:
done
```

Note: Remember that your config may differ from this example.

Add the second stacked control plane node

1. Create a second, different `kubeadm-config.yaml` template file:

```
apiVersion: kubeadm.k8s.io/v1alpha3
kind: ClusterConfiguration
kubernetesVersion: stable
apiServerCertSANs:
- "LOAD_BALANCER_DNS"
controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
etcd:
  local:
    extraArgs:
      name: "CP1_HOSTNAME"
      listen-client-urls: "https://127.0.0.1:2379,https://CP1_IP:2379"
      advertise-client-urls: "https://CP1_IP:2379"
      listen-peer-urls: "https://CP1_IP:2380"
      initial-advertise-peer-urls: "https://CP1_IP:2380"
      initial-cluster: "CP0_HOSTNAME=https://CP0_IP:2380,CP1_HOSTNAME=https://CP1_IP:23
      initial-cluster-state: existing
    serverCertSANs:
      - CP1_HOSTNAME
      - CP1_IP
    peerCertSANs:
      - CP1_HOSTNAME
      - CP1_IP
networking:
    # This CIDR is a calico default. Substitute or remove for your CNI provider.
    podSubnet: "192.168.0.0/16"
```

2. Replace the following variables in the template with the appropriate values for your cluster:

   - `LOAD_BALANCER_DNS`
   - `LOAD_BALANCER_PORT`
   - `CP0_HOSTNAME`
   - `CP0_IP`
   - `CP1_HOSTNAME`

    - CP1_IP

3. Move the copied files to the correct locations:

   ```
   USER=ubuntu # customizable
   mkdir -p /etc/kubernetes/pki/etcd
   mv /home/${USER}/ca.crt /etc/kubernetes/pki/
   mv /home/${USER}/ca.key /etc/kubernetes/pki/
   mv /home/${USER}/sa.pub /etc/kubernetes/pki/
   mv /home/${USER}/sa.key /etc/kubernetes/pki/
   mv /home/${USER}/front-proxy-ca.crt /etc/kubernetes/pki/
   mv /home/${USER}/front-proxy-ca.key /etc/kubernetes/pki/
   mv /home/${USER}/etcd-ca.crt /etc/kubernetes/pki/etcd/ca.crt
   mv /home/${USER}/etcd-ca.key /etc/kubernetes/pki/etcd/ca.key
   mv /home/${USER}/admin.conf /etc/kubernetes/admin.conf
   ```

4. Run the kubeadm phase commands to bootstrap the kubelet:

   ```
   kubeadm alpha phase certs all --config kubeadm-config.yaml
   kubeadm alpha phase kubelet config write-to-disk --config kubeadm-config.yaml
   kubeadm alpha phase kubelet write-env-file --config kubeadm-config.yaml
   kubeadm alpha phase kubeconfig kubelet --config kubeadm-config.yaml
   systemctl start kubelet
   ```

5. Run the commands to add the node to the etcd cluster:

   ```
   export CP0_IP=10.0.0.7
   export CP0_HOSTNAME=cp0
   export CP1_IP=10.0.0.8
   export CP1_HOSTNAME=cp1

   kubeadm alpha phase etcd local --config kubeadm-config.yaml
   export KUBECONFIG=/etc/kubernetes/admin.conf
   kubectl exec -n kube-system etcd-${CP0_HOSTNAME} -- etcdctl --ca-file /etc/kubernetes/p
   ```

   - This command causes the etcd cluster to become unavailable for a brief period, after the node is added to the running cluster, and before the new node is joined to the etcd cluster.

6. Deploy the control plane components and mark the node as a master:

   ```
   kubeadm alpha phase kubeconfig all --config kubeadm-config.yaml
   kubeadm alpha phase controlplane all --config kubeadm-config.yaml
   kubeadm alpha phase kubelet config annotate-cri --config kubeadm-config.yaml
   kubeadm alpha phase mark-master --config kubeadm-config.yaml
   ```

Add the third stacked control plane node

1. Create a third, different `kubeadm-config.yaml` template file:

```
apiVersion: kubeadm.k8s.io/v1alpha3
kind: ClusterConfiguration
kubernetesVersion: stable
apiServerCertSANs:
- "LOAD_BALANCER_DNS"
controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
etcd:
  local:
    extraArgs:
      name: "CP2_HOSTNAME"
      listen-client-urls: "https://127.0.0.1:2379,https://CP2_IP:2379"
      advertise-client-urls: "https://CP2_IP:2379"
      listen-peer-urls: "https://CP2_IP:2380"
      initial-advertise-peer-urls: "https://CP2_IP:2380"
      initial-cluster: "CP0_HOSTNAME=https://CP0_IP:2380,CP1_HOSTNAME=https://CP1_IP:23
      initial-cluster-state: existing
    serverCertSANs:
      - CP2_HOSTNAME
      - CP2_IP
    peerCertSANs:
      - CP2_HOSTNAME
      - CP2_IP
networking:
    # This CIDR is a calico default. Substitute or remove for your CNI provider.
    podSubnet: "192.168.0.0/16"
```

2. Replace the following variables in the template with the appropriate values for your cluster:

   - LOAD_BALANCER_DNS
   - LOAD_BALANCER_PORT
   - CP0_HOSTNAME
   - CP0_IP
   - CP1_HOSTNAME
   - CP1_IP
   - CP2_HOSTNAME
   - CP2_IP

3. Move the copied files to the correct locations:

```
USER=ubuntu # customizable
mkdir -p /etc/kubernetes/pki/etcd
mv /home/${USER}/ca.crt /etc/kubernetes/pki/
mv /home/${USER}/ca.key /etc/kubernetes/pki/
mv /home/${USER}/sa.pub /etc/kubernetes/pki/
mv /home/${USER}/sa.key /etc/kubernetes/pki/
mv /home/${USER}/front-proxy-ca.crt /etc/kubernetes/pki/
mv /home/${USER}/front-proxy-ca.key /etc/kubernetes/pki/
mv /home/${USER}/etcd-ca.crt /etc/kubernetes/pki/etcd/ca.crt
mv /home/${USER}/etcd-ca.key /etc/kubernetes/pki/etcd/ca.key
mv /home/${USER}/admin.conf /etc/kubernetes/admin.conf
```

4. Run the kubeadm phase commands to bootstrap the kubelet:

```
kubeadm alpha phase certs all --config kubeadm-config.yaml
kubeadm alpha phase kubelet config write-to-disk --config kubeadm-config.yaml
kubeadm alpha phase kubelet write-env-file --config kubeadm-config.yaml
kubeadm alpha phase kubeconfig kubelet --config kubeadm-config.yaml
systemctl start kubelet
```

5. Run the commands to add the node to the etcd cluster:

```
export CP0_IP=10.0.0.7
export CP0_HOSTNAME=cp0
export CP2_IP=10.0.0.9
export CP2_HOSTNAME=cp2

export KUBECONFIG=/etc/kubernetes/admin.conf
kubectl exec -n kube-system etcd-${CP0_HOSTNAME} -- etcdctl --ca-file /etc/kubernetes/p
kubeadm alpha phase etcd local --config kubeadm-config.yaml
```

6. Deploy the control plane components and mark the node as a master:

```
kubeadm alpha phase kubeconfig all --config kubeadm-config.yaml
kubeadm alpha phase controlplane all --config kubeadm-config.yaml
kubeadm alpha phase kubelet config annotate-cri --config kubeadm-config.yaml
kubeadm alpha phase mark-master --config kubeadm-config.yaml
```

External etcd

Set up the cluster

- Follow these instructions to set up the etcd cluster.

Copy required files from an etcd node to all control plane nodes

In the following example, replace `USER` and `CONTROL_PLANE_HOSTS` values with values for your environment.

```
# Make a list of required etcd certificate files
cat << EOF > etcd-pki-files.txt
/etc/kubernetes/pki/etcd/ca.crt
/etc/kubernetes/pki/apiserver-etcd-client.crt
/etc/kubernetes/pki/apiserver-etcd-client.key
EOF
```

```
# create the archive
tar -czf etcd-pki.tar.gz -T etcd-pki-files.txt

# copy the archive to the control plane nodes
USER=ubuntu
CONTROL_PLANE_HOSTS="10.0.0.7 10.0.0.8 10.0.0.9"
for host in $CONTROL_PLANE_HOSTS; do
    scp etcd-pki.tar.gz "${USER}"@$host:
done
```

Set up the first control plane node

> **Note:** Optionally replace the string `stable` with a different version of Kubernetes, for example `v1.11.3`.

1. Extract the etcd certificates

   ```
   mkdir -p /etc/kubernetes/pki
   tar -xzf etcd-pki.tar.gz -C /etc/kubernetes/pki --strip-components=3
   ```

2. Create a `kubeadm-config.yaml`:

   ```
   apiVersion: kubeadm.k8s.io/v1alpha3
   kind: ClusterConfiguration
   kubernetesVersion: stable
   apiServerCertSANs:
   - "LOAD_BALANCER_DNS"
   controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
   etcd:
       external:
           endpoints:
           - https://ETCD_0_IP:2379
           - https://ETCD_1_IP:2379
           - https://ETCD_2_IP:2379
           caFile: /etc/kubernetes/pki/etcd/ca.crt
           certFile: /etc/kubernetes/pki/apiserver-etcd-client.crt
           keyFile: /etc/kubernetes/pki/apiserver-etcd-client.key
   networking:
       # This CIDR is a calico default. Substitute or remove for your CNI provider.
       podSubnet: "192.168.0.0/16"
   ```

3. Replace the following variables in the template with the appropriate values for your cluster:

   - `LOAD_BALANCER_DNS`
   - `LOAD_BALANCER_PORT`
   - `ETCD_0_IP`
   - `ETCD_1_IP`

    - ETCD_2_IP
4. Run `kubeadm init --config kubeadm-config.yaml`
5. Copy the output from the join command

Copy required files to the correct locations

The following pki files were created during the `kubeadm init` step and must be shared with all other control plane nodes.

- `/etc/kubernetes/pki/ca.crt`
- `/etc/kubernetes/pki/ca.key`
- `/etc/kubernetes/pki/sa.key`
- `/etc/kubernetes/pki/sa.pub`
- `/etc/kubernetes/pki/front-proxy-ca.crt`
- `/etc/kubernetes/pki/front-proxy-ca.key`

In the following example, replace the list of `CONTROL_PLANE_IPS` values with the IP addresses of the other control plane nodes.

```
# make a list of required kubernetes certificate files
cat << EOF > certificate_files.txt
/etc/kubernetes/pki/ca.crt
/etc/kubernetes/pki/ca.key
/etc/kubernetes/pki/sa.key
/etc/kubernetes/pki/sa.pub
/etc/kubernetes/pki/front-proxy-ca.crt
/etc/kubernetes/pki/front-proxy-ca.key
EOF

# create the archive
tar -czf control-plane-certificates.tar.gz -T certificate_files.txt

USER=ubuntu # customizable
CONTROL_PLANE_IPS="10.0.0.7 10.0.0.8"
for host in ${CONTROL_PLANE_IPS}; do
    scp control-plane-certificates.tar.gz "${USER}"@$host:
done
```

Set up the other control plane nodes

1. Extract the required certificates

   ```
   mkdir -p /etc/kubernetes/pki
   tar -xzf etcd-pki.tar.gz -C /etc/kubernetes/pki --strip-components 3
   tar -xzf control-plane-certificates.tar.gz -C /etc/kubernetes/pki --strip-components 3
   ```

2. Verify the location of the copied files. Your `/etc/kubernetes` directory should look like this:

    - `/etc/kubernetes/pki/apiserver-etcd-client.crt`
    - `/etc/kubernetes/pki/apiserver-etcd-client.key`
    - `/etc/kubernetes/pki/ca.crt`
    - `/etc/kubernetes/pki/ca.key`
    - `/etc/kubernetes/pki/front-proxy-ca.crt`
    - `/etc/kubernetes/pki/front-proxy-ca.key`
    - `/etc/kubernetes/pki/sa.key`
    - `/etc/kubernetes/pki/sa.pub`
    - `/etc/kubernetes/pki/etcd/ca.crt`

3. Run the copied `kubeadm join` command from above. Add the flag "–experimental-control-plane". The final command will look something like this:

    ```
    kubeadm join ha.k8s.example.com:6443 --token 5ynki1.3erp9i3yo7gqg1nv --discovery-token-
    ```

Common tasks after bootstrapping control plane

Install a pod network

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

Install workers

Each worker node can now be joined to the cluster with the command returned from any of the `kubeadm init` commands.

Edit This Page

Set up a High Availability etcd cluster with kubeadm

Kubeadm defaults to running a single member etcd cluster in a static pod managed by the kubelet on the control plane node. This is not a high availability setup as the etcd cluster contains only one member and cannot sustain any members becoming unavailable. This task walks through the process of creating a high availability etcd cluster of three members that can be used as an external etcd when using kubeadm to set up a kubernetes cluster.

- Before you begin

- Setting up the cluster
- What's next

Before you begin

- Three hosts that can talk to each other over ports 2379 and 2380. This document assumes these default ports. However, they are configurable through the kubeadm config file.
- Each host must have docker, kubelet, and kubeadm installed.
- Some infrastructure to copy files between hosts. For example `ssh` and `scp` can satisfy this requirement.

Setting up the cluster

The general approach is to generate all certs on one node and only distribute the *necessary* files to the other nodes.

> **Note:** kubeadm contains all the necessary crytographic machinery to generate the certificates described below; no other cryptographic tooling is required for this example.

1. Configure the kubelet to be a service manager for etcd.

   Running etcd is simpler than running kubernetes so you must override the kubeadm-provided kubelet unit file by creating a new one with a higher precedence.

   ```
   cat << EOF > /etc/systemd/system/kubelet.service.d/20-etcd-service-manager.conf
   [Service]
   ExecStart=
   ExecStart=/usr/bin/kubelet --address=127.0.0.1 --pod-manifest-path=/etc/kubernetes/mani
   Restart=always
   EOF

   systemctl daemon-reload
   systemctl restart kubelet
   ```

2. Create configuration files for kubeadm.

   Generate one kubeadm configuration file for each host that will have an etcd member running on it using the following script.

   ```
   # Update HOST0, HOST1, and HOST2 with the IPs or resolvable names of your hosts
   export HOST0=10.0.0.6
   export HOST1=10.0.0.7
   export HOST2=10.0.0.8
   ```

```
# Create temp directories to store files that will end up on other hosts.
mkdir -p /tmp/${HOST0}/ /tmp/${HOST1}/ /tmp/${HOST2}/

ETCDHOSTS=(${HOST0} ${HOST1} ${HOST2})
NAMES=("infra0" "infra1" "infra2")

for i in "${!ETCDHOSTS[@]}"; do
HOST=${ETCDHOSTS[$i]}
NAME=${NAMES[$i]}
cat << EOF > /tmp/${HOST}/kubeadmcfg.yaml
apiVersion: "kubeadm.k8s.io/v1alpha3"
kind: ClusterConfiguration
etcd:
    local:
        serverCertSANs:
        - "${HOST}"
        peerCertSANs:
        - "${HOST}"
        extraArgs:
            initial-cluster: infra0=https://${ETCDHOSTS[0]}:2380,infra1=https://${ETCDH
            initial-cluster-state: new
            name: ${NAME}
            listen-peer-urls: https://${HOST}:2380
            listen-client-urls: https://${HOST}:2379
            advertise-client-urls: https://${HOST}:2379
            initial-advertise-peer-urls: https://${HOST}:2380
EOF
done
```

3. Generate the certificate authority

   If you already have a CA then the only action that is copying the CA's `crt` and `key` file to `/etc/kubernetes/pki/etcd/ca.crt` and `/etc/kubernetes/pki/etcd/ca.key`. After those files have been copied, proceed to the next step, "Create certificates for each member".

   If you do not already have a CA then run this command on `$HOST0` (where you generated the configuration files for kubeadm).

   ```
   kubeadm alpha phase certs etcd-ca
   ```

   This creates two files

   - `/etc/kubernetes/pki/etcd/ca.crt`
   - `/etc/kubernetes/pki/etcd/ca.key`

4. Create certificates for each member

   ```
   kubeadm alpha phase certs etcd-server --config=/tmp/${HOST2}/kubeadmcfg.yaml
   kubeadm alpha phase certs etcd-peer --config=/tmp/${HOST2}/kubeadmcfg.yaml
   ```

```
kubeadm alpha phase certs etcd-healthcheck-client --config=/tmp/${HOST2}/kubeadmcfg.yam
kubeadm alpha phase certs apiserver-etcd-client --config=/tmp/${HOST2}/kubeadmcfg.yaml
cp -R /etc/kubernetes/pki /tmp/${HOST2}/
# cleanup non-reusable certificates
find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete

kubeadm alpha phase certs etcd-server --config=/tmp/${HOST1}/kubeadmcfg.yaml
kubeadm alpha phase certs etcd-peer --config=/tmp/${HOST1}/kubeadmcfg.yaml
kubeadm alpha phase certs etcd-healthcheck-client --config=/tmp/${HOST1}/kubeadmcfg.yam
kubeadm alpha phase certs apiserver-etcd-client --config=/tmp/${HOST1}/kubeadmcfg.yaml
cp -R /etc/kubernetes/pki /tmp/${HOST1}/
find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete

kubeadm alpha phase certs etcd-server --config=/tmp/${HOST0}/kubeadmcfg.yaml
kubeadm alpha phase certs etcd-peer --config=/tmp/${HOST0}/kubeadmcfg.yaml
kubeadm alpha phase certs etcd-healthcheck-client --config=/tmp/${HOST0}/kubeadmcfg.yam
kubeadm alpha phase certs apiserver-etcd-client --config=/tmp/${HOST0}/kubeadmcfg.yaml
# No need to move the certs because they are for HOST0

# clean up certs that should not be copied off this host
find /tmp/${HOST2} -name ca.key -type f -delete
find /tmp/${HOST1} -name ca.key -type f -delete
```

5. Copy certificates and kubeadm configs

   The certificates have been generated and now they must be moved to their respective hosts.

```
 USER=ubuntu
 HOST=${HOST1}
 scp -r /tmp/${HOST}/* ${USER}@${HOST}:
 ssh ${USER}@${HOST}
 USER@HOST $ sudo -Es
 root@HOST $ chown -R root:root pki
 root@HOST $ mv pki /etc/kubernetes/
```

6. Ensure all expected files exist

   The complete list of required files on `$HOST0` is:

```
/tmp/${HOST0}
  kubeadmcfg.yaml
---
/etc/kubernetes/pki
  apiserver-etcd-client.crt
  apiserver-etcd-client.key
  etcd
      ca.crt
      ca.key
```

```
        healthcheck-client.crt
        healthcheck-client.key
        peer.crt
        peer.key
        server.crt
        server.key
```

On $HOST1:

```
$HOME
  kubeadmcfg.yaml
---
/etc/kubernetes/pki
  apiserver-etcd-client.crt
  apiserver-etcd-client.key
  etcd
      ca.crt
      healthcheck-client.crt
      healthcheck-client.key
      peer.crt
      peer.key
      server.crt
      server.key
```

On $HOST2

```
$HOME
  kubeadmcfg.yaml
---
/etc/kubernetes/pki
  apiserver-etcd-client.crt
  apiserver-etcd-client.key
  etcd
      ca.crt
      healthcheck-client.crt
      healthcheck-client.key
      peer.crt
      peer.key
      server.crt
      server.key
```

7. Create the static pod manifests

   Now that the certificates and configs are in place it's time to create the manifests. On each host run the `kubeadm` command to generate a static manifest for etcd.

```
root@HOST0 $ kubeadm alpha phase etcd local --config=/tmp/${HOST0}/kubeadmcfg.yaml
root@HOST1 $ kubeadm alpha phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
```

```
root@HOST2 $ kubeadm alpha phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
```

8. Optional: Check the cluster health

```
docker run --rm -it \
--net host \
-v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd:v3.2.18 etcdctl \
--cert-file /etc/kubernetes/pki/etcd/peer.crt \
--key-file /etc/kubernetes/pki/etcd/peer.key \
--ca-file /etc/kubernetes/pki/etcd/ca.crt \
--endpoints https://${HOST0}:2379 cluster-health
...
cluster is healthy
```

What's next

Once your have a working 3 member etcd cluster, you can continue setting up a highly available control plane using the external etcd method with kubeadm.

Edit This Page

Troubleshooting kubeadm

As with any program, you might run into an error installing or running kubeadm. This page lists some common failure scenarios and have provided steps that can help you understand and fix the problem.

If your problem is not listed below, please follow the following steps:

- If you think your problem is a bug with kubeadm:
  - Go to github.com/kubernetes/kubeadm and search for existing issues.
  - If no issue exists, please open one and follow the issue template.
- If you are unsure about how kubeadm works, you can ask on Slack in #kubeadm, or open a question on StackOverflow. Please include relevant tags like `#kubernetes` and `#kubeadm` so folks can help you.

- `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
- `coredns` pods have `CrashLoopBackOff` or `Error` state

`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 and nothing happens to the DNS server, 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.
- If you install a version of Docker older than 1.12.1, remove the `MountFlags=slave` option when booting `dockerd` with `systemd` and restart `docker`. You can see the MountFlags in `/usr/lib/systemd/system/docker.service`. MountFlags can interfere with volumes mounted by Kubernetes, and put the Pods in `CrashLoopBackOff` state. The error happens when Kubernetes does not find `var/run/secrets/kubernetes.io/serviceaccount` files.

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

This is **expected** and part of the design. kubeadm is network provider-agnostic, so the admin should install the pod network solution of choice. You have to install a Pod Network before CoreDNS may deployed fully. Hence the `Pending` state before the network is set up.

`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 kubeadm `NodeRegistrationOptions` structure 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
)"
```

`coredns` pods have `CrashLoopBackOff` or `Error` state

If you have nodes that are running SELinux with an older version of Docker you might experience a scenario where the `coredns` pods are not starting. To solve that you can try one of the following options:

- Upgrade to a newer version of Docker.
- Disable SELinux.
- Modify the `coredns` deployment to set `allowPrivilegeEscalation` to `true`:

```
kubectl -n kube-system get deployment coredns -o yaml | \
  sed 's/allowPrivilegeEscalation: false/allowPrivilegeEscalation: true/g' | \
  kubectl apply -f -
```

Another cause for CoreDNS to have `CrashLoopBackOff` is when a CoreDNS Pod deployed in Kubernetes detects a loop. A number of workarounds are available to avoid Kubernetes trying to restart the CoreDNS Pod every time CoreDNS detects the loop and exits.

> **Warning:** Disabling SELinux or setting `allowPrivilegeEscalation` to `true` can compromise the security of your cluster.

Edit This Page

Running Kubernetes on Multiple Clouds with Stackpoint.io

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
- GCE
- Google Kubernetes Engine
- DigitalOcean
- Microsoft Azure
- Packet

AWS

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

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Amazon Web Services (AWS).

2. Configure Your Provider

   a. 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.

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Google Compute Engine (GCE).

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Google Kubernetes Engine.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

c. Click to select DigitalOcean.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Microsoft Azure.

2. Configure Your Provider

   a. 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.

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Packet.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

Edit This Page

Running Kubernetes on AWS EC2

This page describes how to install a Kubernetes cluster on AWS.

- Before you begin
- Getting started with your cluster
- Scaling the cluster
- Tearing down the cluster
- Support Level
- Further reading

Before you begin

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

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:

```
# macOS
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.

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.

Edit This Page

Running Kubernetes on Azure

- - Azure Kubernetes Service (AKS)
  - Custom Deployments: ACS-Engine
  - CoreOS Tectonic for Azure

Azure Kubernetes Service (AKS)

The Azure Kubernetes Service offers simple deployments for Kubernetes clusters.

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

Microsoft Azure Kubernetes Service

Custom Deployments: ACS-Engine

The core of the Azure Kubernetes 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 Kubernetes 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 Kubernetes Service.

The input to ACS-Engine is similar to the ARM template syntax used to deploy a cluster directly with the Azure Kubernetes 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.

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 macOS)
- 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

~~~
authentication certificates for the particular cluster.  The configuration file is
written to the  ```${CLC_CLUSTER_HOME}/kube``` directory

```shell
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 macOS.

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

~~~
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 mini
* ```hosts/```: hosts files listing access information for the ansible playbooks
* ```kube/```: ```kubectl``` configuration files, and the basic-authentication password for
* ```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:

```shell
~~~

```
kubectl get nodes
kubectl get --all-namespaces pods
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/kuberne` 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.

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).

- Before you begin
- Starting a cluster
- Installing the Kubernetes command line tools on your workstation
- Getting started with your cluster
- Tearing down the cluster
- Customizing
- Troubleshooting
- Support Level
- Further reading

Before you begin

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 Forum, 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          TYPE             CLUSTER_IP       EXTERNAL_IP       PORT(S)
default       kubernetes    ClusterIP        10.0.0.1         <none>            443/TCP
kube-system   kube-dns      ClusterIP        10.0.0.2         <none>            53/TCP,53/UI
kube-system   kube-ui       ClusterIP        10.0.0.3         <none>            80/TCP
...
```

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   coredns-5f4fbb68df-mc8z8                      1/1       Running   0
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-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.

Edit This Page

Running Kubernetes on Multiple Clouds with Stackpoint.io

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
- GCE
- Google Kubernetes Engine
- DigitalOcean
- Microsoft Azure
- Packet

AWS

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

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Amazon Web Services (AWS).

2. Configure Your Provider

   a. 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.

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Google Compute Engine (GCE).

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Google Kubernetes Engine.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select DigitalOcean.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Microsoft Azure.

2. Configure Your Provider

   a. 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.

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

3. Configure Your Cluster

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

4. Run 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.

1. Choose a Provider

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

   b. Click **+ADD A CLUSTER NOW**.

   c. Click to select Packet.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run 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.

Edit This Page

CoreOS on AWS or GCE

There are multiple guides on running Kubernetes with CoreOS.

- Official CoreOS Guides
- Community Guides
- Support Level

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 macOS App**

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

- **Multi-node cluster with Vagrant and fleet units using a small macOS App**

  Guide to running a single master, multi-worker cluster controlled by an macOS 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, @AntonioM

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

Edit This Page

CoreOS on AWS or GCE

There are multiple guides on running Kubernetes with CoreOS.

- Official CoreOS Guides
- Community Guides
- Support Level

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 macOS App**

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

- **Multi-node cluster with Vagrant and fleet units using a small macOS App**

  Guide to running a single master, multi-worker cluster controlled by an macOS 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, @AntonioM

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

Edit This Page

Installing Kubernetes On-premises/Cloud Providers with Kubespray

This quickstart helps to install a Kubernetes cluster hosted on GCE, Azure, OpenStack, AWS, vSphere, Oracle Cloud Infrastructure (Experimental) 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
  - Container Linux by CoreOS
  - Debian Jessie, Stretch, Wheezy
  - Ubuntu 16.04, 18.04
  - CentOS/RHEL 7
  - Fedora/CentOS Atomic
  - openSUSE Leap 42.3/Tumbleweed
- continuous integration tests

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

- Creating a cluster
- Cluster operations
- Cleanup
- Feedback

- What's next

Creating a cluster

(1/5) Meet the underlay requirements

Provision servers with the following requirements:

- **Ansible v2.5 (or newer) and python-netaddr is installed on the machine that will run Ansible commands**
- **Jinja 2.9 (or newer) is required to run the Ansible Playbooks**
- The target servers must have **access to the Internet** in order to pull docker images
- The target servers are configured to allow **IPv4 forwarding**
- **Your ssh key must be copied** to all the servers part of your inventory
- The **firewalls are not managed**, you'll need to implement your own rules the way you used to. in order to avoid any issue during deployment you should disable your firewall
- If kubespray is ran from non-root user account, correct privilege escalation method should be configured in the target servers. Then the `ansible_become` flag or command parameters `--become` or `-b` should be specified

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:

- Choice deployment mode: kubeadm or non-kubeadm
- 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
  - docker
  - rkt
  - cri-o
- Certificate generation methods (**Vault being discontinued**)

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".

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

What's next

Check out planned work on Kubespray's roadmap.

Edit This Page

Installing Kubernetes on AWS with kops

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
- Multiple OS support (Debian, Ubuntu 16.04 supported, CentOS & RHEL, Amazon Linux and CoreOS) - see the images.md
- High-Availability support - see the high_availability.md
- Can directly provision, or generate terraform manifests - see the terraform.md

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

- Cleanup
- Feedback
- What's next

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/1.10.0/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/1.10.0/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.

Cleanup

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

Feedback

- Slack Channel: #kops-users
- GitHub Issues

What's next

- Learn more about Kubernetes concepts and `kubectl`.
- Learn about `kops` advanced usage
- See the `kops` docs section for tutorials, best practices and advanced configuration options.

Edit This Page

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
- Using the oVirt Cloud Provider
- oVirt Cloud Provider Screencast
- Support Level

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
```

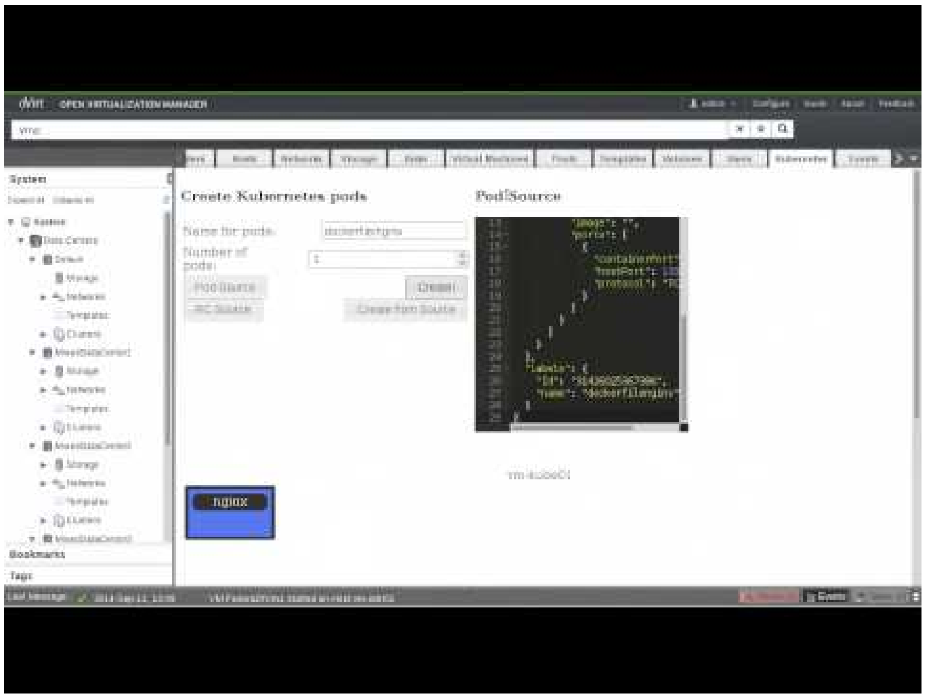
In the above example all the VMs tagged with the `kubernetes` label will be reported as nodes to Kubernetes.

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.

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
- 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.

```
ssh -i ~/.ssh/id_rsa_k8s core@<master IP>
```

And you can list the machines in your cluster:

```
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.

Edit This Page

Kubernetes on DC/OS

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

Official Mesosphere Guide

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

Edit This Page

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
- Using the oVirt Cloud Provider
- oVirt Cloud Provider Screencast
- Support Level

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
```

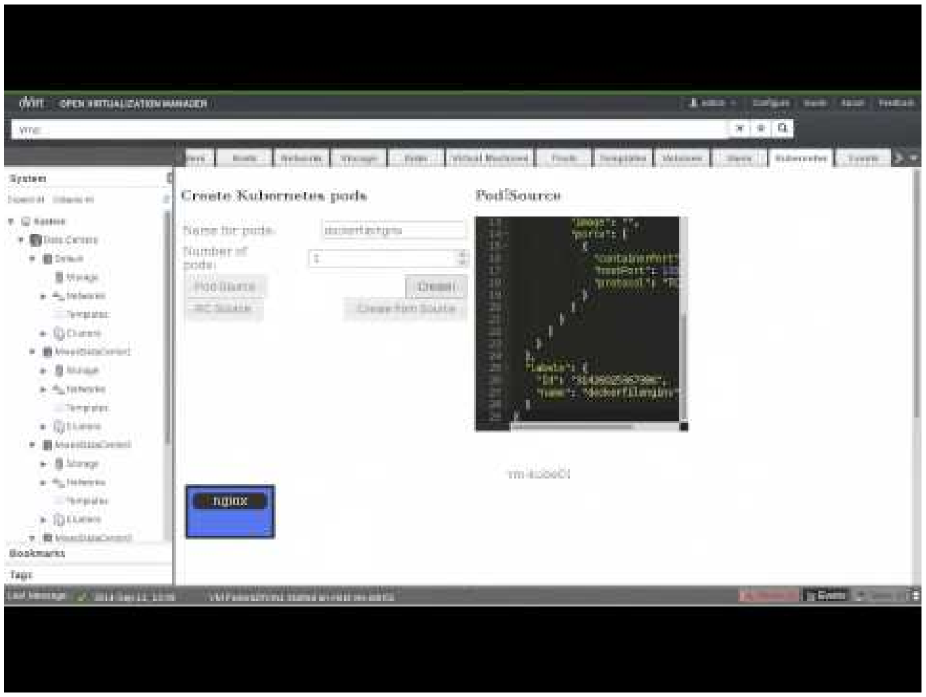
In the above example all the VMs tagged with the `kubernetes` label will be reported as nodes to Kubernetes.

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.

Edit This Page

Building Large Clusters

Support

At v1.12, 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:

  On Google Kubernetes Engine, the size of the master node adjusts automatically based on the size of your cluster. For more information, see this blog post.

  On AWS, master node sizes are currently set at cluster startup time and do not change, even if you later scale your cluster up or down by 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`.

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).

    * Pods are spread across zones
    * Shutting down the cluster

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 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.

Volume limitations

The following limitations are addressed with topology-aware volume binding.

- 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                     ROLES    AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>   6m    v1.12.0          beta.kub
kubernetes-minion-87j9   Ready                      <none>   6m    v1.12.0          beta.kub
kubernetes-minion-9vlv   Ready                      <none>   6m    v1.12.0          beta.kub
kubernetes-minion-a12q   Ready                      <none>   6m    v1.12.0          beta.kub
```

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                     ROLES    AGE   VERSION           LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>   16m   v1.12.0           beta.ku
kubernetes-minion-281d   Ready                      <none>   2m    v1.12.0           beta.ku
kubernetes-minion-87j9   Ready                      <none>   16m   v1.12.0           beta.ku
kubernetes-minion-9vlv   Ready                      <none>   16m   v1.12.0           beta.ku
kubernetes-minion-a12q   Ready                      <none>   17m   v1.12.0           beta.ku
kubernetes-minion-pp2f   Ready                      <none>   2m    v1.12.0           beta.ku
kubernetes-minion-wf8i   Ready                      <none>   2m    v1.12.0           beta.ku
```

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   RECLAIM POLICY   STATUS    CLAIM            STORAGEC
pv-gce-mj4gm   5Gi        RWO           Retain           Bound     default/claim1   manual
```

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    ROLES    AGE    VERSION          LABELS
kubernetes-minion-9vlv   Ready     <none>   34m    v1.12.0          beta.kubernetes.io/insta
kubernetes-minion-281d   Ready     <none>   20m    v1.12.0          beta.kubernetes.io/insta
kubernetes-minion-olsh   Ready     <none>   3m     v1.12.0          beta.kubernetes.io/insta
```

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 | uniq
  "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
```

Edit This Page

CRI installation

Since v1.6.0, Kubernetes has enabled the use of CRI, Container Runtime Interface, by default. This page contains installation instruction for various runtimes.

- Docker
- CRI-O
- Containerd
- Other CRI runtimes: rktlet and frakti

Please proceed with executing the following commands based on your OS as root. You may become the root user by executing `sudo -i` after SSH-ing to each host.

Docker

On each of your machines, install Docker. Version 18.06 is recommended, but 1.11, 1.12, 1.13 and 17.03 are known to work as well. Keep track of the latest verified Docker version in the Kubernetes release notes.

Use the following commands to install Docker on your system:

- Ubuntu 16.04
- CentOS/RHEL 7.4+

```
# Install Docker from Ubuntu's repositories:
apt-get update
apt-get install -y docker.io

# or install Docker CE 18.06 from Docker's repositories for Ubuntu or Debian:

## Install prerequisites.
apt-get update && apt-get install apt-transport-https ca-certificates curl software-properti

## Download GPG key.
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

## Add docker apt repository.
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

## Install docker.
apt-get update && apt-get install docker-ce=18.06.0~ce~3-0~ubuntu

# Setup daemon.
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
```

```
EOF

mkdir -p /etc/systemd/system/docker.service.d

# Restart docker.
systemctl daemon-reload
systemctl restart docker


# Install Docker from CentOS/RHEL repository:
yum install -y docker

# or install Docker CE 18.06 from Docker's CentOS repositories:

## Install prerequisites.
yum install yum-utils device-mapper-persistent-data lvm2

## Add docker repository.
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

## Install docker.
yum update && yum install docker-ce-18.06.1.ce

## Create /etc/docker directory.
mkdir /etc/docker

# Setup daemon.
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
EOF

mkdir -p /etc/systemd/system/docker.service.d

# Restart docker.
```

```
systemctl daemon-reload
systemctl restart docker
```

Refer to the official Docker installation guides for more information.

CRI-O

This section contains the necessary steps to install `CRI-O` as CRI runtime.

Use the following commands to install CRI-O on your system:

Prerequisites

```
modprobe overlay
modprobe br_netfilter

# Setup required sysctl params, these persist across reboots.
cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sysctl --system
```

- Ubuntu 16.04
- CentOS/RHEL 7.4+

```
# Install prerequisites
apt-get update
apt-get install software-properties-common

add-apt-repository ppa:projectatomic/ppa
apt-get update

# Install CRI-O
apt-get install cri-o-1.11
```

```
# Install prerequisites
yum-config-manager --add-repo=https://cbs.centos.org/repos/paas7-crio-311-candidate/x86_64/c

# Install CRI-O
yum install --nogpgcheck cri-o
```

Start CRI-O

```
systemctl start crio
```

Refer to the CRI-O installation guide for more information.

Containerd

This section contains the necessary steps to use `containerd` as CRI runtime.

Use the following commands to install Containerd on your system:

Prerequisites

```
modprobe overlay
modprobe br_netfilter

# Setup required sysctl params, these persist across reboots.
cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sysctl --system
```

- Ubuntu 16.04+
- CentOS/RHEL 7.4+

```
apt-get install -y libseccomp2
```

```
yum install -y libseccomp
```

Install containerd

Containerd releases are published regularly, the values below are hardcoded to the latest version available at the time of writing. Please check for newer versions and hashes here.

```
# Export required environment variables.
export CONTAINERD_VERSION="1.1.2"
export CONTAINERD_SHA256="d4ed54891e90a5d1a45e3e96464e2e8a4770cd380c21285ef5c9895c40549218"

# Download containerd tar.
wget https://storage.googleapis.com/cri-containerd-release/cri-containerd-${CONTAINERD_VERS
```

```
# Check hash.
echo "${CONTAINERD_SHA256} cri-containerd-${CONTAINERD_VERSION}.linux-amd64.tar.gz" | sha25

# Unpack.
tar --no-overwrite-dir -C / -xzf cri-containerd-${CONTAINERD_VERSION}.linux-amd64.tar.gz

# Start containerd.
systemctl start containerd
```

Other CRI runtimes: rktlet and frakti

Refer to the Frakti QuickStart guide and Rktlet Getting Started guide for more information.

Edit This Page

Installing Kubernetes with Digital Rebar Provision (DRP) via KRIB

Overview

This guide helps to install a Kubernetes cluster hosted on bare metal with Digital Rebar Provision using only its Content packages and *kubeadm*.

Digital Rebar Provision (DRP) is an integrated Golang DHCP, bare metal provisioning (PXE/iPXE) and workflow automation platform. While DRP can be used to invoke kubespray, it also offers a self-contained Kubernetes installation known as KRIB (Kubernetes Rebar Integrated Bootstrap).

> **Note:** KRIB is not a *stand-alone* installer: Digital Rebar templates drive a standard *kubeadm* configuration that manages the Kuber-

netes installation with the Digital Rebar cluster pattern to elect leaders *without external supervision*.

KRIB features:

- zero-touch, self-configuring cluster without pre-configuration or inventory
- very fast, no-ssh required automation
- bare metal, on-premises focused platform
- highly available cluster options (including splitting etcd from the controllers)
- dynamic generation of a TLS infrastructure
- composable attributes and automatic detection of hardware by profile
- options for persistent, immutable and image-based deployments
- support for Ubuntu 18.04, CentOS/RHEL 7 and others

Creating a cluster

Review Digital Rebar documentation for details about installing the platform.

The Digital Rebar Provision Golang binary should be installed on a Linux-like system with 16 GB of RAM or larger (Packet.net Tiny and Rasberry Pi are also acceptable).

(1/5) Discover servers

Following the Digital Rebar installation, allow one or more servers to boot through the *Sledgehammer* discovery process to register with the API. This will automatically install the Digital Rebar runner and to allow for next steps.

(2/5) Install KRIB Content and Certificate Plugin

Upload the KRIB Content bundle (or build from source) and the Cert Plugin for your DRP platform (e.g.: amd64 Linux v2.4.0). Both are freely available via the RackN UX.

(3/5) Start your cluster deployment

> **Note:** KRIB documentation is dynamically generated from the source and will be more up to date than this guide.

Following the KRIB documentation, create a Profile for your cluster and assign your target servers into the cluster Profile. The Profile must set `krib\cluster-name` and `etcd\cluster-name` Params to be the name of the Profile. Cluster configuration choices can be made by adding additional Params to the Profile; however, safe defaults are provided for all Params.

Once all target servers are assigned to the cluster Profile, start a KRIB installation Workflow by assigning one of the included Workflows to all cluster servers. For example, selecting `krib-live-cluster` will perform an immutable deployment into the Sledgehammer discovery operating system. You may use one of the pre-created read-only Workflows or choose to build your own custom variation.

For basic installs, no further action is required. Advanced users may choose to assign the controllers, etcd servers or other configuration values in the relevant Params.

(4/5) Monitor your cluster deployment

Digital Rebar Provision provides detailed logging and live updates during the installation process. Workflow events are available via a websocket connection or monitoring the Jobs list.

During the installation, KRIB writes cluster configuration data back into the cluster Profile.

(5/5) Access your cluster

The cluster is available for access via *kubectl* once the `krib/cluster-admin-conf` Param has been set. This Param contains the `kubeconfig` information necessary to access the cluster.

For example, if you named the cluster Profile `krib` then the following commands would allow you to connect to the installed cluster from your local terminal.

::

```
drpcli profiles get krib params krib/cluster-admin-conf > admin.conf
export KUBECONFIG=admin.conf
kubectl get nodes
```

The installation continues after the `krib/cluster-admin-conf` is set to install the Kubernetes UI and Helm. You may interact with the cluster as soon as the `admin.conf` file is available.

Cluster operations

KRIB provides additional Workflows to manage your cluster. Please see the KRIB documentation for an updated list of advanced cluster operations.

Scale your cluster

You can add servers into your cluster by adding the cluster Profile to the server and running the appropriate Workflow.

Cleanup your cluster (for developers)

You can reset your cluster and wipe out all configuration and TLS certificates using the `krib-reset-cluster` Workflow on any of the servers in the cluster.

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

Feedback

- Slack Channel: #community
- GitHub Issues

Edit This Page

PKI Certificates and Requirements

Kubernetes requires PKI certificates for authentication over TLS. If you install Kubernetes with kubeadm, the certificates that your cluster requires are automatically generated. You can also generate your own certificates – for example, to keep your private keys more secure by not storing them on the API server. This page explains the certificates that your cluster requires.

- How certificates are used by your cluster
- Where certificates are stored
- Configure certificates manually
- Configure certificates for user accounts

How certificates are used by your cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd

- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

  Note: `front-proxy` certificates are required only if you run kube-proxy to support an extension API server.

etcd also implements mutual TLS to authenticate clients and peers.

Where certificates are stored

If you install Kubernetes with kubeadm, certificates are stored in `/etc/kubernetes/pki`. All paths in this documentation are relative to that directory.

Configure certificates manually

If you don't want kubeadm to generate the required certificates, you can create them in either of the following ways.

Single root CA

You can create a single root CA, controlled by an administrator. This root CA can then create multiple intermediate CAs, and delegate all further creation to Kubernetes itself.

Required CAs:

path	Default CN	description
ca.crt,key	kubernetes-ca	Kubernetes general CA
etcd/ca.crt,key	etcd-ca	For all etcd-related functions
front-proxy-ca.crt,key	kubernetes-front-proxy-ca	For the front-end proxy

All certificates

If you don't wish to copy these private keys to your API servers, you can generate all certificates yourself.

Required certificates:

Default CN	Parent CA	O (in Subject)	kind	host
kube-etcd	etcd-ca		server, client [1][etcdbug]	`loca`
kube-etcd-peer	etcd-ca		server, client	`<hos`

Default CN	Parent CA	O (in Subject)	kind	host
kube-etcd-healthcheck-client	etcd-ca		client	
kube-apiserver-etcd-client	etcd-ca	system:masters	client	
kube-apiserver	kubernetes-ca		server	`<hos`
kube-apiserver-kubelet-client	kubernetes-ca	system:masters	client	
front-proxy-client	kubernetes-front-proxy-ca		client	

[1]: `kubernetes`, `kubernetes.default`, `kubernetes.default.svc`, `kubernetes.default.svc.cluster`, `kubernetes.default.svc.cluster.local`

where `kind` maps to one or more of the x509 key usage types:

kind	Key usage
server	digital signature, key encipherment, server auth
client	digital signature, key encipherment, client auth

Certificate paths

Certificates should be placed in a recommended path (as used by kubeadm). Paths should be specified using the given argument regardless of location.

Default CN	recommend key path	recommended cert path	command	ke
etcd-ca		etcd/ca.crt	kube-apiserver	
etcd-client	apiserver-etcd-client.crt	apiserver-etcd-client.crt	kube-apiserver	–e
kubernetes-ca		ca.crt	kube-apiserver	–c
kube-apiserver	apiserver.crt	apiserver.key	kube-apiserver	–t
apiserver-kubelet-client	apiserver-kubelet-client.crt		kube-apiserver	–k
front-proxy-client	front-proxy-client.key	front-proxy-client.crt	kube-apiserver	–p
etcd-ca		etcd/ca.crt	etcd	
kube-etcd		etcd/server.crt	etcd	
kube-etcd-peer	etcd/peer.key	etcd/peer.crt	etcd	–p
etcd-ca		etcd/ca.crt	etcdctl[2]	
kube-etcd-healthcheck-client	etcd/healthcheck-client.key	etcd/healthcheck-client.crt	etcdctl[2]	–k

[2]: For a liveness probe, if self-hosted

Configure certificates for user accounts

You must manually configure these administrator account and service accounts:

filename	credential name	Default CN	O (in Subject)
admin.conf	default-admin	kubernetes-admin	system:masters
kubelet.conf	default-auth	system:node:`<nodename>`	system:nodes
controller-manager.conf	default-controller-manager	system:kube-controller-manager	
scheduler.conf	default-manager	system:kube-scheduler	

1. For each config, generate an x509 cert/key pair with the given CN and O.
2. Run `kubectl` as follows for each config:

```
KUBECONFIG=<filename> kubectl config set-cluster default-cluster --server=https://<host ip>:
KUBECONFIG=<filename> kubectl config set-credentials <credential-name> --client-key <path-to
KUBECONFIG=<filename> kubectl config set-context default-system --cluster default-cluster --
KUBECONFIG=<filename> kubectl config use-context default-system
```

These files are used as follows:

filename	command	comment
admin.conf	kubectl	Configures administrator user for the cluster
kubelet.conf	kubelet	One required for each node in the cluster.
controller-manager.conf	kube-controller-manager	Must be added to manifest in `manifests/kube-control`
scheduler.conf	kube-scheduler	Must be added to manifest in `manifests/kube-schedul`

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.

- Additional Links
- Community

Minikube Features

- Minikube supports Kubernetes features such as:
  - DNS
  - NodePorts
  - ConfigMaps and Secrets
  - Dashboards
  - Container Runtime: Docker, rkt, CRI-O and containerd
  - 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
- kvm2 (driver installation)
- 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.apps/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)


Hostname: hello-minikube-7c77b68cff-8wdzq

Pod Information:
    -no pod information available-

Server values:
    server_version=nginx: 1.13.3 - lua: 10008

Request Information:
    client_address=172.17.0.1
    method=GET
    real path=/
    query=
    request_version=1.1
    request_scheme=http
    request_uri=http://192.168.99.100:8080/

Request Headers:
    accept=*/*
    host=192.168.99.100:30674
    user-agent=curl/7.47.0

Request Body:
    -no body in request-


$ kubectl delete services hello-minikube
service "hello-minikube" deleted
$ kubectl delete deployment hello-minikube
deployment.extensions "hello-minikube" deleted
$ minikube stop
```

```
Stopping local Kubernetes cluster...
Stopping "minikube"...
```

Alternative Container Runtimes

containerd

To use containerd as the container runtime, run:

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

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.

Use local images by re-using 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, 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 to the `minikube start` command:

```
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

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/minikube`
- `/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	macOS	/Users	/Users
VirtualBox	Windows	C://Users	/c/Users
VMware Fusion	macOS	/Users	/Users
Xhyve	macOS	/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 kubeadm to provision a Kubernetes 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: ".

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.