Edit This Page

Getting started

This section covers different options to set up and run Kubernetes.

Different Kubernetes solutions meet different requirements: ease of maintenance, security, control, available resources, and expertise required to operate and manage a cluster.

You can deploy a Kubernetes cluster on a local machine, cloud, on-prem datacenter; or choose a managed Kubernetes cluster. You can also create custom solutions across a wide range of cloud providers, or bare metal environments.

More simply, you can create a Kubernetes cluster in learning and production environments.

- Learning environment
- Production environment

Learning environment

If you're learning Kubernetes, use the Docker-based solutions: tools supported by the Kubernetes community, or tools in the ecosystem to set up a Kubernetes cluster on a local machine.

Community	Ecosystem
Minikube	CDK on LXD
kind (Kubernetes IN Docker)	Docker Desktop
	Minishift
	MicroK8s
	IBM Cloud Private-CE (Community Edition)
	IBM Cloud Private-CE (Community Edition) on Linux Containers
	k3s
	Ubuntu on LXD

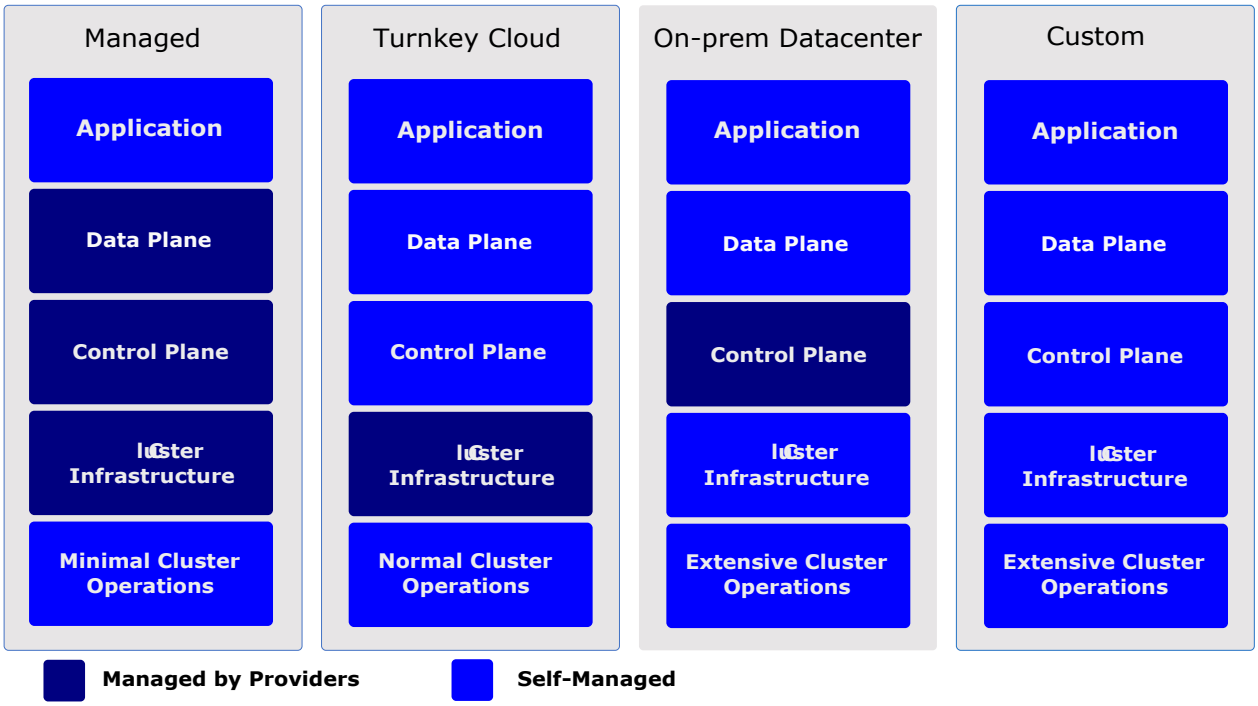
Production environment

When evaluating a solution for a production environment, consider which aspects of operating a Kubernetes cluster (or *abstractions*) you want to manage yourself or offload to a provider.

Some possible abstractions of a Kubernetes cluster are applicationsThe layer where various containerized applications run. , data planeThe layer that provides capacity such as CPU, memory, network, and storage so that the containers can run and connect to a network. , control planeThe container orchestration layer that exposes the API and interfaces to define, deploy, and manage the lifecycle of containers. , cluster infrastructureThe infrastructure layer provides and maintains VMs, networking, security groups and others. , and cluster operationsActivities such as upgrading the clusters, implementing security, storage, ingress, networking, logging and monitoring, and other operations involved in managing a Kubernetes cluster. .

The following diagram lists the possible abstractions of a Kubernetes cluster and whether an abstraction is self-managed or managed by a provider.

Production environment solutions



The following production environment solutions table lists the providers and the solutions that they offer.

Providers	Managed	Turnkey cloud	On-prem datacenter	Custom (cloud)	Custom (On-premises VMs)	Custom (Bare Metal)
Agile Stacks		âœ”	âœ”			
Alibaba Cloud		âœ”				
Amazon	Amazon EKS	Amazon EC2				
AppsCode	âœ”					
APPUiOÂ	âœ”	âœ”	âœ”			
Banzai Cloud Pipeline Kubernetes Engine (PKE)		âœ”		âœ”	âœ”	âœ”
CenturyLink Cloud		âœ”				
Cisco Container Platform			âœ”			
Cloud Foundry Container Runtime (CFCR)				âœ”	âœ”	
CloudStack					âœ”	
Canonical	âœ”	âœ”	âœ”	âœ”	âœ”	âœ”
Containership	âœ”	âœ”				
Digital Rebar						âœ”
DigitalOcean	âœ”					

Providers	Managed	Turnkey cloud	On-prem datacenter	Custom (cloud)	Custom (On-premises VMs)	Custom (Bare Metal)
Docker Enterprise		âœ”	âœ”			âœ”
Fedora (Multi Node)Â					âœ”	âœ”
Fedora (Single Node)Â						âœ”
Gardener	âœ”	âœ”	âœ” (via OpenStack)	âœ”		
Giant Swarm	âœ”	âœ”	âœ”			
Google	Google Kubernetes Engine (GKE)	Google Compute Engine (GCE)	GKE On-Prem			
IBM	IBM Cloud Kubernetes Service		IBM Cloud Private			
Ionos	Ionos Managed Kubernetes	Ionos Enterprise Cloud				
Kontena Pharos		âœ”	âœ”			
Kubermatic	âœ”	âœ”	âœ”			
KubeSail	âœ”					
Kubespray				âœ”	âœ”	âœ”
Kublr	âœ”	âœ”	âœ”	âœ”	âœ”	âœ”
Microsoft Azure	Azure Kubernetes Service (AKS)					
Mirantis Cloud Platform			âœ”			
Nirmata		âœ”	âœ”			
Nutanix	Nutanix Karbon	Nutanix Karbon			Nutanix AHV	
OpenShift	OpenShift Dedicated and OpenShift Online		OpenShift Container Platform		OpenShift Container Platform	OpenShift Container Platform
Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE)	âœ”	âœ”				
oVirt					âœ”	

Providers	Managed	Turnkey cloud	On-prem datacenter	Custom (cloud)	Custom (On-premises VMs)	Custom (Bare Metal)
Pivotal		Enterprise Pivotal Container Service (PKS)	Enterprise Pivotal Container Service (PKS)			
Platform9	âœ”	âœ”	âœ”		âœ”	âœ”
Rancher		Rancher 2.x		Rancher Kubernetes Engine (RKE)		k3s
StackPointÂ	âœ”	âœ”				
Supergiant		âœ”				
SUSE		âœ”				
SysEleven	âœ”					
Tencent Cloud	Tencent Kubernetes Engine	âœ”	âœ”			âœ”
VEXXHOST	âœ”	âœ”				
VMware	VMware Cloud PKS	VMware Enterprise PKS	VMware Enterprise PKS	VMware Essential PKS		VMware Essential PKS
Z.A.R.V.I.S.	âœ”					

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Edit This Page
Page last modified on September 09, 2019 at 10:31 AM PST by Update Gardener info (#16031) (Page History)

Edit This Page

Kubernetes version and version skew support policy

This document describes the maximum version skew supported between various Kubernetes components. Specific cluster deployment tools may place additional restrictions on version skew.

Supported versions

Kubernetes versions are expressed as **x.y.z**, where **x** is the major version, **y** is the minor version, and **z** is the patch version, following Semantic Versioning terminology. For more information, see Kubernetes Release Versioning.

The Kubernetes project maintains release branches for the most recent three minor releases.

Applicable fixes, including security fixes, may be backported to those three release branches, depending on severity and feasibility. Patch releases are cut from those branches at a regular cadence, or as needed. This decision is owned by the patch release manager. The patch release manager is a member of the release team for each release.

Minor releases occur approximately every 3 months, so each minor release branch is maintained for approximately 9 months.

Supported version skew

kube-apiserver

In highly-available (HA) clusters, the newest and oldest `kube-apiserver` instances must be within one minor version.

Example:

- newest `kube-apiserver` is at **1.13**
- other `kube-apiserver` instances are supported at **1.13** and **1.12**

kubelet

`kubelet` must not be newer than `kube-apiserver`, and may be up to two minor versions older.

Example:

- `kube-apiserver` is at **1.13**
- `kubelet` is supported at **1.13**, **1.12**, and **1.11**

  Note: If version skew exists between `kube-apiserver` instances in an HA cluster, this narrows the allowed `kubelet` versions.

Example:

- `kube-apiserver` instances are at **1.13** and **1.12**
- `kubelet` is supported at **1.12**, and **1.11** (**1.13** is not supported because that would be newer than the `kube-apiserver` instance at version **1.12**)

kube-controller-manager, kube-scheduler, and cloud-controller-manager

`kube-controller-manager`, `kube-scheduler`, and `cloud-controller-manager` must not be newer than the `kube-apiserver` instances they communicate with. They are expected to match the `kube-apiserver` minor version, but may be up to one minor version older (to allow live upgrades).

Example:

- `kube-apiserver` is at **1.13**
- `kube-controller-manager`, `kube-scheduler`, and `cloud-controller-manager` are supported at **1.13** and **1.12**

  Note: If version skew exists between `kube-apiserver` instances in an HA cluster, and these components can communicate with any `kube-apiserver` instance in the cluster (for example, via a load balancer), this narrows the allowed versions of these components.

Example:

- `kube-apiserver` instances are at **1.13** and **1.12**
- `kube-controller-manager`, `kube-scheduler`, and `cloud-controller-manager` communicate with a load balancer that can route to any `kube-apiserver` instance
- `kube-controller-manager`, `kube-scheduler`, and `cloud-controller-manager` are supported at **1.12** (**1.13** is not supported because that would be newer than the `kube-apiserver` instance at version **1.12**)

kubectl

`kubectl` is supported within one minor version (older or newer) of `kube-apiserver`.

Example:

- `kube-apiserver` is at **1.13**
- `kubectl` is supported at **1.14**, **1.13**, and **1.12**

  Note: If version skew exists between `kube-apiserver` instances in an HA cluster, this narrows the supported `kubectl` versions.

Example:

- `kube-apiserver` instances are at **1.13** and **1.12**
- `kubectl` is supported at **1.13** and **1.12** (other versions would be more than one minor version skewed from one of the `kube-apiserver` components)

Supported component upgrade order

The supported version skew between components has implications on the order in which components must be upgraded. This section describes the order in which components must be upgraded to transition an existing cluster from version **1.n** to version **1.(n+1)**.

kube-apiserver

Pre-requisites:

- In a single-instance cluster, the existing `kube-apiserver` instance is **1.n**
- In an HA cluster, all `kube-apiserver` instances are at **1.n** or **1.(n+1)** (this ensures maximum skew of 1 minor version between the oldest and newest `kube-apiserver` instance)
- The `kube-controller-manager`, `kube-scheduler`, and `cloud-controller-manager` instances that communicate with this server are at version **1.n** (this ensures they are not newer than the existing API server version, and are within 1 minor version of the new API server version)
- `kubelet` instances on all nodes are at version **1.n** or **1.(n-1)** (this ensures they are not newer than the existing API server version, and are within 2 minor versions of the new API server version)
- Registered admission webhooks are able to handle the data the new `kube-apiserver` instance will send them:
  - `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` objects are updated to include any new versions of REST resources added in **1.(n+1)** (or use the `matchPolicy: Equivalent` option available in v1.15+)
  - The webhooks are able to handle any new versions of REST resources that will be sent to them, and any new fields added to existing versions in **1.(n+1)**

Upgrade `kube-apiserver` to **1.(n+1)**

> **Note:** Project policies for API deprecation and API change guidelines require `kube-apiserver` to not skip minor versions when upgrading, even in single-instance clusters.

kube-controller-manager, kube-scheduler, and cloud-controller-manager

Pre-requisites:

- The `kube-apiserver` instances these components communicate with are at **1.(n+1)** (in HA clusters in which these control plane components can communicate with any `kube-apiserver` instance in the cluster, all `kube-apiserver` instances must be upgraded before upgrading these components)

Upgrade `kube-controller-manager`, `kube-scheduler`, and `cloud-controller-manager` to **1.(n+1)**

kubelet

Pre-requisites:

- The `kube-apiserver` instances the `kubelet` communicates with are at **1.(n+1)**

Optionally upgrade `kubelet` instances to **1.(n+1)** (or they can be left at **1.n** or **1.(n-1)**)

> **Warning:** Running a cluster with `kubelet` instances that are persistently two minor versions behind `kube-apiserver` is not recommended:
>
> - they must be upgraded within one minor version of `kube-apiserver` before the control plane can be upgraded
> - it increases the likelihood of running `kubelet` versions older than the three maintained minor releases

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on August 27, 2019 at 8:36 AM PST by Fix some error links (#15992) (Page History)

Edit This Page

v1.16 Release Notes

v1.16.0

Documentation

Downloads for v1.16.0

filename	sha512 hash
kubernetes.tar.gz	`99aa74225dd999d112ebc3e7b7d586a2312ec9c99de7a7fef8bbbfb198a5b4`
kubernetes-src.tar.gz	`0be7d1d6564385cc20ff4d26bab55b71cc8657cf795429d04caa5db133a672`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	`a5fb80d26c2a75741ad0efccdacd5d5869fbc303ae4bb1920a6883ebd93a`
kubernetes-client-darwin-amd64.tar.gz	`47a9a78fada4b840d9ae4dac2b469a36d0812ac83d22fd798c4cb0f1673f`
kubernetes-client-linux-386.tar.gz	`916e4dd98f5ed8ee111eeb6c2cf5c5f313e1d98f3531b40a5a777240ddb9`
kubernetes-client-linux-amd64.tar.gz	`fccf152588edbaaa21ca94c67408b8754f8bc55e49470380e10cf987be274`
kubernetes-client-linux-arm.tar.gz	`066c55fabbe3434604c46574c51c324336a02a5bfaed2e4d83b67012d26b`

filename	sha512 hash
kubernetes-client-linux-arm64.tar.gz	e41be74cc36240a64ecc962a066988b5ef7c3f3112977efd4e307b35dd78
kubernetes-client-linux-ppc64le.tar.gz	08783eb3bb2e35b48dab3481e17d6e345d43bab8b8dee25bb5ff184ba46c
kubernetes-client-linux-s390x.tar.gz	bcb6eb9cd3d8c92dfaf4f102ff2dc7517f632b1e955be6a02e7f223b15fc
kubernetes-client-windows-386.tar.gz	efbc764d8e2889ce13c9eaaa61f685a8714563ddc20464523140d6f5bef0
kubernetes-client-windows-amd64.tar.gz	b34bce694c6a0e4c8c5ddabcecb6adcb4d35f8c126b4b5ced7e44ef39cd4

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	a6bdac1eba1b87dc98b2bf5bf3690758960ecb50ed067736459b757fca0c3b01
kubernetes-server-linux-arm.tar.gz	0560e1e893fe175d74465065d43081ee7f40ba7e7d7cafa53e5d7491f89c6195
kubernetes-server-linux-arm64.tar.gz	4d5dd001fa3ac2b28bfee64e85dbedab0706302ffd634c34330617674e7a90e0
kubernetes-server-linux-ppc64le.tar.gz	cc642fca57e22bf6edd371e61e254b369b760c67fa00cac50e34464470f7eea6
kubernetes-server-linux-s390x.tar.gz	1f480ba6f593a3aa20203e82e9e34ac206e35839fd9135f495c5d154480c57d1

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	e987f141bc0a248e99a371ce220403b78678c739a39dad1c1612e63a0bee4525
kubernetes-node-linux-arm.tar.gz	8b084c1063beda2dd4000e8004634d82e580f05cc300c2ee13ad84bb884987b2
kubernetes-node-linux-arm64.tar.gz	365bdf9759e24d22cf507a0a5a507895ed44723496985e6d8f0bd10b03ffe7c7
kubernetes-node-linux-ppc64le.tar.gz	ff54d83dd0fd3c447cdd76cdffd253598f6800045d2b6b91b513849d15b0b602
kubernetes-node-linux-s390x.tar.gz	527cd9bf4bf392c3f097f232264c0f0e096ac410b5211b0f308c9d964f86900f

filename	sha512 hash
kubernetes-node-windows-amd64.tar.gz	`4f76a94c70481dd1d57941f156f395df008835b5d1cc17708945e8f560234dbd`

Kubernetes v1.16.0 Release Notes

A complete changelog for the release notes is now hosted in a customizable format at relnotes.k8s.io. Check it out and please give us your feedback!

What's New (Major Themes)

We're pleased to announce the delivery of Kubernetes 1.16, our third release of 2019! Kubernetes 1.16 consists of 31 enhancements: 8 enhancements moving to stable, 8 enhancements in beta, and 15 enhancements in alpha.

The main themes of this release are:

- **Custom resources:** CRDs are in widespread use as a way to extend Kubernetes to persist and serve new resource types, and have been available in beta since the 1.7 release. The 1.16 release marks the graduation of CRDs to general availability (GA).
- **Admission webhooks:** Admission webhooks are in widespread use as a Kubernetes extensibility mechanism and have been available in beta since the 1.9 release. The 1.16 release marks the graduation of admission webhooks to general availability (GA).
- **Overhauled metrics**: Kubernetes has previously made extensive use of a global metrics registry to register metrics to be exposed. By implementing a metrics registry, metrics are registered in more transparent means. Previously, Kubernetes metrics have been excluded from any kind of stability requirements.
- **Volume Extension**: There are quite a few enhancements in this release that pertain to volumes and volume modifications. Volume resizing support in CSI specs is moving to beta which allows for any CSI spec volume plugin to be resizable.

Additional Notable Feature Updates

- Topology Manager, a new Kubelet component, aims to co-ordinate resource assignment decisions to provide optimized resource allocations.
- IPv4/IPv6 dual-stack enables the allocation of both IPv4 and IPv6 addresses to Pods and Services.
- API Server Network Proxy going alpha in 1.16.
- Extensions for Cloud Controller Manager Migration.
- Continued deprecation of extensions/v1beta1, apps/v1beta1, and apps/v1beta2 APIs; these extensions will be retired in 1.16!

Known Issues

- The etcd and KMS plugin health checks are not exposed in the new `livez` and `readyz` endpoints. This will be fixed in 1.16.1.
- Systems running `iptables` 1.8.0 or newer should start it in legacy mode. Please note that this affects all versions of Kubernetes and not only v1.16.0. For more detailed information about the issue and how to apply a workaround, please refer to the official documentation

Urgent Upgrade Notes

(No, really, you MUST read this before you upgrade)

Cluster Lifecycle

- Container images tar files for `amd64` will now contain the architecture in the RepoTags manifest.json section. If you are using docker manifests there are not visible changes. (#80266, @javier-b-perez)
- kubeadm now deletes the bootstrap-kubelet.conf file after TLS bootstrap User relying on bootstrap-kubelet.conf should switch to kubelet.conf that contains node credentials (#80676, @fabriziopandini)
- Node labels `beta.kubernetes.io/metadata-proxy-ready`, `beta.kubernetes.io/metadata-proxy-ready` and `beta.kubernetes.io/kube-proxy-ds-ready` are no longer added on new nodes.
    - ip-mask-agent addon starts to use the label `node.kubernetes.io/masq-agent-ds-ready` instead of `beta.kubernetes.io/masq-agent-ds-ready` as its node selector.
    - kube-proxy addon starts to use the label `node.kubernetes.io/kube-proxy-ds-ready` instead of `beta.kubernetes.io/kube-proxy-ds-ready` as its node selector.
    - metadata-proxy addon starts to use the label `cloud.google.com/metadata-proxy-ready` instead of `beta.kubernetes.io/metadata-proxy-ready` as its node selector.

Storage

- When PodInfoOnMount is enabled for a CSI driver, the new csi.storage.k8s.io/ephemeral parameter in the volume context allows a driver's NodePublishVolume implementation to determine on a case-by-case basis whether the volume is ephemeral or a normal persistent volume (#79983, @pohly)
- Add CSI Migration Shim for VerifyVolumesAreAttached and BulkVolumeVerify (#80443, @davidz627)
- Promotes VolumePVCDataSource (Cloning) feature to beta for 1.16 release (#81792, @j-griffith)
- Integrated volume limits for in-tree and CSI volumes into one scheduler predicate. (#77595, @bertinatto)

Deprecations and Removals

- API
    - The following APIs are no longer served by default:
    - All resources under `apps/v1beta1` and `apps/v1beta2` - use `apps/v1` instead
    - `daemonsets`, `deployments`, `replicasets` resources under `extensions/v1beta1` - use `apps/v1` instead
    - `networkpolicies` resources under `extensions/v1beta1` - use `networking.k8s.io/v1` instead

- `podsecuritypolicies` resources under `extensions/v1beta1` - use `policy/v1beta1` instead

Serving these resources can be temporarily re-enabled using the `--runtime-config` apiserver flag.

- `apps/v1beta1=true`
- `apps/v1beta2=true`
- `extensions/v1beta1/daemonsets=true,extensions/v1beta1/deployments=true,extensions/v1beta1/replicasets=true,extensions/v1beta1/networkpolicies=true,extensions/v1beta1/podsecuritypolicies=true`

The ability to serve these resources will be completely removed in v1.18. (#70672, @liggitt)

- Ingress resources will no longer be served from `extensions/v1beta1` in v1.20. Migrate use to the `networking.k8s.io/v1beta1` API, available since v1.14. Existing persisted data can be retrieved via the `networking.k8s.io/v1beta1` API.
- PriorityClass resources will no longer be served from `scheduling.k8s.io/v1beta1` and `scheduling.k8s.io/v1alpha1` in v1.17. Migrate to the `scheduling.k8s.io/v1` API, available since v1.14. Existing persisted data can be retrieved via the `scheduling.k8s.io/v1` API.
- The `export` query parameter for list API calls, deprecated since v1.14, will be removed in v1.18.
- The `series.state` field in the events.k8s.io/v1beta1 Event API is deprecated and will be removed in v1.18 (#75987, @yastij)
- The `apiextensions.k8s.io/v1beta1` version of `CustomResourceDefinition` is deprecated and will no longer be served in v1.19. Use `apiextensions.k8s.io/v1` instead. (#79604, @liggitt)
- The `admissionregistration.k8s.io/v1beta1` versions of `MutatingWebhookConfiguration` and `ValidatingWebhookConfiguration` are deprecated and will no longer be served in v1.19. Use `admissionregistration.k8s.io/v1` instead. (#79549, @liggitt)
- The alpha `metadata.initializers` field, deprecated in 1.13, has been removed. (#79504, @yue9944882)
- The deprecated node condition type `OutOfDisk` has been removed. Use the `DiskPressure` condition instead. (#72420, @Pingan2017)
- The `metadata.selfLink` field is deprecated in individual and list objects. It will no longer be returned starting in v1.20, and the field will be removed entirely in v1.21. (#80978, @wojtek-t)
- The deprecated cloud providers `ovirt`, `cloudstack` and `photon` have been removed (#72178, @dims)
- The `Cinder` and `ScaleIO` volume providers have been deprecated and will be removed in a future release. (#80099, @dims)
- The GA `PodPriority` feature gate is now on by default and cannot be disabled. The feature gate will be removed in v1.18. (#79262, @draveness)
- Aggregated discovery requests can now timeout. Aggregated API servers must complete discovery calls within 5 seconds (other requests can take longer). Use the feature gate `EnableAggregatedDiscoveryTimeout=false` to temporarily revert behavior to the previous 30 second timeout if required (the temporary `Enabl`

eAggregatedDiscoveryTimeout feature gate will be removed in v1.17). (#82146, @deads2k)
    - the scheduler.alpha.kubernetes.io/critical-pod annotation is removed. Pod priority (spec.priorityClassName) should be used instead to mark pods as critical. (#80342, @draveness)
    - the NormalizeScore plugin set is removed from scheduler framework config API. Use ScorePlugin only. (#80930, @liu-cong)

- Features:
    - The following features are now GA, and the associated feature gates are deprecated and will be removed in v1.17:
    - GCERegionalPersistentDisk (since 1.15.0)
    - CustomResourcePublishOpenAPI
    - CustomResourceSubresources
    - CustomResourceValidation
    - CustomResourceWebhookConversion
    - The feature flags HugePages, VolumeScheduling, CustomPodDNS and Pod ReadinessGates have been removed (#79307, @draveness)

- hyperkube
    - the --make-symlinks flag, deprecated in v1.14, has been removed. (#80017, @Pothulapati)

- kube-apiserver
    - the --basic-auth-file flag and authentication mode is deprecated and will be removed in a future release. It is not recommended for production environments. (#81152, @tedyu)
    - the --cloud-provider-gce-lb-src-cidrs flag has been deprecated. This flag will be removed once the GCE Cloud Provider is removed from kube-apiserver. (#81094, @andrewsykim)
    - the --enable-logs-handler flag and log-serving functionality is deprecated since v1.15, and scheduled to be removed in v1.19. (#77611, @rohitsardesai83)
    - Deprecate the default service IP CIDR. The previous default was 10.0.0.0/24 which will be removed in 6 months/2 releases. Cluster admins must specify their own desired value, by using --service-cluster-ip-range on kube-apiserver. (#81668, @darshanime)

- kube-proxy
    - the --resource-container flag has been removed from kube-proxy, and specifying it will now cause an error. The behavior is now as if you specified --resource-container="". If you previously specified a non-empty --resource-container, you can no longer do so as of kubernetes 1.16. (#78294, @vllry)

- kube-scheduler
    - Migrate scheduler to use v1beta1 Event API. any tool targeting scheduler events needs to use v1beta1 Event API (#78447, @yastij)

- kubeadm
  - The CoreDNS Deployment now checks readiness via the `ready` plugin.
  - The `proxy` plugin has been deprecated. The `forward` plugin is to be used instead.
  - `kubernetes` plugin removes the `resyncperiod` option.
  - The `upstream` option is deprecated and ignored if included. (#82127, @rajansandeep)
- kubectl
  - `kubectl convert`, deprecated since v1.14, will be removed in v1.17.
  - The `--export` flag for the `kubectl get` command, deprecated since v1.14, will be removed in v1.18.
  - `kubectl cp` no longer supports copying symbolic links from containers; to support this use case, see `kubectl exec --help` for examples using `tar` directly (#82143, @soltysh)
  - Removed deprecated flag `--include-uninitialized`. (#80337, @draveness)
- kubelet
  - the `--containerized` flag was deprecated in 1.14 and has been removed (#80043, @dims)
  - the `beta.kubernetes.io/os` and `beta.kubernetes.io/arch` labels, deprecated since v1.14, are targeted for removal in v1.18.
  - cAdvisor json endpoints have been deprecated since 1.15. (#78504, @dashpole)
  - removed the ability to set `kubernetes.io`- or `k8s.io`-prefixed labels via `--node-labels`, other than the specifically allowed labels/prefixes. (#79305, @paivagustavo)
- client-go
  - Remove `DirectCodecFactory` (replaced with `serializer.WithoutConversionCodecFactory`), `DirectEncoder` (replaced with `runtime.WithVersionEncoder`) and `DirectDecoder` (replaced with `runtime.WithoutVersionDecoder`). (#79263, @draveness)

Metrics Changes

Added metrics

- Added metrics `aggregator_openapi_v2_regeneration_count`, `aggregator_openapi_v2_regeneration_gauge` and `apiextension_openapi_v2_regeneration_count` counting the triggering APIService and CRDs and the reason (add, update, delete) when kube-apiserver regenerates the OpenAPI spec. (#81786, @sttts)
- Added metrics `authentication_attempts` that can be used to understand the attempts of authentication. (#81509, @RainbowMango)
- Add a new counter metrics `apiserver_admission_webhook_rejection_count` with details about the causing for a webhook rejection. (#81399, @roycaihw)
- NFS Drivers are now enabled to collect metrics, StatFS metrics provider is used to collect the metrics. (@brahmaroutu) (#75805, @brahmaroutu)
- Add `container_sockets`, `container_threads`, and `container_threads_max` metrics (#81972, @dashpole)

- Add `container_state` label to `running_container_count` kubelet metrics, to get count of containers based on their state(running/exited/created/unknown) (#81573, @irajdeep)
- Added metric `apiserver_watch_events_total` that can be used to understand the number of watch events in the system. (#78732, @mborsz)
- Added metric `apiserver_watch_events_sizes` that can be used to estimate sizes of watch events in the system. (#80477, @mborsz)
- Added a new Prometheus counter metric `sync_proxy_rules_iptables_restore_failures_total` for kube-proxy iptables-restore failures (both ipvs and iptables modes) (#81210, @figo)
- kubelet now exports an `kubelet_evictions` metric that counts the number of pod evictions carried out by the kubelet to reclaim resources (#81377, @sjenning)

Removed metrics

- Removed cadvisor metric labels `pod_name` and `container_name` to match instrumentation guidelines. Any Prometheus queries that match `pod_name` and `container_name` labels (e.g. cadvisor or kubelet probe metrics) must be updated to use `pod` and `container` instead. (#80376, @ehashman)

Deprecated/changed metrics

- kube-controller-manager and cloud-controller-manager metrics are now marked as with the ALPHA stability level. (#81624, @logicalhan)
- kube-proxy metrics are now marked as with the ALPHA stability level. (#81626, @logicalhan)
- kube-apiserver metrics are now marked as with the ALPHA stability level. (#81531, @logicalhan)
- kubelet metrics for /metrics and /metrics/probes are now marked as with the ALPHA stability level. (#81534, @logicalhan)
- Scheduler metrics are now marked as with the ALPHA stability level. (#81576, @logicalhan)
- The `rejected` label in `apiserver_admission_webhook_admission_duration_seconds` metrices now properly indicates if the request was rejected. (#81399, @roycaihw)
- Fixed a bug in the CSI metrics that does not return not supported error when a CSI driver does not support metrics. (#79851, @jparklab)
- Fix disk stats in LXD using ZFS storage pool and CRI-O missing network metris bug (#81972, @dashpole)

Notable Features

Beta

- Promote WatchBookmark feature to beta and enable it by default. With WatchBookmark feature, clients are able to request watch events with BOOKMARK type. Clients should not assume bookmarks are returned at any specific interval, nor may they assume the server will send any BOOKMARK event during a session. (#79786, @wojtek-t)
- The server-side apply feature is now beta (#81956, @apelisse)
- Server-side apply will now use the openapi provided in the CRD validation field to help figure out how to correctly merge objects and update ownership. (#77354, @jennybuckley)
- The `CustomResourceDefaulting` feature is promoted to beta and enabled by default. Defaults may be specified in structural schemas via the `apiextensions.k8s.`

`io/v1` API. See https://kubernetes.io/docs/tasks/access-kubernetes-api/custom-resources/custom-resource-definitions/#specifying-a-structural-schema for details. (#81872, @sttts)
- Finalizer Protection for Service LoadBalancers is now in beta (enabled by default). This feature ensures the Service resource is not fully deleted until the correlating load balancer resources are deleted. (#81691, @MrHohn)
- Graduating Windows GMSA support from alpha to beta (#82110, @wk8)

Alpha

- Introduce a new admission controller for RuntimeClass. Initially, RuntimeClass will be used to apply the pod overhead associated with a given RuntimeClass to the Pod `spec` if a corresponding RuntimeClassName is specified. PodOverhead is an alpha feature as of Kubernetes 1.16. (#78484, @egernst)
- Introduction of the pod overhead feature to the scheduler. This functionality is alpha-level as of Kubernetes v1.16, and is only honored by servers that enable the PodOverhead feature.gate. (#78319, @egernst)
- Ephemeral containers have been added in alpha. These temporary containers can be added to running pods for purposes such as debugging, similar to how `kubectl exec` runs a process in an existing container. Also like `kubectl exec`, no resources are reserved for ephemeral containers and they are not restarted when they exit. Note that container namespace targeting is not yet implemented, so process namespace sharing must be enabled to view process from other containers in the pod. (#59484, @verb)

CLI Improvements

- the new flag `--endpoint-updates-batch-period` in kube-controller-manager can be used to reduce the number of endpoints updates generated by pod changes. (#80509, @mborsz)
- the kubectl `--all-namespaces` flag is now honored by `kubectl wait` (#81468, @ashutoshgngwr)
- `kubectl get -w` now takes an `--output-watch-events` flag to indicate the event type (ADDED, MODIFIED, DELETED) (#72416, @liggitt)
- Adds Endpoint Slice support for kubectl when discovery API group is enabled. (#81795, @robscott)

Misc

- Add `--shutdown-delay-duration` to kube-apiserver in order to delay a graceful shutdown. `/healthz` will keep returning success during this time and requests are normally served, but `/readyz` will return failure immediately. This delay can be used to allow the SDN to update iptables on all nodes and stop sending traffic. (#74416, @sttts) Kubeadm now seamlessly migrates the CoreDNS Configuration when upgrading CoreDNS. (#78033, @rajansandeep)
- Add Endpoint Slice Controller for managing new EndpointSlice resource, disabled by default. (#81048, @robscott)
- Adds `\livez` for liveness health checking for kube-apiserver. Using the parameter `--maximum-startup-sequence-duration` will allow the liveness endpoint to defer boot-sequence failures for the specified duration period. (#81969, @logicalhan)
- Adds EndpointSlice integration to kube-proxy, can be enabled with EndpointSlice feature gate. (#81430, @robscott)
- Add status condition to namespace resource (#73405, @wozniakjan)

- Enhance Azure cloud provider code to support both AAD and ADFS authentication. (#80841, @rjaini)
- kubeadm: implement support for concurrent add/remove of stacked etcd members (#79677, @neolit123)
- kubeadm: support any Linux kernel version newer than 3.10 (#81623, @neolit123)
- Volume expansion is enabled in the default GCE storageclass (#78672, @msau42)
- kubeadm ClusterConfiguration now supports featureGates: IPv6DualStack: true (#80145, @Arvinderpal)
- In order to enable dual-stack support within kubeadm and kubernetes components, as part of the init config file, the user should set feature-gate `IPv6DualStack=true` in the ClusterConfiguration. Additionally, for each worker node, the user should set the feature-gate for kubelet using either `nodeRegistration.kubeletExtraArgs` or `KUBELET_EXTRA_ARGS`. (#80531, @Arvinderpal)
- Add possibility to configure controller manager to use IPv6 dual stack: use `--cluster-cidr="<cidr1>,<cidr2>"`. Notes:
  1. Only the first two CIDRs are used (soft limits for Alpha, might be lifted later on).
  2. Only the "RangeAllocator" (default) is allowed as a value for `--cidr-allocator-type`. Cloud allocators are not compatible with IPv6 dual stack (#73977, @khenidak)
- Add scheduling support for RuntimeClasses. RuntimeClasses can now specify nodeSelector constraints & tolerations, which are merged into the PodSpec for pods using that RuntimeClass. (#80825, @tallclair)
- When specifying `--(kube|system)-reserved-cgroup`, with `--cgroup-driver=systemd`, it is now possible to use the fully qualified cgroupfs name (i.e. `/test-cgroup.slice`). (#78793, @mattjmcnaughton)
- Adds support for vSphere volumes on Windows (#80911, @gab-satchi)

API Changes

- The `MutatingWebhookConfiguration` and `ValidatingWebhookConfiguration` APIs have been promoted to `admissionregistration.k8s.io/v1`:
  - `failurePolicy` default changed from `Ignore` to `Fail` for v1
  - `matchPolicy` default changed from `Exact` to `Equivalent` for v1
  - `timeout` default changed from `30s` to `10s` for v1
  - `sideEffects` default value is removed, and the field made required, and only `None` and `NoneOnDryRun` are permitted for v1
  - `admissionReviewVersions` default value is removed and the field made required for v1 (supported versions for AdmissionReview are `v1` and `v1beta1`)
  - The `name` field for specified webhooks must be unique for `MutatingWebhookConfiguration` and `ValidatingWebhookConfiguration` objects created via `admissionregistration.k8s.io/v1`
- The `AdmissionReview` API sent to and received from admission webhooks has been promoted to `admission.k8s.io/v1`. Webhooks can specify a preference for receiving `v1` AdmissionReview objects with `admissionReviewVersions: ["v1","v1beta1"]`, and must respond with an API object in the same `apiVersion` they are sent. When webhooks use `admission.k8s.io/v1`, the following additional validation is performed on their responses:
  - `response.patch` and `response.patchType` are not permitted from validating admission webhooks
  - `apiVersion: "admission.k8s.io/v1"` is required
  - `kind: "AdmissionReview"` is required
  - `response.uid: "<value of request.uid>"` is required

    - `response.patchType: "JSONPatch"` is required (if `response.patch` is set) (#80231, @liggitt)
- The `CustomResourceDefinition` API type is promoted to `apiextensions.k8s.io/v1` with the following changes:
    - Use of the new `default` feature in validation schemas is limited to v1
    - `spec.scope` is no longer defaulted to `Namespaced` and must be explicitly specified
    - `spec.version` is removed in v1; use `spec.versions` instead
    - `spec.validation` is removed in v1; use `spec.versions[*].schema` instead
    - `spec.subresources` is removed in v1; use `spec.versions[*].subresources` instead
    - `spec.additionalPrinterColumns` is removed in v1; use `spec.versions[*].additionalPrinterColumns` instead
    - `spec.conversion.webhookClientConfig` is moved to `spec.conversion.webhook.clientConfig` in v1
    - `spec.conversion.conversionReviewVersions` is moved to `spec.conversion.webhook.conversionReviewVersions` in v1
    - `spec.versions[*].schema.openAPIV3Schema` is now required when creating v1 CustomResourceDefinitions
    - `spec.preserveUnknownFields: true` is disallowed when creating v1 CustomResourceDefinitions; it must be specified within schema definitions as `x-kubernetes-preserve-unknown-fields: true`
    - In `additionalPrinterColumns` items, the `JSONPath` field was renamed to `jsonPath` in v1 (fixes https://github.com/kubernetes/kubernetes/issues/66531) The `apiextensions.k8s.io/v1beta1` version of `CustomResourceDefinition` is deprecated and will no longer be served in v1.19. (#79604, @liggitt)
- The `ConversionReview` API sent to and received from custom resource CustomResourceDefinition conversion webhooks has been promoted to `apiextensions.k8s.io/v1`. CustomResourceDefinition conversion webhooks can now indicate they support receiving and responding with `ConversionReview` API objects in the `apiextensions.k8s.io/v1` version by including `v1` in the `conversionReviewVersions` list in their CustomResourceDefinition. Conversion webhooks must respond with a ConversionReview object in the same apiVersion they receive. `apiextensions.k8s.io/v1` `ConversionReview` responses must specify a `response.uid` that matches the `request.uid` of the object they were sent. (#81476, @liggitt)
- Add scheduling support for RuntimeClasses. RuntimeClasses can now specify nodeSelector constraints & tolerations, which are merged into the PodSpec for pods using that RuntimeClass. (#80825, @tallclair)
- Kubelet should now more reliably report the same primary node IP even if the set of node IPs reported by the CloudProvider changes. (#79391, @danwinship)
- Omit nil or empty field when calculating container hash value to avoid hash changed. For a new field with a non-nil default value in the container spec, the hash would still get changed. (#57741, @dixudx)
- Property `conditions` in `apiextensions.v1beta1.CustomResourceDefinitionStatus` and `apiextensions.v1.CustomResourceDefinitionStatus` is now optional instead of required. (#64996, @roycaihw)
- When the status of a CustomResourceDefinition condition changes, its corresponding `lastTransitionTime` is now updated. (#69655, @CaoShuFeng)

Other notable changes

API Machinery

- Remove `GetReference()` and `GetPartialReference()` function from `pkg/api/ref`, as the same function exists also in `staging/src/k8s.io/client-go/tools/ref` (#80361, @wojtek-t)
- Verify that CRD default values in OpenAPI specs are pruned, with the exceptions of values under `metadata`. (#78829, @sttts)
- Fixes a bug that when there is a "connection refused" error, the reflector's ListAndWatch func will return directly but what expected is that sleep 1 second and rewatch since the specified resourceVersion. (#81634, @likakuli)
- Resolves an issue serving aggregated APIs backed by services that respond to requests to `/` with non-2xx HTTP responses (#79895, @deads2k)
- The CRD handler now properly re-creates stale CR storage to reflect CRD update. (#79114, @roycaihw)
- Fix CVE-2019-11247: API server allows access to custom resources via wrong scope (#80750, @sttts)
- Fixed a bug with the openAPI definition for `io.k8s.apimachinery.pkg.runtime.RawExtension`, which previously required a field `raw` to be specified (#80773, @jennybuckley)
- Property `conditions` in `apiextensions.v1beta1.CustomResourceDefinitionStatus` and `apiextensions.v1.CustomResourceDefinitionStatus` is now optional instead of required. (#64996, @roycaihw)
- Resolves a transient 404 response to custom resource requests during server startup (#81244, @liggitt)
- OpenAPI now advertises correctly supported patch types for custom resources (#81515, @liggitt)
- When the status of a CRD Condition changes, it's corresponding `LastTransitionTime` is now updated. (#69655, @CaoShuFeng)
- Add `metadata.generation=1` to old CustomResources. (#82005, @sttts)
- Fix a bug in the apiserver that could cause a valid update request to be rejected with a precondition check failure. (#82303, @roycaihw)
- Fixes regression in logging spurious stack traces when proxied connections are closed by the backend (#82588, @liggitt)
- RateLimiter add a context-aware method, fix client-go request goruntine backlog in async timeout scene. (#79375, @answer1991)
- Add a `Patch` method to `ScaleInterface` (#80699, @knight42)
- CRDs under k8s.io and kubernetes.io must have the `api-approved.kubernetes.io` set to either `unapproved.*` or a link to the pull request approving the schema. See https://github.com/kubernetes/enhancements/pull/1111 for more details. (#79992, @deads2k)
- KMS Providers will install a healthz check for the status of kms-plugin in kube-apiservers' encryption config. (#78540, @immutableT)
- Improves validation errors for custom resources (#81212, @liggitt)
- Populate object name for admission attributes when CREATE (#53185, @dixudx)
- Add Overhead field to the PodSpec and RuntimeClass types as part of the Pod Overhead KEP (#76968, @egernst)

Apps

- Fix a bug that pods not be deleted from unmatched nodes by daemon controller (#78974, @DaiHao)
- Fix a bug that causes DaemonSet rolling update hang when there exist failed pods. (#78170, @DaiHao)

Auth

- Service account tokens now include the JWT Key ID field in their header. (#78502, @ahmedtd)
- The nbf (not before) claim, if present in ID token, is now enforced. (#81413, @anderseknert)

CLI

- Fix CVE-2019-11249: Incomplete fixes for CVE-2019-1002101 and CVE-2019-11246, kubectl cp potential directory traversal (#80436, @M00nF1sh)
- Fix the bash completion error with override flags. (#80802, @dtaniwaki)
- Fix a bug in server printer that could cause kube-apiserver to panic. (#79349, @roycaihw)
- Fix invalid "time stamp is the future" error when kubectl cp-ing a file (#73982, @tanshanshan)
- Fix a bug where `kubectl set config` hangs and uses 100% CPU on some invalid property names (#79000, @pswica)
- Fix output of `kubectl get --watch-only` when watching a single resource (#79345, @liggitt)
- Make kubectl get `--ignore-not-found` continue processing when encountering error. (#82120, @soltysh)
- Correct a reference to a not/no longer used kustomize subcommand in the documentation (#82535, @demobox)
- kubectl could scale custom resource again (#81342, @knight42)
- Add PodOverhead awareness to kubectl (#81929, @egernst)

Cloud Provider

- When a load balancer type service is created in a k8s cluster that is backed by Azure Standard Load Balancer, the corresponding load balancer rule added in the Azure Standard Load Balancer would now have the "EnableTcpReset" property set to true. (#80624, @xuto2)
- Switch to VM Update call in attach/detach disk operation, original CreateOrUpdate call may lead to orphaned VMs or blocked resources (#81208, @andyzhangx)
- Fix azure disk naming matching issue due to case sensitive comparison (#81720, @andyzhangx)
- Fix retry issues when the nodes are under deleting on Azure (#80419, @feiskyer)
- Fix conflicted cache when the requests are canceled by other Azure operations. (#81282, @feiskyer)
- Fix make azure disk URI as case insensitive (#79020, @andyzhangx)
- Fix VMSS LoadBalancer backend pools so that the network won't be broken when instances are upgraded to latest model (#81411, @nilo19)
- Default resourceGroup should be used when the value of annotation azure-load-balancer-resource-group is an empty string. (#79514, @feiskyer)

- Kubelet could be run with no Azure identity without subscriptionId configured now. A sample cloud provider configure is: â€˜{"vmType": "vmss", "useInstanceMetadata": true} â€˜. (#81500, @feiskyer)
- Fix public IP not found issues for VMSS nodes (#80703, @feiskyer)
- Fix Azure client requests stuck issues on http.StatusTooManyRequests (HTTP Code 429). (#81279, @feiskyer)
- Add a service annotation `service.beta.kubernetes.io/azure-pip-name` to specify the public IP name for Azure load balancer. (#81213, @nilo19)
- Optimize EC2 DescribeInstances API calls in aws cloud provider library by querying instance ID instead of EC2 filters when possible (#78140, @zhan849)
- Creates an annotation `service.beta.kubernetes.io/aws-load-balancer-eip-allocations` to assign AWS EIP to the newly created Network Load Balancer. Number of allocations and subnets must match. (#69263, @brooksgarrett)
- Add an azure cloud configuration `LoadBalancerName` and `LoadBalancerResourceGroup` to allow the corresponding customizations of azure load balancer. (#81054, @nilo19)

Cluster Lifecycle

- Fix error handling and potential go null pointer exception in kubeadm upgrade diff (#80648, @odinuge)
- kubeadm: fall back to client version in case of certain HTTP errors (#80024, @RainbowMango)
- kubeadm: fix a potential panic if kubeadm discovers an invalid, existing kubeconfig file (#79165, @neolit123)
- kubeadm: treat non-fatal errors as warnings when doing reset (#80862, @drpaneas)
- kubeadm: prevent PSP blocking of upgrade image prepull by using a non-root user (#77792, @neolit123)
- kubeadm: fix "certificate-authority" files not being pre-loaded when using file discovery (#80966, @neolit123)
- Add instruction to setup "Application Default Credentials" to run GCE Windows e2e tests locally. (#81337, @YangLu1031)
- Fix error in `kubeadm join --discovery-file` when using discovery files with embedded credentials (#80675, @fabriziopandini)
- Fix remove the etcd member from the cluster during a kubeadm reset. (#79326, @bradbeam)
- kubeadm: the permissions of generated CSR files are changed from 0644 to 0600 (#81217, @SataQiu)
- kubeadm: avoid double deletion of the upgrade prepull DaemonSet (#80798, @xlgao-zju)
- kubeadm: introduce deterministic ordering for the certificates generation in the phase command `kubeadm init phase certs`. (#78556, @neolit123)
- kubeadm: implement retry logic for certain ConfigMap failures when joining nodes (#78915, @ereslibre)
- kubeadm: use etcd's /health endpoint for a HTTP liveness probe on localhost instead of having a custom health check using etcdctl (#81385, @neolit123)
- kubeadm reset: unmount directories under `/var/lib/kubelet` for Linux only (#81494, @Klaven)
- kubeadm: fix the bug that `--cri-socket` flag does not work for `kubeadm reset` (#79498, @SataQiu)
- kubeadm: produce errors if they occur when resetting cluster status for a control-plane node (#80573, @bart0sh)
- Fix an error when using external etcd but storing etcd certificates in the same folder with the same name used by kubeadm for local etcd certificates; for an older version of

kubeadm, the workaround is to avoid file name used by kubeadm for local etcd. (#80867, @fabriziopandini)
- `kubeadm join` fails if file-based discovery is too long, with a default timeout of 5 minutes. (#80804, @olivierlemasle)
- kubeadm: fixed ignoring errors when pulling control plane images (#80529, @bart0sh)
- Fix a bug in kube-addon-manager's leader election logic that made all replicas active. (#80575, @mborsz)
- kubeadm: prevent overriding of certain kubelet security configuration parameters if the user wished to modify them (#81903, @jfbai)
- kubeadm no longer performs IPVS checks as part of its preflight checks (#81791, @yastij)
- kubeadm: fix for HTTPProxy check for IPv6 addresses (#82267, @kad)
- kubeadm: Allow users to skip the kube-proxy init addon phase during init and still be able to join a cluster and perform some other minor operations (but not upgrade). (#82248, @rosti)
- Mounts `/home/kubernetes/bin/nvidia/vulkan/icd.d` on the host to `/etc/vulkan/icd.d` inside containers requesting GPU. (#78868, @chardch)
- kubeadm: use the `--pod-network-cidr` flag to init or use the podSubnet field in the kubeadm config to pass a comma separated list of pod CIDRs. (#79033, @Arvinderpal)
- kubeadm: provide `--control-plane-endpoint` flag for `controlPlaneEndpoint` (#79270, @SataQiu)
- kubeadm: enable secure serving for the kube-scheduler (#80951, @neolit123)
- kubeadm: print the stack trace of an error for klog level `--v>=5` (#80937, @neolit123)
- Add `--kubernetes-version` to `kubeadm init phase certs ca` and `kubeadm init phase kubeconfig` (#80115, @gyuho)
- kubeadm: support fetching configuration from the original cluster for `upgrade diff` (#80025, @SataQiu)
- When using the conformance test image, a new environment variable `E2E_USE_GO_RUNNER` will cause the tests to be run with the new golang-based test runner rather than the current bash wrapper. (#79284, @johnSchnake)
- Implement a new feature that allows applying kustomize patches to static pod manifests generated by kubeadm. (#80905, @fabriziopandini)

- The 404 request handler for the GCE Ingress load balancer now exports prometheus metrics, including:

    - `http_404_request_total` (the number of 404 requests handled)
    - `http_404_request_duration_ms` (the amount of time the server took to respond in ms)

Also includes percentile groupings. The directory for the default 404 handler includes instructions on how to enable prometheus for monitoring and setting alerts. (#79106, @vbannai)

Instrumentation

- Kibana has been slightly revamped/improved in the latest version (#80421, @lostick)

Network

- Fix a string comparison bug in IPVS graceful termination where UDP real servers are not deleted. (#78999, @andrewsykim)
- `kube-proxy --cleanup will` return the correct exit code if the cleanup was successful (#78775, @johscheuer)
- Fix a bug in the IPVS proxier where virtual servers are not cleaned up even though the corresponding Service object was deleted. (#80942, @gongguan)

- kube-proxy waits for some duration for the node to be defined. (#77167, @paulsubrata55)
- Increase log level for graceful termination to `v=5` (#80100, @andrewsykim)
- Reduce kube-proxy CPU usage in IPVS mode when a large number of nodePort services exist. (#79444, @cezarsa)
- Fix in kube-proxy for SCTP nodeport service which only works for node's InternalIP, but doesn't work for other IPs present in the node when ipvs is enabled. (#81477, @paulsubrata55)
- Ensure the `KUBE-MARK-DROP` chain in kube-proxy IPVS mode. The chain is ensured for both IPv4 and IPv6 in dual-stack operation. (#82214, @uablrek)
- Introduce `node.kubernetes.io/exclude-balancer` and `node.kubernetes.io/exclude-disruption` labels in alpha to prevent cluster deployers from being dependent on the optional `node-role` labels which not all clusters may provide. (#80238, @smarterclayton)
- If targetPort is changed that will process by service controller (#77712, @Sn0rt)

Node

- Remove PIDs cgroup controller requirement when related feature gates are disabled (#79073, @rafatio)
- Fix kubelet NodeLease potential performance issues. Kubelet now will try to update lease using cached one instead of get from API Server every time. (#81174, @answer1991)
- Passing an invalid policy name in the `--cpu-manager-policy` flag will now cause the kubelet to fail instead of simply ignoring the flag and running the `cpumanager`'s default policy instead. (#80294, @klueska)
- Make node lease renew interval more heuristic based on node-status-update-frequency in kubelet (#80173, @gaorong)
- Kubelet should now more reliably report the same primary node IP even if the set of node IPs reported by the CloudProvider changes. (#79391, @danwinship)
- Omit `nil` or empty field when calculating container hash value to avoid hash changed. For a new field with a non-nil default value in the container spec, the hash would still get changed. (#57741, @dixudx)
- Fix a bug where kubelet would not retry pod sandbox creation when the restart policy of the pod is Never (#79451, @yujuhong)
- Limit the body length of exec readiness/liveness probes. remote CRIs and Docker shim read a max of 16MB output of which the exec probe itself inspects 10kb. (#82514, @dims)
- Single static pod files and pod files from http endpoints cannot be larger than 10 MB. HTTP probe payloads are now truncated to 10KB. (#82669, @rphillips)
- Introduce support for applying pod overhead to pod cgroups, if the PodOverhead feature is enabled. (#79247, @egernst)
- Node-Problem-Detector v0.7.1 is used on GCI (#80726, @wangzhen127)
- Node-Problem-Detector v0.7.1 is used for addon daemonset. (#82140, @wangzhen127)
- Enable cAdvisor ProcessMetrics collecting. (#79002, @jiayingz)
- kubelet: change `node-lease-renew-interval` to 0.25 of lease-renew-duration (#80429, @gaorong)
- Attempt to set the kubelet's hostname & internal IP if `--cloud-provider=external` and no node addresses exists (#75229, @andrewsykim)

Scheduling

- Scheduler should terminate when it loses leader lock. (#81306, @ravisantoshgudimetla)
- If scheduler extender filtered a not found node, current scheduling round for this pod will just be skipped. (#79641, @yqwang-ms)
- Extender bind should respect IsInterested (#79804, @yqwang-ms)

- Fix an issue with toleration merging & whitelist checking in the PodTolerationRestriction admission controller. (#81732, @tallclair)
- Add a helper function to decode scheduler plugin args. (#80696, @hex108)
- Fix filter plugins are not been called during preemption (#81876, @wgliang)
- Fix an issue that the correct PluginConfig.Args is not passed to the corresponding PluginFactory in kube-scheduler when multiple PluginConfig items are defined. (#82483, @everpeace)
- Take the context as the first argument of Schedule. (#82119, @wgliang)
- Implement `post-filter` extension point for scheduling framework (#78097, @draveness)
- Add Bind extension point of the scheduling framework (#78513, @chenchun)
- Add Filter extension point to the scheduling framework. (#78477, @YoubingLi)
- Return error when the scoring plugin returns score out of range `[0, 100]`. (#81015, @draveness)
- Use a named array instead of a score array in normalizing-score phase. (#80901, @draveness)
- Updates the `requestedToCapacityRatioArguments` to add resources parameter that allows the users to specify the resource name along with weights for each resource to score nodes based on the request to capacity ratio. (#77688, @sudeshsh)
- Add `UnschedulableAndUnresolvable` status code for scheduling framework (#82034, @alculquicondor)
- Add normalize plugin extension point for the scheduling framework. (#80383, @liu-cong)
- Add Bind extension point to the scheduling framework. (#79313, @chenchun)
- Add Score extension point to the scheduling framework. (#79109, @ahg-g)
- Add Pre-filter extension point to the scheduling framework. (#78005, @ahg-g)
- Add support for writing out of tree custom scheduler plugins. (#78162, @hex108)

Storage

- Fix possible file descriptor leak and closing of dirs in `doSafeMakeDir` (#79534, @odinuge)
- Azure disks of shared kind will no longer fail if they do not contain `skuname` or `storag eaccounttype`. (#80837, @rmweir)
- Fix CSI plugin supporting raw block that does not need attach mounted failed (#79920, @cwdsuzhou)
- Reduces GCE PD Node Attach Limits by 1 since the node boot disk is considered an attachable disk (#80923, @davidz627)
- Remove iSCSI volume storage cleartext secrets in logs (#81215, @zouyee)
- Fixes validation of VolumeAttachment API objects created with inline volume sources. (#80945, @tedyu)
- Changes timeout value in csi plugin from 15s to 2min which fixes the timeout issue (#79529, @andyzhangx)
- Fix kubelet fail to delete orphaned pod directory when the kubelet's pods directory (default is `/var/lib/kubelet/pods`) symbolically links to another disk device's directory (#79094, @gaorong)

Testing

- Fix pod list return value of `framework.WaitForPodsWithLabelRunningReady` (#78687, @pohly)
- Adding `TerminationGracePeriodSeconds` to the test framework API (#82170, @vivekbagade)

- `/test/e2e/framework`: Adds a flag `non-blocking-taints` which allows tests to run in environments with tainted nodes. String value should be a comma-separated list. (#81043, @johnSchnake)
- Move CSI volume expansion to beta. (#81467, @bertinatto)
- Added E2E tests validating WindowsOptions.RunAsUserName. (#79539, @bclau)
- `framework.ExpectNoError` no longer logs the error and instead relies on using the new `log.Fail` as gomega fail handler. (#80253, @pohly)

Windows

- On Windows systems, `%USERPROFILE%` is now preferred over `%HOMEDRIVE%\%HOMEPATH%` as the home folder if `%HOMEDRIVE%\%HOMEPATH%` does not contain a `.kube\config` file, and `%USERPROFILE%` exists and is writable. (#73923, @liggitt)
- Add support for AWS EBS on windows (#79552, @wongma7)
- Support Kubelet plugin watcher on Windows nodes. (#81397, @ddebroy)

Dependencies

Changed

- the default Go version was updated to v1.12.9. (#78958, #79966, #81390, #81489)
- etcd has been updated to v3.3.15 (#82199, @dims)
- CoreDNS for kubeadm and kube-up has been updated to v1.6.2 (#82127)
- Cluster Autoscaler has been updated to v1.16.0 (#82501, @losipiuk)
- fluentd has been updated to v1.5.1 (#79014)
- fluentd-elasticsearch plugin has been updated to v3.5.3 (#79014)
- elasticsearch has been updated to v7.1.1 (#79014)
- kibana has been updated to v7.1.1 (#79014)
- Azure SDK and go-autorest API versions have been updated (#79574)
- Azure API versions have been updated (container registry to 2018-09-01, network to 2018-08-01) (#79583)
- kube-addon-manager has been updated to v9.0.2 (#80861)
- golang/x/net has been updated to bring in fixes for CVE-2019-9512, CVE-2019-9514 (#81394)
- GCE windows node image has been updated. (#81106)
- portworx plugin has been updated on libopenstorage/openstorage to v1.0.0 (#80495)
- metrics-server has been updated to v0.3.4 (#82322, @olagacek)
- klog has been updated to v0.4.0 (#81164)

Unchanged

- The list of validated docker versions remains unchanged.
  - The current list is 1.13.1, 17.03, 17.06, 17.09, 18.06, 18.09. (#72823, #72831)
- CNI remains unchanged at v0.7.5. (#75455)
- cri-tools remains unchanged at v1.14.0. (#75658)
- CAdvisor remains unchanged at v0.33.2. (#76291)
- event-exporter remains unchanged at v0.2.5. (#77815)
- ip-masq-agent remains unchanged at v2.4.1. (#77844)
- k8s-dns-node-cache remains unchanged at v1.15.1 (#76640, @george-angel)
- CSI remains unchanged at to v1.1.0. (#75391)
- The dashboard add-on remains unchanged at v1.10.1. (#72495)
- kube-dns is unchanged at v1.14.13 as of Kubernetes 1.12. (#68900)
- Influxdb is unchanged at v1.3.3 as of Kubernetes 1.10. (#53319)

- Grafana is unchanged at v4.4.3 as of Kubernetes 1.10. (#53319)
- The fluent-plugin-kubernetes_metadata_filter plugin in fluentd-elasticsearch is unchanged at v2.1.6. (#71180)
- fluentd-gcp is unchanged at v3.2.0 as of Kubernetes 1.13. (#70954)
- OIDC authentication is unchanged at coreos/go-oidc v2 as of Kubernetes 1.10. (#58544)
- Calico is unchanged at v3.3.1 as of Kubernetes 1.13. (#70932)
- GLBC remains unchanged at v1.2.3 as of Kubernetes 1.12. (#66793)
- Ingress-gce remains unchanged at v1.2.3 as of Kubernetes 1.12. (#66793)

Removed

- Remove deprecated github.com/kardianos/osext dependency (#80142)

Detailed go Dependency Changes

Added

- github.com/Azure/go-autorest/autorest/adal: v0.5.0
- github.com/Azure/go-autorest/autorest/date: v0.1.0
- github.com/Azure/go-autorest/autorest/mocks: v0.2.0
- github.com/Azure/go-autorest/autorest/to: v0.2.0
- github.com/Azure/go-autorest/autorest/validation: v0.1.0
- github.com/Azure/go-autorest/autorest: v0.9.0
- github.com/Azure/go-autorest/logger: v0.1.0
- github.com/Azure/go-autorest/tracing: v0.5.0
- github.com/armon/consul-api: eb2c6b5
- github.com/bifurcation/mint: 93c51c6
- github.com/caddyserver/caddy: v1.0.3
- github.com/cenkalti/backoff: v2.1.1+incompatible
- github.com/checkpoint-restore/go-criu: bdb7599
- github.com/cheekybits/genny: 9127e81
- github.com/coredns/corefile-migration: v1.0.2
- github.com/coreos/go-etcd: v2.0.0+incompatible
- github.com/dustin/go-humanize: v1.0.0
- github.com/fatih/color: v1.6.0
- github.com/flynn/go-shlex: 3f9db97
- github.com/go-acme/lego: v2.5.0+incompatible
- github.com/go-bindata/go-bindata: v3.1.1+incompatible
- github.com/go-logr/logr: v0.1.0
- github.com/google/martian: v2.1.0+incompatible
- github.com/google/pprof: 3ea8567
- github.com/google/renameio: v0.1.0
- github.com/googleapis/gax-go/v2: v2.0.4
- github.com/hashicorp/go-syslog: v1.0.0
- github.com/jimstudt/http-authentication: 3eca13d
- github.com/kisielk/errcheck: v1.2.0
- github.com/kisielk/gotool: v1.0.0
- github.com/klauspost/cpuid: v1.2.0
- github.com/kr/pty: v1.1.5
- github.com/kylelemons/godebug: d65d576
- github.com/lucas-clemente/aes12: cd47fb3
- github.com/lucas-clemente/quic-clients: v0.1.0
- github.com/lucas-clemente/quic-go-certificates: d2f8652

- github.com/lucas-clemente/quic-go: v0.10.2
- github.com/marten-seemann/qtls: v0.2.3
- github.com/mattn/go-colorable: v0.0.9
- github.com/mattn/go-isatty: v0.0.3
- github.com/mholt/certmagic: 6a42ef9
- github.com/mitchellh/go-homedir: v1.1.0
- github.com/naoina/go-stringutil: v0.1.0
- github.com/naoina/toml: v0.1.1
- github.com/rogpeppe/go-internal: v1.3.0
- github.com/thecodeteam/goscaleio: v0.1.0
- github.com/ugorji/go/codec: d75b2dc
- github.com/xordataexchange/crypt: b2862e3
- go.opencensus.io: v0.21.0
- golang.org/x/mod: 4bf6d31
- gopkg.in/airbrake/gobrake.v2: v2.0.9
- gopkg.in/errgo.v2: v2.1.0
- gopkg.in/gemnasium/logrus-airbrake-hook.v2: v2.1.2
- gopkg.in/mcuadros/go-syslog.v2: v2.2.1
- gotest.tools/gotestsum: v0.3.5
- honnef.co/go/tools: v0.0.1-2019.2.2

Changed

- cloud.google.com/go: v0.34.0 â†’ v0.38.0
- github.com/Azure/azure-sdk-for-go: v21.4.0+incompatible â†’ v32.5.0+incompatible
- github.com/BurntSushi/toml: v0.3.0 â†’ v0.3.1
- github.com/GoogleCloudPlatform/k8s-cloud-provider: f8e9959 â†’ 27a4ced
- github.com/PuerkitoBio/purell: v1.1.0 â†’ v1.1.1
- github.com/asaskevich/govalidator: f9ffefc â†’ f61b66f
- github.com/client9/misspell: 9ce5d97 â†’ v0.3.4
- github.com/containernetworking/cni: v0.6.0 â†’ v0.7.1
- github.com/coreos/etcd: v3.3.13+incompatible â†’ v3.3.15+incompatible
- github.com/coreos/go-oidc: 065b426 â†’ v2.1.0+incompatible
- github.com/coreos/go-semver: e214231 â†’ v0.3.0
- github.com/cpuguy83/go-md2man: v1.0.4 â†’ v1.0.10
- github.com/cyphar/filepath-securejoin: ae69057 â†’ v0.2.2
- github.com/dgrijalva/jwt-go: 01aeca5 â†’ v3.2.0+incompatible
- github.com/docker/distribution: edc3ab2 â†’ v2.7.1+incompatible
- github.com/emicklei/go-restful: ff4f55a â†’ v2.9.5+incompatible
- github.com/evanphx/json-patch: 5858425 â†’ v4.2.0+incompatible
- github.com/fatih/camelcase: f6a740d â†’ v1.0.0
- github.com/go-openapi/analysis: v0.17.2 â†’ v0.19.2
- github.com/go-openapi/errors: v0.17.2 â†’ v0.19.2
- github.com/go-openapi/jsonpointer: v0.19.0 â†’ v0.19.2
- github.com/go-openapi/jsonreference: v0.19.0 â†’ v0.19.2
- github.com/go-openapi/loads: v0.17.2 â†’ v0.19.2
- github.com/go-openapi/runtime: v0.17.2 â†’ v0.19.0
- github.com/go-openapi/spec: v0.17.2 â†’ v0.19.2
- github.com/go-openapi/strfmt: v0.17.0 â†’ v0.19.0
- github.com/go-openapi/swag: v0.17.2 â†’ v0.19.2
- github.com/go-openapi/validate: v0.18.0 â†’ v0.19.2
- github.com/godbus/dbus: c7fdd8b â†’ v4.1.0+incompatible
- github.com/gogo/protobuf: 342cbe0 â†’ 65acae2
- github.com/golang/mock: bd3c8e8 â†’ v1.2.0

- github.com/golang/protobuf: v1.2.0 â†’ v1.3.1
- github.com/google/btree: 7d79101 â†’ 4030bb1
- github.com/google/cadvisor: 9db8c7d â†’ v0.34.0
- github.com/google/gofuzz: 24818f7 â†’ v1.0.0
- github.com/google/uuid: v1.0.0 â†’ v1.1.1
- github.com/gophercloud/gophercloud: c818fa6 â†’ v0.1.0
- github.com/gorilla/websocket: 4201258 â†’ v1.4.0
- github.com/grpc-ecosystem/go-grpc-prometheus: 2500245 â†’ v1.2.0
- github.com/hashicorp/golang-lru: v0.5.0 â†’ v0.5.1
- github.com/hashicorp/hcl: d8c773c â†’ v1.0.0
- github.com/heketi/heketi: 558b292 â†’ v9.0.0+incompatible
- github.com/jonboulle/clockwork: 72f9bd7 â†’ v0.1.0
- github.com/json-iterator/go: ab8a2e0 â†’ v1.1.7
- github.com/kr/pretty: f31442d â†’ v0.1.0
- github.com/kr/text: 6807e77 â†’ v0.1.0
- github.com/libopenstorage/openstorage: 093a0c3 â†’ v1.0.0
- github.com/magiconair/properties: 61b492c â†’ v1.8.1
- github.com/mailru/easyjson: 60711f1 â†’ 94de47d
- github.com/mattn/go-shellwords: f8471b0 â†’ v1.0.5
- github.com/miekg/dns: 5d001d0 â†’ v1.1.4
- github.com/mistifyio/go-zfs: 1b4ae6f â†’ v2.1.1+incompatible
- github.com/mitchellh/go-wordwrap: ad45545 â†’ v1.0.0
- github.com/mvdan/xurls: 1b768d7 â†’ v1.1.0
- github.com/onsi/ginkgo: v1.6.0 â†’ v1.8.0
- github.com/onsi/gomega: 5533ce8 â†’ v1.5.0
- github.com/opencontainers/go-digest: a6d0ee4 â†’ v1.0.0-rc1
- github.com/opencontainers/image-spec: 372ad78 â†’ v1.0.1
- github.com/opencontainers/runc: f000fe1 â†’ 6cc5158
- github.com/opencontainers/selinux: 4a2974b â†’ v1.2.2
- github.com/robfig/cron: df38d32 â†’ v1.1.0
- github.com/russross/blackfriday: 300106c â†’ v1.5.2
- github.com/seccomp/libseccomp-golang: 1b506fc â†’ v0.9.1
- github.com/sirupsen/logrus: v1.2.0 â†’ v1.4.2
- github.com/spf13/afero: b28a7ef â†’ v1.2.2
- github.com/spf13/cast: e31f36f â†’ v1.3.0
- github.com/spf13/cobra: c439c4f â†’ v0.0.5
- github.com/spf13/jwalterweatherman: 33c24e7 â†’ v1.1.0
- github.com/spf13/pflag: v1.0.1 â†’ v1.0.3
- github.com/spf13/viper: 7fb2782 â†’ v1.3.2
- github.com/stretchr/objx: v0.1.1 â†’ v0.2.0
- github.com/stretchr/testify: v1.2.2 â†’ v1.3.0
- golang.org/x/net: 65e2d4e â†’ cdfb69a
- golang.org/x/tools: aa82965 â†’ 6e04913
- google.golang.org/api: 583d854 â†’ 5213b80
- google.golang.org/genproto: 09f6ed2 â†’ 54afdca
- google.golang.org/grpc: v1.13.0 â†’ v1.23.0
- gopkg.in/check.v1: 20d25e2 â†’ 788fd78
- gopkg.in/natefinch/lumberjack.v2: 20b71e5 â†’ v2.0.0
- gopkg.in/square/go-jose.v2: 89060de â†’ v2.2.2
- gopkg.in/yaml.v2: v2.2.1 â†’ v2.2.2
- k8s.io/gengo: f8a0810 â†’ 26a6646
- k8s.io/klog: v0.3.1 â†’ v0.4.0
- k8s.io/kube-openapi: b3a7cee â†’ 743ec37
- k8s.io/utils: c2654d5 â†’ 581e001

- sigs.k8s.io/structured-merge-diff: e85c7b2 â†’ 6149e45

Removed

- github.com/Azure/go-autorest: v11.1.2+incompatible
- github.com/codedellemc/goscaleio: 20e2ce2
- github.com/d2g/dhcp4: a1d1b6c
- github.com/d2g/dhcp4client: 6e570ed
- github.com/jteeuwen/go-bindata: a0ff256
- github.com/kardianos/osext: 8fef92e
- github.com/kr/fs: 2788f0d
- github.com/marstr/guid: 8bdf7d1
- github.com/mholt/caddy: 2de4950
- github.com/natefinch/lumberjack: v2.0.0+incompatible
- github.com/pkg/sftp: 4d0e916
- github.com/shurcooL/sanitized_anchor_name: 10ef21a
- github.com/sigma/go-inotify: c87b6cf
- github.com/vmware/photon-controller-go-sdk: 4a435da
- github.com/xanzy/go-cloudstack: 1e2cbf6
- gopkg.in/yaml.v1: 9f9df34
- v1.16.0-rc.2
- v1.16.0-rc.1
- v1.16.0-beta.2
- v1.16.0-beta.1
- v1.16.0-alpha.3
- v1.16.0-alpha.2
- v1.16.0-alpha.1

v1.16.0-rc.2

Documentation

Downloads for v1.16.0-rc.2

filename	sha512 hash
kubernetes.tar.gz	68837f83bcf380e22b50f145fb64404584e96e5714a6c0cbc1ba76e290dc26
kubernetes-src.tar.gz	922552ed60d425fa6d126ffb34db6a7f123e1b9104e751edaed57b49928266

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	d0df8f57f4d9c2822badc507345f82f87d0e8e49c79ca907a0e4e4dd634d
kubernetes-client-darwin-amd64.tar.gz	0bc7daaf1165189b57dcdbe59f402731830b6f4db53b8533500568226025
kubernetes-client-linux-386.tar.gz	7735c607bb99b47924140a6a3e794912b2b97b6b54024af1de5db6765b8c

filename	sha512 hash
kubernetes-client-linux-amd64.tar.gz	d35f70cea4780a80c24588bc760c38c138d73e5f80f9fe89d952075c24cb
kubernetes-client-linux-arm.tar.gz	e1fc50b6884c42e92649a231db60e35d4e13e58728e4af7f6eca8b0baa71
kubernetes-client-linux-arm64.tar.gz	defc25fe403c20ef322b2149be28a5b44c28c7284f11bcf193a07d7f4511
kubernetes-client-linux-ppc64le.tar.gz	e87b16c948d09ddbc5d6e3fab05ad3c5a58aa7836d4f42c59edab6404655
kubernetes-client-linux-s390x.tar.gz	2c25a1860fa81cea05a1840d6a200a3a794cc50cfe45a4efec57d7122208
kubernetes-client-windows-386.tar.gz	267654a7ecfa37c800c1c94ea78343f5466783881cfac62091cfbd8c6248
kubernetes-client-windows-amd64.tar.gz	bd4c25b80e54f9fc0c07f64550d020878f899e4e3a28ca57dd532fdbab9a

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	13a93bb9bd5599b669af7bd25537ee81cefd6d8c73bedfbac845703c01950c70
kubernetes-server-linux-arm.tar.gz	781d127f32d8479bc21beed855ec73e383702e6e982854138adce8edb0ee4d1d
kubernetes-server-linux-arm64.tar.gz	6d6dfa49288e4a4ce77ca4f7e83a51c78a2b1844dd95df10cb12fff5a104e750
kubernetes-server-linux-ppc64le.tar.gz	97804d87ea984167fdbdedcfb38380bd98bb2ef150c1a631c6822905ce527093
kubernetes-server-linux-s390x.tar.gz	d45bd651c7f4b6e62ceb661c2ec70afca06a8d1fde1e50bb7783d05401c37823

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	42c57b59ce43f8961e427d622ee9cfa85cc2346877994526 2d59aa8cd31afd49
kubernetes-node-linux-arm.tar.gz	034a5611909df462ef6408f5ba5ff5ebfb4e1178b2ad06a59097560040c4fcdb
kubernetes-node-linux-arm64.tar.gz	df1493fa2d67b59eaf02096889223bbf0d71797652d3cbd89e8a3106ff6012ea

filename	sha512 hash
kubernetes-node-linux-ppc64le.tar.gz	812a5057bbf832c93f741cc39d04fc0087e36b81b6b123ec5ef02465f7ab145c
kubernetes-node-linux-s390x.tar.gz	2a58a4b201631789d4309ddc665829aedcc05ec4fe6ad6e4d965ef3283a381b8
kubernetes-node-windows-amd64.tar.gz	7fb09e7667715f539766398fc1bbbc4bf17c64913ca09d4e3535dfc4d1ba2bf6

Changelog since v1.16.0-rc.1

Other notable changes

- Single static pod files and pod files from http endpoints cannot be larger than 10 MB. HTTP probe payloads are now truncated to 10KB. (#82669, @rphillips)
- Restores compatibility with <=1.15.x custom resources by not publishing OpenAPI for non-structural custom resource definitions (#82653, @liggitt)
- Fixes regression in logging spurious stack traces when proxied connections are closed by the backend (#82588, @liggitt)

v1.16.0-rc.1

Documentation

Downloads for v1.16.0-rc.1

filename	sha512 hash
kubernetes.tar.gz	2feadb470a8b0d498dff2c122d792109bc48e24bfc7f49b4b2b40a268061c8
kubernetes-src.tar.gz	6d8877e735e041c989c0fca9dd9e57e5960299e74f66f69907b5e1265419c6

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	27bbfcb709854a9625dbb22c357492c1818bc1986b94e8cf727c046d596c4
kubernetes-client-darwin-amd64.tar.gz	9c2ea22e188282356cd532801cb94d799bde5a5716f037b81e7f83273f69
kubernetes-client-linux-386.tar.gz	bbba78b8f972d0c247ed11e88010fc934a694efce8d2605635902b4a22f5
kubernetes-client-linux-amd64.tar.gz	f2f04dc9b93d1c8f5295d3f559a3abdd19ea7011741aa006b2cd96542c06
kubernetes-client-linux-arm.tar.gz	77d1f5b4783f7480d879d0b7682b1d46e730e7fb8edbc6eccd96986c31ce

filename	sha512 hash
kubernetes-client-linux-arm64.tar.gz	0b57aa1dbbce51136789cb373d93e641d1f095a4bc9695d60917e85c814c8
kubernetes-client-linux-ppc64le.tar.gz	847065d541dece0fc931947146dbc90b181f923137772f26c7c93476e022
kubernetes-client-linux-s390x.tar.gz	d7e8a808da9e2551ca7d8e7cb25222cb9ac01595f78ebbc86152ae1c2162
kubernetes-client-windows-386.tar.gz	c9cf6a6b9f2f29152af974d30f3fd97ba33693d5cbbf8fc76bcf6590979e
kubernetes-client-windows-amd64.tar.gz	ebea0c0b64d251e6023e8a5a100aa609bc278c797170765da2e35c8997ef

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	2fe7ccce15e705826c4ccfce48df8130ba89a0c930bca4b61f49267e9d490f57
kubernetes-server-linux-arm.tar.gz	6eb77e59095a1de9eb21e7065e8d10b7d0baf1888991a42089ede6d4f8a8cac0
kubernetes-server-linux-arm64.tar.gz	429ce0d5459384c9d3a2bb103924eebc4c30343c821252dde8f4413fcf29cc73
kubernetes-server-linux-ppc64le.tar.gz	18041d9c99efc00c8c9dbb6444974efdbf4969a4f75faea75a3c859b1ee8485d
kubernetes-server-linux-s390x.tar.gz	2124c3d8856e50ca6b2b61d83f108ab921a1217fac2a80bf765d51b68f4e67d5

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	ea1bcd8cc51fbc95058a8a592eb454c07ab5dadc1c237bbc59f278f8adc46bda
kubernetes-node-linux-arm.tar.gz	e5d62df5fd086ff5712f59f71ade9efcf617a13c567de965ce54c79f3909372b
kubernetes-node-linux-arm64.tar.gz	5aa0a7a3d02b65253e4e814e51cea6dd895170f2838fea02f94e4efd3f938dbf
kubernetes-node-linux-ppc64le.tar.gz	f54bc5ae188f8ecb3ddcae20e06237430dd696f444a5c65b0aa3be79ad85c5b5
kubernetes-node-linux-s390x.tar.gz	afa4f9b747fff20ed03d40092a2df60dbd6ced0de7fd0c83c001866c4fe5b711

filename	sha512 hash
kubernetes-node-windows-amd64.tar.gz	e9b76014a1d4268ad66ade06883dd3344c6312ece14ee988af645bdf9c5e9b62

Changelog since v1.16.0-beta.2

Other notable changes

- Update Cluster Autoscaler to 1.16.0; changelog: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.16.0 (#82501, @losipiuk)
- Resolved regression serving custom resources with unhandled validation schemas with the ServerSideApply feature enabled (#82438, @liggitt)
- Fix filter plugins are not been called during preemption (#81876, @wgliang)
- Fix the dns suffix search list for GCE window clusters. (#82314, @lzang)
- Install and start logging agent based on kube env ENABLE_NODE_LOGGING. (#81300, @liyanhui1228)
- kubeadm: Allow users to skip the kube-proxy init addon phase during init and still be able to join a cluster and perform some other minor operations (but not upgrade). (#82248, @rosti)
- Bump metrics-server to v0.3.4 (#82322, @olagacek)
- Updated default etcd service used by kubernetes to 3.3.15 (#82199, @dims)

v1.16.0-beta.2

Documentation

Downloads for v1.16.0-beta.2

filename	sha512 hash
kubernetes.tar.gz	d1f4e9badc6a4422b9a261a5375769d63f0cac7fff2aff4122a325417b77d5
kubernetes-src.tar.gz	2ab20b777311746bf9af0947a2bea8ae36e27da7d917149518d7c2d2612f51

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	55523fd5cfce0c5b79e981c6a4d5572790cfe4488ed23588be45ee13367e
kubernetes-client-darwin-amd64.tar.gz	13e696782713da96f5fb2c3fa54d99ca40bc71262cb2cbc8e77a6d19ffd3
kubernetes-client-linux-386.tar.gz	7f4818599b84712edd2bf1d94f02f9a53c1f827b428a888356e793ff62e8
kubernetes-client-linux-amd64.tar.gz	8a2656289d7d86cbded42831f6bc660b579609622c16428cf6cc782ac8b5
kubernetes-client-linux-arm.tar.gz	418606bc109b9acb2687ed297fa2eec272e8cb4ad3ce1173acd15a4b43ce

filename	sha512 hash
kubernetes-client-linux-arm64.tar.gz	2eb943b745c270cd161e01a12195cfb38565de892a1da89e851495fb6f9d
kubernetes-client-linux-ppc64le.tar.gz	262e7d61e167e7accd43c47e9ce28323ae4614939a5af09ecc1023299cd2
kubernetes-client-linux-s390x.tar.gz	8f0cfe669a211423dd697fdab722011ea9641ce3db64debafa539d4a424d
kubernetes-client-windows-386.tar.gz	b1deab89653f4cd3ad8ad68b8ec3e1c038d1ef35bd2e4475d71d4781acf0
kubernetes-client-windows-amd64.tar.gz	0e3b5150767efd0ed5d60b2327d2b7f6f2bda1a3532fca8e84a7ca161f6e

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	32688295df1fcdb9472ed040dc5e8b19d04d62789d2eca64cfe08080d08ffee1
kubernetes-server-linux-arm.tar.gz	c8ea6d66e966889a54194f9dce2021131e9bae34040c56d8839341c47fc4074d
kubernetes-server-linux-arm64.tar.gz	12b42cfa33ff824392b81a604b7edcab95ecc67cddfc24c47ef67adb356a3339
kubernetes-server-linux-ppc64le.tar.gz	e03f0eba181c03ddb7535e56ff330dafebb7dcb40889fd04f5609617ebb717f9
kubernetes-server-linux-s390x.tar.gz	4e7bd061317a3445ad4b6b308f26218777677a1fef5fda181ee1a19e532a758f

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	dc5606c17f0191afc6f28dce5ab566fd8f21a69fa3989a1c8f0976d7b8ccd32e
kubernetes-node-linux-arm.tar.gz	3d5d9893e06fd7be51dca11182ecb9e93108e86af40298fe66bb62e5e86f0bf4
kubernetes-node-linux-arm64.tar.gz	fd18a02f32aeafc5cce8f3f2eadd0e532857bd5264b7299b4e48f458f77ebaa5
kubernetes-node-linux-ppc64le.tar.gz	703afd80140db2fae897d83b3d2bc8889ff6c6249bb79be7a1cce6f0c9326148
kubernetes-node-linux-s390x.tar.gz	445d4ef4f9d63eabe3b7c16114906bc450cfde3e7bf7c8aedd084c79a5e399bd

filename	sha512 hash
kubernetes-node-windows-amd64.tar.gz	`88b04171c3c0134044b7555fbc9b88071f5a73dbf2dac21f8a27b394b0870dff`

Changelog since v1.16.0-beta.1

Other notable changes

- Fix a bug in apiserver that could cause a valid update request to be rejected with a precondition check failure. (#82303, @roycaihw)
- Webhook client credentials configured with `--admission-control-config-file` must include non-default ports in the configured hostnames. For example, a webhook configured to speak to port 8443 on service `mysvc` in namespace `myns` would specify client credentials in a stanza with `name: mysvc.myns.svc:8443`. See https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#authenticate-apiservers for more details. (#82252, @liggitt)
- Ensure the KUBE-MARK-DROP chain in kube-proxy mode=ipvs. The chain is ensured for both ipv4 and ipv6 in dual-stack operation. (#82214, @uablrek)
- `kubectl cp` no longer supports copying symbolic links from containers; to support this use case, see `kubectl exec --help` for examples using `tar` directly (#82143, @soltysh)
- kubeadm: fix for HTTPProxy check for IPv6 addresses (kubernetes/kubeadm#1769) (#82267, @kad)
- Add PodOverhead awareness to kubectl (#81929, @egernst)
- The nbf (not before) claim, if present in ID token, is now enforced. (#81413, @anderseknert)
- Server-side apply will now use the openapi provided in the CRD validation field to help figure out how to correctly merge objects and update ownership. (#77354, @jennybuckley)
- - Fix disk stats in LXD using ZFS storage pool (#81972, @dashpole)
  - - Fix CRI-O missing network metris bug
  - - Add `container_sockets`, `container_threads`, and `container_threads_max` metrics
- Adds Endpoint Slice support for kubectl when discovery API group is enabled. (#81795, @robscott)
- Node-Problem-Detector v0.7.1 is used for addon daemonset. (#82140, @wangzhen127)
- aggregated discovery requests can now timeout. Aggregated apiserver *must* complete discovery calls within five seconds. Other requests can take longer. (#82146, @deads2k)
  - Use feature gate `EnableAggregatedDiscoveryTimeout=false` if you *must* remove this check, but that feature gate will be removed next release.
- Graduating Windows GMSA support from alpha to beta (#82110, @wk8)
- Add UnschedulableAndUnresolvable status code for Scheduler Framework (#82034, @alculquicondor)
- Kubeadm now includes CoreDNS version 1.6.2 (#82127, @rajansandeep)
  - - The CoreDNS Deployment now checks readiness via the `ready` plugin.
  - - The `proxy` plugin has been deprecated. The `forward` plugin is to be used instead.
  - - `kubernetes` plugin removes the `resyncperiod` option.
  - - The `upstream` option is deprecated and ignored if included.
- Make kubectl get -ignore-not-found continue processing when encountering error. (#82120, @soltysh)

- Dual stack services (Phase II of IPv6DualStack feature) are enabled via the IPVS proxier. iptables proxier does not support dualstack yet. Dualstack iptables proxier is WIP and should catchup soon. (#82091, @khenidak)
    - to enable, kube-proxy must be have the following flags:
    - -proxy-mode=ipvs
    - -cluster-cidrs=,
- The apiserver now uses http/1.1 to communicate with admission webhooks, opening multiple connections to satisfy concurrent requests, and allowing spreading requests across multiple backing pods. (#82090, @liggitt)
- Added support to specify a global-access annotation for gce ILB. (#81549, @prameshj)
- Added new startupProbe, related to KEP https://github.com/kubernetes/enhancements/issues/950. (#77807, @matthyx)
- Adds \livez for liveness health checking for kube-apiserver. Using the parameter `--maximum-startup-sequence-duration` will allow the liveness endpoint to defer boot-sequence failures for the specified duration period. (#81969, @logicalhan)
- Server-side apply is now Beta. (#81956, @apelisse)
- The `rejected` label in `apiserver_admission_webhook_admission_duration_seconds` metrices now properly indicates if a request was rejected. Add a new counter metrics `apiserver_admission_webhook_rejection_count` with details about the causing for a webhook rejection. (#81399, @roycaihw)
- Add `container_state` label to `running_container_count` kubelet metrics, to get count of containers based on their state(running/exited/created/unknown) (#81573, @irajdeep)
- Fix a bug in CRD openapi controller that user-defined CRD can overwrite OpenAPI definition/path for the CRD API. (#81436, @roycaihw)
- Service account tokens now include the JWT Key ID field in their header. (#78502, @ahmedtd)
- Adds EndpointSlice integration to kube-proxy, can be enabled with EndpointSlice feature gate. (#81430, @robscott)
- Azure supports IPv6 only on ELB not ILB. The cloud provider will return an error if the service is internal and is IPv6. (#80485, @khenidak)
    - Notes on LB name:
    - to ensure backword and forward compat:
    - - SingleStack -v4 (pre v1.16) => BackendPool name == clusterName
    - - SingleStack -v6 => BackendPool name == clusterName (all cluster bootstrap uses this name)
    - DualStack:
    - => IPv4 BackendPool name == clusterName
    - => IPv6 BackendPool name == -IPv6
    - This result into:
    - - clusters moving from IPv4 to duakstack will require no changes
    - - clusters moving from IPv6 (while not seen in the wild, we can not rule out thier existance) to dualstack will require deleting backend pools (the reconciler will take care of creating correct backendpools)
- Promotes VolumePVCDataSource (Cloning) feature to beta for 1.16 release (#81792, @j-griffith)
- Remove kubectl log, use kubectl logs instead (#78098, @soltysh)
- CSI ephemeral inline volume support is beta, i.e. the CSIInlineVolume feature gate is enabled by default (#82004, @pohly)
- kubectl: the -all-namespaces flag is now honored by `kubectl wait` (#81468, @ashutoshgngwr)
- Kube-proxy metrics are now marked as with the ALPHA stability level. (#81626, @logicalhan)

- Kube-controller-manager and cloud-controller-manager metrics are now marked as with the ALPHA stability level. (#81624, @logicalhan)
- Adds Endpoint Slice Controller for managing new EndpointSlice resource, disabled by default. (#81048, @robscott)
- + to run: (#79386, @khenidak)
    - Master: convert service CIDR to list `--service-cluster-ip-range=<CIDR>,<CIDR>` and make sure `IPv6DualStack` feature flag is turned on. The flag is validated and used as the following:
    - 1. `--service-cluster-ip-range[0]` is consider primary service range, and will be used for any service with `Service.Spec.IPFamily = nil` or any service in the at the time of turning on the feature flag.
    - 2. A cluster can be dualstack (i.e. Pods and nodes carry dualstack IPs) but does not need to support ingress on dualstack. In this case the cluster can perform egress using `PodIPs` (according to family and binding selection in user code) but will ingress will only be performed against the pod primary IP. This can be configured by supplying single entry to `--service-cluster-ip-range` flag.
    - 3. Maximum of two entries is allowed in `--service-cluster-ip-range` and they are validated to be dual stacked `i.e. --service-cluster-ip-range=<v4>,<v6> or --service-cluster-ip-range=<v6>,<v4>`
    - 4. Max 20 bit for range (min network bits `<v6>/108` or /12)
    - kube-controller-manager: convert service CIDR to list `--service-cluster-ip-range=<CIDR>,<CIDR>` and make sure `IPv6DualStack` feature flag is turned on. The flag is validated as above.
    - + to use:
    - A new service spec field `Service.Spec.IPFamily` has been added. The default of this field is family of (first service cidr in -service-cluster-ip-range flag). The value is defaulted as described above once the feature gate is turned on. Here are the possible values for this field:
    - 2. IPv4: api-server will assign an IP from a `service-cluster-ip-range` that is `ipv4` (either the primary or the secondary, according to how they were configured).
    - 2. IPv6: api-server will assign an IP from a `service-cluster-ip-range` that is `ipv6` (either the primary or the secondary, according to how they were configured).
    - Notes (v1.16):
    - 1. IPVS is the only proxy supported (as of v1.16) by Dualstack.
    - 2. Dualstack is mutually exclusive with `EndpointSlice` feature. They can not be turned on together. `metaproxy` is yet to implement EndpointSlice handling.
- Kubelet metrics for /metrics and /metrics/probes are now marked as with the ALPHA stability level. (#81534, @logicalhan)
- Added metrics â€˜authentication_attempts' that can be used to understand the attempts of authentication. (#81509, @RainbowMango)
- Fix in kube-proxy for SCTP nodeport service which only works for node's InternalIP, but doesn't work for other IPs present in the node when ipvs is enabled. (#81477, @paulsubrata55)
- The `CustomResourceValidation`, `CustomResourceSubresources`, `CustomResourceWebhookConversion` and `CustomResourcePublishOpenAPI` features are now GA, and the associated feature gates deprecated and will be removed in v1.18. (#81965, @roycaihw)
- Node-Problem-Detector v0.7.1 is used on GCI. (#80726, @wangzhen127)
- kubeadm: prevent overriding of certain kubelet security configuration parameters if the user wished to modify them (#81903, @jfbai)
- Introduce `node.kubernetes.io/exclude-balancer` and `node.kubernetes.io/exclude-disruption` labels in alpha to prevent cluster deployers from being

dependent on the optional `node-role` labels which not all clusters may provide. (#80238, @smarterclayton)
- Scheduler metrics are now marked as with the ALPHA stability level. (#81576, @logicalhan)
- cache-control headers are now set appropriately. Only openapi is cacheable if etags match. (#81946, @deads2k)
- Added E2E tests validating WindowsOptions.RunAsUserName. (#79539, @bclau)
- Kube-apiserver metrics are now marked as with the ALPHA stability level. (#81531, @logicalhan)
- Move CSI volume expansion to beta. (#81467, @bertinatto)
- Support Kubelet plugin watcher on Windows nodes. (#81397, @ddebroy)
- Updates the requestedToCapacityRatioArguments to add resources parameter that allows the users to specify the resource name along with weights for each resource to score nodes based on the request to capacity ratio. (#77688, @sudeshsh)
- Finalizer Protection for Service LoadBalancers is now in Beta (enabled by default). This feature ensures the Service resource is not fully deleted until the correlating load balancer resources are deleted. (#81691, @MrHohn)
- Adds support for vSphere volumes on Windows (#80911, @gab-satchi)
- Log when kube-apiserver regenerates the OpenAPI spec and why. OpenAPI spec generation is a very CPU-heavy process that is sensitive to continuous updates of CRDs and APIServices. (#81786, @sttts)
    - Added metrics aggregator_openapi_v2_regeneration_count, aggregator_openapi_v2_regeneration_gauge and apiextension_openapi_v2_regeneration_count metrics counting the triggering APIService and CRDs and the reason (add, update, delete).
- Fix an issue with toleration merging & whitelist checking in the PodTolerationRestriction admission controller. (#81732, @tallclair)
- Add a helper function to decode scheduler plugin args. (#80696, @hex108)
- Add metadata.generation=1 to old CustomResources. (#82005, @sttts)
- kubeadm no longer performs IPVS checks as part of its preflight checks (#81791, @yastij)
- The RemainingItemCount feature is now beta. (#81682, @caesarxuchao)
    - remainingItemCount is the number of subsequent items in the list which are not included in this list response. If the list request contained label or field selectors, then the number of remaining items is unknown and the field will be left unset and omitted during serialization. If the list is complete (either because it is not chunking or because this is the last chunk), then there are no more remaining items and this field will be left unset and omitted during serialization. Servers older than v1.15 do not set this field. The intended use of the remainingItemCount is *estimating* the size of a collection. Clients should not rely on the remainingItemCount to be set or to be exact.
- The CustomResourceDefaulting feature is promoted to beta and enabled by default. Defaults may be specified in structural schemas via the `apiextensions.k8s.io/v1` API. See https://kubernetes.io/docs/tasks/access-kubernetes-api/custom-resources/custom-resource-definitions/#specifying-a-structural-schema for details. (#81872, @sttts)
- kubectl could scale custom resource again (#81342, @knight42)
- a CSI driver that supports ephemeral inline volumes must explicitly declare that by providing a CSIDriver object where the new "mode" field is set to "ephemeral" or "persistent+ephemeral" (#80568, @pohly)
- `framework.ExpectNoError` no longer logs the error and instead relies on using the new `log.Fail` as Gomega fail handler. (#80253, @pohly)
- Audit events now log the existence and patch of mutating webhooks. (#77824, @roycaihw) * At Metadata audit level or higher, an annotation with key "mutation.webhook.admission.k8s.io/round_{round idx}*index*{order idx}" gets logged with JSON payload indicating a webhook gets invoked for given request and whether it

mutated the object or not. * At Request audit level or higher, an annotation with key "patch.webhook.admission.k8s.io/round_{round idx}*index*{order idx}" get logged with the JSON payload logging the patch sent by a webhook for given request.
- Resolves an issue that prevented block volumes from being resized. (#81429, @huffmanca)
- Verify that CRD default values in OpenAPI specs are pruned, with the exceptions of values under `metadata`. (#78829, @sttts)
- Use PostFilter instead of Postfilter in the scheduling framework (#81800, @draveness)
    - Use PreFilter instead of Prefilter in the scheduling framework
    - Use PreBind instead of Prebind in the scheduling framework
- Fix `kubectl logs -f` for windows server containers. (#81747, @Random-Liu)
- fix azure disk naming matching issue due to case sensitive comparison (#81720, @andyzhangx)
- Fixes a bug that when there is a "connection refused" error, the reflector's ListAndWatch func will return directly but what expected is that sleep 1 second and rewatch since the specified resourceVersion. (#81634, @likakuli)
- Fixed a bug with the openAPI definition for io.k8s.apimachinery.pkg.runtime.RawExtension, which previously required a field "raw" to be specified (#80773, @jennybuckley)
- kubeadm: print the stack trace of an error for klog level -v>=5 (#80937, @neolit123)
- Fixes a problem with the iptables proxy mode that could result in long delays (#80368, @danwinship)
    - updating Service/Endpoints IPs in very large clusters on RHEL/CentOS 7.
- kubeadm: support any Linux kernel version newer than 3.10 (#81623, @neolit123)
- Added a metric â€˜apiserver_watch_events_sizes' that can be used to estimate sizes of watch events in the system. (#80477, @mborsz)
- NormalizeScore plugin set is removed from scheduler framework config API. Use ScorePlugin only. (#80930, @liu-cong)
- kubeadm reset: unmount directories under "/var/lib/kubelet" for linux only (#81494, @Klaven)
- updates fluentd-elasticsearch docker image to fluentd 1.6.3 (#80912, @monotek)
- Kubeadm now seamlessly migrates the CoreDNS Configuration when upgrading CoreDNS. (#78033, @rajansandeep)
- Introduce support for applying pod overhead to pod cgroups, if the PodOverhead feature is enabled. (#79247, @egernst)
- Windows nodes on GCE now run with Windows Defender enabled. (#81625, @pjh)

v1.16.0-beta.1

Documentation

Downloads for v1.16.0-beta.1

filename	sha512 hash
kubernetes.tar.gz	`16513ebb52b01afee26156dcd4c449455dc328d7a080ba54b3f3a4584dbd92`
kubernetes-src.tar.gz	`3933f441ebca812835d6f893ec378896a8adb7ae88ca53247fa402aee1fda0`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	`28f0a8979f956aa5b3be1c1158a3ade1b242aac332696cb604fbdba44c42`
kubernetes-client-darwin-amd64.tar.gz	`8804f60b690e5180125cf6ac6d739ad5432b364c5e0d0ee0d2f06220c86c`
kubernetes-client-linux-386.tar.gz	`8f7f86db5a496afd269b926b6baf341bbd4208f49b48fad1a44c5424812`
kubernetes-client-linux-amd64.tar.gz	`7407dc1216cac39f15ca9f75be47c0463a151a3fda7d9843a67c0043c698`
kubernetes-client-linux-arm.tar.gz	`249a82a0af7d8062f49edd9221b3823590b6d166c1bca12c787ae640d6a1`
kubernetes-client-linux-arm64.tar.gz	`3a8416d99b6ae9bb6d568ff15d1783dc521fe58c60230f38126c64a7739b`
kubernetes-client-linux-ppc64le.tar.gz	`105bf4afeccf0b314673265b969d1a7f3796ca3098afa788c43cd9ff3e14`
kubernetes-client-linux-s390x.tar.gz	`98de73accb7deba9896e14a5012a112f6fd00d6e6868e4d21f61b06605ef`
kubernetes-client-windows-386.tar.gz	`7a43f3285b0ab617990497d41ceadfbd2be2b72d433b02508c198e9d380f`
kubernetes-client-windows-amd64.tar.gz	`f3fafcffc949bd7f8657dd684c901e199b21c4812009aca1f8cf3c8bf3c3`

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	`87b46e73ae2162ee49f510da6549e57503d3ea94b3c4488f39b0b93d45603f54`
kubernetes-server-linux-arm.tar.gz	`80ba8e615497c0b9c339fbd2d6a4dda54fdbd5659abd7d8e8d448d8d8c24ba7f`
kubernetes-server-linux-arm64.tar.gz	`b4a76a5fc026b4b3b5f9666df05e46896220591b21c147982ff3d91cec7330ed`
kubernetes-server-linux-ppc64le.tar.gz	`fb435dfd5514e4cd3bc16b9e71865bff3cdd5123fc272c8cbc5956c260449e0d`
kubernetes-server-linux-s390x.tar.gz	`65ed3d372a4d03493d0a586c7f67f1236aa99f02552195f1fb58079bc2478720`

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	`c9161689532a5e995a68bb0985a983dc43d8e747a05f37849cd33062c07e5202`
kubernetes-node-linux-arm.tar.gz	`7dba9fdb290f33678983c046eb145446edb1b7479c2403f9e8bd835c3d832ab1`
kubernetes-node-linux-arm64.tar.gz	`8c435824667cd9ec7efdfb72c1d060f62ca61b285cbb9575a6e6013e20ec5b37`
kubernetes-node-linux-ppc64le.tar.gz	`2cfca30dbe49a38cd1f3c78135f60bf7cb3dae0a8ec5d7fa651e1c5949254876`
kubernetes-node-linux-s390x.tar.gz	`63bbe469ddd1be48624ef5627fef1e1557a691819c71a77d419d59d101e8e6ee`
kubernetes-node-windows-amd64.tar.gz	`07cb97d5a3b7d0180a9e22696f417422a0c043754c81ae68338aab7b520aa7c1`

Changelog since v1.16.0-alpha.3

Action Required

- scheduler.alpha.kubernetes.io/critical-pod annotation is removed. Pod priority (spec.priorityClassName) should be used instead to mark pods as critical. Action required! (#80342, @draveness)
- Removed cadvisor metric labels `pod_name` and `container_name` to match instrumentation guidelines. (#80376, @ehashman)
  - Action required: any Prometheus queries that match `pod_name` and `container_name` labels (e.g. cadvisor or kubelet probe metrics) must be updated to use `pod` and `container` instead.
- Remove DirectCodecFactory(replace with serializer.WithoutConversionCodecFactory), DirectEncoder(replace with runtime.WithVersionEncoder) and DirectDecoder(replace with runtime.WithoutVersionDecoder). action required (#79263, @draveness)

Other notable changes

- fix: detach azure disk issue using dangling error (#81266, @andyzhangx)
- Conversion webhooks can now indicate they support receiving and responding with `ConversionReview` API objects in the `apiextensions.k8s.io/v1` version by including `v1` in the `conversionReviewVersions` list in their CustomResourceDefinition. Conversion webhooks must respond with a ConversionReview object in the same apiVersion they receive. `apiextensions.k8s.io/v1` `ConversionReview` responses must specify a `response.uid` that matches the `request.uid` of the object they were sent. (#81476, @liggitt)

- The `CustomResourceDefinition` API type is promoted to `apiextensions.k8s.io/v1` with the following changes: (#79604, @liggitt)
  - Use of the new `default` feature in validation schemas is limited to v1
  - `spec.scope` is no longer defaulted to `Namespaced` and must be explicitly specified
  - `spec.version` is removed; use `spec.versions` instead
  - `spec.validation` is removed; use `spec.versions[*].schema` instead
  - `spec.subresources` is removed; use `spec.versions[*].subresources` instead
  - `spec.additionalPrinterColumns` is removed; use `spec.versions[*].additionalPrinterColumns` instead
  - `spec.conversion.webhookClientConfig` is moved to `spec.conversion.webhook.clientConfig`
  - `spec.conversion.conversionReviewVersions` is moved to `spec.conversion.webhook.conversionReviewVersions`
  - `spec.versions[*].schema.openAPIV3Schema` is now required when creating v1 CustomResourceDefinitions
  - `spec.preserveUnknownFields: true` is disallowed when creating v1 CustomResourceDefinitions; it must be specified within schema definitions as `x-kubernetes-preserve-unknown-fields: true`
  - In `additionalPrinterColumns` items, the `JSONPath` field was renamed to `jsonPath` (fixes https://github.com/kubernetes/kubernetes/issues/66531)
- openapi now advertises correctly supported patch types for custom resources (#81515, @liggitt)
- Kubelet could be run with no Azure identity without subscriptionId configured now. (#81500, @feiskyer)
  - A sample cloud provider configure is: â€˜{"vmType": "vmss", "useInstanceMetadata": true}â€˜.
- Volumes specified in a pod but not used in it are no longer unnecessarily formatted, mounted and reported in `node.status.volumesInUse`. (#81163, @jsafrane)
- kubeadm: use etcd's /health endpoint for a HTTP liveness probe on localhost instead of having a custom health check using etcdctl (#81385, @neolit123)
- kubeamd: use the -pod-network-cidr flag to init or use the podSubnet field in the kubeadm config to pass a comma separated list of pod CIDRs. (#79033, @Arvinderpal)
- Update to use go 1.12.9 (#81489, @BenTheElder)
- Update Azure SDK + go-autorest API versions (#79574, @justaugustus)
- Extender bind should respect IsInterested (#79804, @yqwang-ms)
- Add instruction to setup "Application Default Credentials" to run GCE Windows e2e tests locally. (#81337, @YangLu1031)
- Scheduler should terminate when it looses leader lock. (#81306, @ravisantoshgudimetla)
- kubelet now exports an "kubelet_evictions" metric that counts the number of pod evictions carried out by the kubelet to reclaim resources (#81377, @sjenning)
- Return error when the scoring plugin returns score out of range [0, 100]. (#81015, @draveness)
- Update to use go 1.12.8 (#81390, @cblecker)
- kube-proxy -cleanup will return the correct exit code if the cleanup was successful (#78775, @johscheuer)
- remove iSCSI volume storage cleartext secrets in logs (#81215, @zouyee)
- Use a named array instead of a score array in normalizing-score phase. (#80901, @draveness)
- If scheduler extender filtered a not found node, current scheduling round for this pod will just be skipped. (#79641, @yqwang-ms)
- Update golang/x/net dependency to bring in fixes for CVE-2019-9512, CVE-2019-9514 (#81394, @cblecker)

- Fixes CVE-2019-11250: client-go header logging (at verbosity levels >= 7) now masks `Authorization` header contents (#81330, @tedyu)
- Resolves a transient 404 response to custom resource requests during server startup (#81244, @liggitt)
- Non nil DataSource entries on PVC's are now displayed as part of `describe pvc` output. (#76463, @j-griffith)
- Fix Azure client requests stuck issues on http.StatusTooManyRequests (HTTP Code 429). (#81279, @feiskyer)
- Implement a new feature that allows applying kustomize patches to static pod manifests generated by kubeadm. (#80905, @fabriziopandini)
- Add a service annotation `service.beta.kubernetes.io/azure-pip-name` to specify the public IP name for Azure load balancer. (#81213, @nilo19)
- Fix a bug in the IPVS proxier where virtual servers are not cleaned up even though the corresponding Service object was deleted. (#80942, @gongguan)
- Add scheduling support for RuntimeClasses. RuntimeClasses can now specify nodeSelector constraints & tolerations, which are merged into the PodSpec for pods using that RuntimeClass. (#80825, @tallclair)
- etcd Docker image can be run as non-root (#79722, @randomvariable)
- kubeadm: the permissions of generated CSR files are changed from 0644 to 0600 (#81217, @SataQiu)
- Fix conflicted cache when the requests are canceled by other Azure operations. (#81282, @feiskyer)
- Fix kubelet NodeLease potential performance issues. Kubelet now will try to update lease using cached one instead of get from API Server every time. (#81174, @answer1991)
- Improves validation errors for custom resources (#81212, @liggitt)
- Improvement in Kube-proxy. Kube-proxy waits for some duration for the node to be defined. (#77167, @paulsubrata55)
- hyperkube will drop support for cloud-controller-manager in a future release (#81219, @dims)
- added an new Prometheus counter metric "sync_proxy_rules_iptables_restore_failures_total" for kube-proxy iptables-restore failures (both ipvs and iptables modes) (#81210, @figo)
- Add a `Patch` method to `ScaleInterface` (#80699, @knight42)
- switch to VM Update call in attach/detach disk operation, original CreateOrUpdate call may lead to orphaned VMs or blocked resources (#81208, @andyzhangx)
- Add a azure cloud configuration `LoadBalancerName` and `LoadBalancerResourceGroup` to allow the corresponding customizations of azure load balancer. (#81054, @nilo19)
- Update the GCE windows node image to include hot fixes since July. (#81106, @YangLu1031)
- Kubelet considers all static pods as critical. Static pods pass kubelet admission even if a node does not have enough resources. Users must ensure that they account for resources when creating static pods. (#80491, @hpandeycodeit)
- kube-apiserver: the `--basic-auth-file` flag and authentication mode is deprecated and will be removed in a future release. It is not recommended for production environments. (#81152, @tedyu)
- Fix a bug that pretty printer marshals empty byte or uint8 slice as null (#81096, @roycaihw)
- Deprecate the `--cloud-provider-gce-lb-src-cidrs` flag in the kube-apiserver. This flag will be removed once the GCE Cloud Provider is removed from kube-apiserver. (#81094, @andrewsykim)
- cloud-controller-manager binaries and docker images are no longer shipped with kubernetes releases. (#81029, @dims)

- API: the metadata.selfLink field is deprecated in individual and list objects. It will no longer be returned starting in v1.20, and the field will be removed entirely in v1.21. (#80978, @wojtek-t)

v1.16.0-alpha.3

Documentation

Downloads for v1.16.0-alpha.3

filename	sha512 hash
kubernetes.tar.gz	82bc119f8d1e44518ab4f4bdefb96158b1a3634c003fe1bc8dcd6241018944
kubernetes-src.tar.gz	bbf330b887a5839e3d3219f5f4aa38f1c70eab64228077f846da80395193b2

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	8d509bdc1ca62463cbb25548ec270792630f6a883f3194e5bdbbb3d6f856
kubernetes-client-darwin-amd64.tar.gz	1b00b3a478c210e3c3e6c346f5c4f7f43a00d5ef6acb8d9c1feaf26f913b
kubernetes-client-linux-386.tar.gz	82424207b4ef52c3722436eaaf86dbe5c93c6670fd09c2b04320251028fd
kubernetes-client-linux-amd64.tar.gz	57ba937e58755d3b7dfd19626fedb95718f9c1d44ac1c5b4c8c46d11ba0f
kubernetes-client-linux-arm.tar.gz	3a3601026e019b299a6f662b887ebe749f08782d7ed0d37a807c38a01c6b
kubernetes-client-linux-arm64.tar.gz	4cdeb2e678c6b817a04f9f5d92c5c6df88e0f954550961813fca63af4501
kubernetes-client-linux-ppc64le.tar.gz	0cc7c8f7b48f5affb679352a94e42d8b4003b9ca6f8cbeaf315d2eceddd2
kubernetes-client-linux-s390x.tar.gz	9d8fa639f543e707dc65f24ce2f8c73a50c606ec7bc27d17840f45ac150d
kubernetes-client-windows-386.tar.gz	05bf6e696da680bb8feec4f411f342a9661b6165f4f0c72c069871983f19
kubernetes-client-windows-amd64.tar.gz	b2097bc851f5d3504e562f68161910098b46c66c726b92b092a040acda96

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	721bd09b64e5c8f220332417089a772d9073c0dc5cdfa240984cfeb0d681b4a0

filename	sha512 hash
kubernetes-server-linux-arm.tar.gz	e7638ce4b88b4282f0a157593cfe809fa9cc9139ea7ebae4762ef5ac1dfaa516
kubernetes-server-linux-arm64.tar.gz	395566c4be3c2ca5b07e81221b3370bc7ccbef0879f96a9384650fcaf4f699f3
kubernetes-server-linux-ppc64le.tar.gz	90fcba066efd76d2f271a0eb26ed4d90483674d04f5e8cc39ec1e5b7f343311f
kubernetes-server-linux-s390x.tar.gz	b25014bcf4138722a710451f6e58ee57588b4d47fcceeda8f6866073c1cc0864

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	6925a71096530f7114a68e755d07cb8ba714bc60b477360c85d76d7b71d3a3c0
kubernetes-node-linux-arm.tar.gz	073310e1ccf9a8af998d4c0402ae86bee4f253d2af233b0c45cea55902268c2f
kubernetes-node-linux-arm64.tar.gz	c55e9aecef906e56a6003f441a7d336846edb269aed1c7a31cf834b073050870
kubernetes-node-linux-ppc64le.tar.gz	e89d72d27bb0a7f9133ef7310f455ba2b4c46e9852c43e0a981b68a413bcdd18
kubernetes-node-linux-s390x.tar.gz	6ef8a25f2f80a806672057dc030654345e87d269babe7cf166f7443e04c0b3a9
kubernetes-node-windows-amd64.tar.gz	22fd1cea6e0150c06dbdc7249635bbf93c4297565d5a9d13e653f9365cd61a0b

Changelog since v1.16.0-alpha.2

Action Required

- ACTION REQUIRED: (#80676, @fabriziopandini)
  - kubeadm now deletes the bootstrap-kubelet.conf file after TLS bootstrap
  - User relying on bootstrap-kubelet.conf should switch to kubelet.conf that contains node credentials

Other notable changes

- Fixes validation of VolumeAttachment API objects created with inline volume sources. (#80945, @tedyu)

- Azure disks of shared kind will no longer fail if they do not contain skuname or (#80837, @rmweir)
    - storageaccounttype.
- kubeadm: fix "certificate-authority" files not being pre-loaded when using file discovery (#80966, @neolit123)
- Errors from pod volume set up are now propagated as pod events. (#80369, @jsafrane)
- kubeadm: enable secure serving for the kube-scheduler (#80951, @neolit123)
- Kubernetes client users may disable automatic compression when invoking Kubernetes APIs by setting the `DisableCompression` field on their rest.Config. This is recommended when clients communicate primarily over high bandwidth / low latency networks where response compression does not improve end to end latency. (#80919, @smarterclayton)
- kubectl get did not correctly count the number of binaryData keys when listing config maps. (#80827, @smarterclayton)
- Implement "post-filter" extension point for scheduling framework (#78097, @draveness)
- Reduces GCE PD Node Attach Limits by 1 since the node boot disk is considered an attachable disk (#80923, @davidz627)
- This PR fixes an error when using external etcd but storing etcd certificates in the same folder and with the same name used by kubeadm for local etcd certificates; for an older version of kubeadm, the workaround is to avoid file name used by kubeadm for local etcd. (#80867, @fabriziopandini)
- When specifying `--(kube|system)-reserved-cgroup`, with `--cgroup-driver=systemd`, it is now possible to use the fully qualified cgroupfs name (i.e. `/test-cgroup.slice`). (#78793, @mattjmcnaughton)
- kubeadm: treat non-fatal errors as warnings when doing reset (#80862, @drpaneas)
- kube-addon-manager has been updated to v9.0.2 to fix a bug in leader election (https://github.com/kubernetes/kubernetes/pull/80575) (#80861, @mborsz)
- Determine system model to get credentials for windows nodes. (#80764, @liyanhui1228)
- TBD (#80730, @jennybuckley)
- The `AdmissionReview` API sent to and received from admission webhooks has been promoted to `admission.k8s.io/v1`. Webhooks can specify a preference for receiving `v1` AdmissionReview objects with `admissionReviewVersions: ["v1","v1beta1"]`, and must respond with an API object in the same `apiVersion` they are sent. When webhooks use `admission.k8s.io/v1`, the following additional validation is performed on their responses: (#80231, @liggitt) * `response.patch` and `response.patchType` are not permitted from validating admission webhooks * `apiVersion: "admission.k8s.io/v1"` is required * `kind: "AdmissionReview"` is required * `response.uid: "<value of request.uid>"` is required * `response.patchType: "JSONPatch"` is required (if `response.patch` is set)
- "kubeadm join" fails if file-based discovery is too long, with a default timeout of 5 minutes. (#80804, @olivierlemasle)
- enhance Azure cloud provider code to support both AAD and ADFS authentication. (#80841, @rjaini)
- Attempt to set the kubelet's hostname & internal IP if `--cloud-provider=external` and no node addresses exists (#75229, @andrewsykim)
- kubeadm: avoid double deletion of the upgrade prepull DaemonSet (#80798, @xlgao-zju)
- Fixes problems with connecting to services on localhost on some systems; in particular, DNS queries to systemd-resolved on Ubuntu. (#80591, @danwinship)
- Implement normalize plugin extension point for the scheduler framework. (#80383, @liu-cong)
- Fixed the bash completion error with override flags. (#80802, @dtaniwaki)
- Fix CVE-2019-11247: API server allows access to custom resources via wrong scope (#80750, @sttts)
- Failed iscsi logout is now re-tried periodically. (#78941, @jsafrane)

- Fix public IP not found issues for VMSS nodes (#80703, @feiskyer)
- In order to enable dual-stack support within kubeadm and kubernetes components, as part of the init config file, the user should set feature-gate IPv6DualStack=true in the ClusterConfiguration. Additionally, for each worker node, the user should set the feature-gate for kubelet using either nodeRegistration.kubeletExtraArgs or KUBELET_EXTRA_ARGS. (#80531, @Arvinderpal)
- Fix error in `kubeadm join --discovery-file` when using discovery files with embedded credentials (#80675, @fabriziopandini)

v1.16.0-alpha.2

Documentation

Downloads for v1.16.0-alpha.2

filename	sha512 hash
kubernetes.tar.gz	7dfa3f8b9e98e528e2b49ed9cca5e95f265b9e102faac636ff0c29e0456891
kubernetes-src.tar.gz	7cf14b92c96cab5fcda3115ec66b44562ca26ea6aa46bc7fa614fa66bda1bd

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	4871756de2cd1add0b07ec1e577c500d18a59e2f761595b939e1d4e10fbe
kubernetes-client-darwin-amd64.tar.gz	dbd9ca5fd90652ffc1606f50029d711eb52d34b707b7c04f29201f85aa8a
kubernetes-client-linux-386.tar.gz	6b049098b1dc65416c5dcc30346b82e5cf69a1cdd7e7b065429a76d302ef
kubernetes-client-linux-amd64.tar.gz	7240a9d49e445e9fb0c9d360a9287933c6c6e7d81d6e11b0d645d3f9b6f3
kubernetes-client-linux-arm.tar.gz	947b0d9aeeef08961c0582b4c3c94b7ae1016d20b0c9f50af5fe760b3573
kubernetes-client-linux-arm64.tar.gz	aff0258a223f5061552d340cda36872e3cd7017368117bbb14dc0f8a3a4d
kubernetes-client-linux-ppc64le.tar.gz	3eabecd62290ae8d876ae45333777b2c9959e39461197dbe90e6ba07d0a4
kubernetes-client-linux-s390x.tar.gz	6651b2d95d0a8dd748c33c9e8018ab606b4061956cc2b6775bd0b008b04e
kubernetes-client-windows-386.tar.gz	4b6c11b7a318e5fcac19144f6ab1638126c299e08c7b908495591674abcf
kubernetes-client-windows-amd64.tar.gz	760ae08da6045ae7089fb27a9324e77bed907662659364857e1a8d103d19

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	`69db41f3d79aa0581c36a3736ab8dc96c92127b82d3cf25c5effc675758fe713`
kubernetes-server-linux-arm.tar.gz	`ca302f53ee91ab4feb697bb34d360d0872a7abea59c5f28cceefe9237a914c77`
kubernetes-server-linux-arm64.tar.gz	`79ab1f0a542ce576ea6d81cd2a7c068da6674177b72f1b5f5e3ca47edfdb228f`
kubernetes-server-linux-ppc64le.tar.gz	`fbe5b45326f1d03bcdd9ffd46ab454917d79f629ba23dae9d667d0c7741bc2f5`
kubernetes-server-linux-s390x.tar.gz	`eb13ac306793679a3a489136bb7eb6588472688b2bb2aa0e54e61647d8c9da6d`

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	`a4bde88f3e0f6233d04f04d380d5f612cd3c574bd66b9f3ee531fa76e3e0f1c6`
kubernetes-node-linux-arm.tar.gz	`7d72aa8c1d883b9f047e5b98dbb662bdfd314f9c06af4213068381ffaac116e6`
kubernetes-node-linux-arm64.tar.gz	`c9380bb59ba26dcfe1ab52b5cb02e2d920313defda09ec7d19ccbc18f54def4b`
kubernetes-node-linux-ppc64le.tar.gz	`7bcd79b368a62c24465fce7dcb024bb629eae034e09fb522fb43bb5798478ca2`
kubernetes-node-linux-s390x.tar.gz	`9bda9dd24ee5ca65aaefece4213b46ef57cde4904542d94e6147542e42766f8b`
kubernetes-node-windows-amd64.tar.gz	`d5906f229d2d8e99bdb37e7d155d54560b82ea28ce881c5a0cde8f8d20bff8fd`

Changelog since v1.16.0-alpha.1

Action Required

- Revert "scheduler.alpha.kubernetes.io/critical-pod annotation is removed. Pod priority (spec.priorityClassName) should be used instead to mark pods as critical. Action required!" (#80277, @draveness)

- ACTION REQUIRED: container images tar files for â€˜amd64' will now contain the architecture in the RepoTags manifest.json section. (#80266, @javier-b-perez)
    - If you are using docker manifests there are not visible changes.

Other notable changes

- Use HTTPS as etcd-apiserver protocol when mTLS between etcd and kube-apiserver on master is enabled, change etcd metrics/health port to 2382. (#77561, @wenjiaswe)
- kubelet: change node-lease-renew-interval to 0.25 of lease-renew-duration (#80429, @gaorong)
- Fix error handling and potential go null pointer exception in kubeadm upgrade diff (#80648, @odinuge)
- New flag -endpoint-updates-batch-period in kube-controller-manager can be used to reduce number of endpoints updates generated by pod changes. (#80509, @mborsz)
- kubeadm: produce errors if they occur when resetting cluster status for a control-plane node (#80573, @bart0sh)
- When a load balancer type service is created in a k8s cluster that is backed by Azure Standard Load Balancer, the corresponding load balancer rule added in the Azure Standard Load Balancer would now have the "EnableTcpReset" property set to true. (#80624, @xuto2)
- Update portworx plugin dependency on libopenstorage/openstorage to v1.0.0 (#80495, @adityadani)
- Fixed detachment of deleted volumes on OpenStack / Cinder. (#80518, @jsafrane)
- when PodInfoOnMount is enabled for a CSI driver, the new csi.storage.k8s.io/ephemeral parameter in the volume context allows a driver's NodePublishVolume implementation to determine on a case-by-case basis whether the volume is ephemeral or a normal persistent volume (#79983, @pohly)
- Update gogo/protobuf to serialize backward, as to get better performance on deep objects. (#77355, @apelisse)
- Remove GetReference() and GetPartialReference() function from pkg/api/ref, as the same function exists also in staging/src/k8s.io/client-go/tools/ref (#80361, @wojtek-t)
- Fixed a bug in the CSI metrics that does not return not supported error when a CSI driver does not support metrics. (#79851, @jparklab)
- Fixed a bug in kube-addon-manager's leader election logic that made all replicas active. (#80575, @mborsz)
- Kibana has been slightly revamped/improved in the latest version (#80421, @lostick)
- kubeadm: fixed ignoring errors when pulling control plane images (#80529, @bart0sh)
- CRDs under k8s.io and kubernetes.io must have the "api-approved.kubernetes.io" set to either `unapproved.*` or a link to the pull request approving the schema. See https://github.com/kubernetes/enhancements/pull/1111 for more details. (#79992, @deads2k)
- Reduce kube-proxy cpu usage in IPVS mode when a large number of nodePort services exist. (#79444, @cezarsa)
- Add CSI Migration Shim for VerifyVolumesAreAttached and BulkVolumeVerify (#80443, @davidz627)
- Fix a bug that causes DaemonSet rolling update hang when there exist failed pods. (#78170, @DaiHao)
- Fix retry issues when the nodes are under deleting on Azure (#80419, @feiskyer)
- Add support for AWS EBS on windows (#79552, @wongma7)
- Passing an invalid policy name in the `--cpu-manager-policy` flag will now cause the kubelet to fail instead of simply ignoring the flag and running the `cpumanagers` default policy instead. (#80294, @klueska)
- Add Filter extension point to the scheduling framework. (#78477, @YoubingLi)
- cpuUsageNanoCores is now reported in the Kubelet summary API on Windows nodes (#80176, @liyanhui1228)

- `[]TopologySpreadConstraint` is introduced into PodSpec to support the "Even Pods Spread" alpha feature. (#77327, @Huang-Wei)
- kubeadm: fall back to client version in case of certain HTTP errors (#80024, @RainbowMango)
- NFS Drivers are now enabled to collect metrics, StatFS metrics provider is used to collect the metrics. (#75805 , @brahmaroutu) (#75805, @brahmaroutu)
- make node lease renew interval more heuristic based on node-status-update-frequency in kubelet (#80173, @gaorong)
- Introduction of the pod overhead feature to the scheduler. This functionality is alpha-level as of (#78319, @egernst)
    - Kubernetes v1.16, and is only honored by servers that enable the PodOverhead feature.gate.
- N/A (#80260, @khenidak)
- Add v1.Container.SecurityContext.WindowsOptions.RunAsUserName to the pod spec (#79489, @bclau)
- Pass-through volume MountOptions to global mount (NodeStageVolume) on the node for CSI (#80191, @davidz627)
- Add Score extension point to the scheduling framework. (#79109, @ahg-g)

v1.16.0-alpha.1

Documentation

Downloads for v1.16.0-alpha.1

filename	sha512 hash
kubernetes.tar.gz	`4834c52267414000fa93c0626bded5a969cf65d3d4681c20e5ae2c5f62002a`
kubernetes-src.tar.gz	`9329d51f5c73f830f3c895c2601bc78e51d2d412b928c9dae902e9ba8d4633`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	`3cedffb92a0fca4f0b2d41f8b09baa59dff58df96446e8eece4e1b81022d`
kubernetes-client-darwin-amd64.tar.gz	`14de6bb296b4d022f50778b160c98db3508c9c7230946e2af4eb2a1d662d`
kubernetes-client-linux-386.tar.gz	`8b2b9fa55890895239b99fabb866babe50aca599591db1ecf9429e49925a`
kubernetes-client-linux-amd64.tar.gz	`e927ac7b314777267b95e0871dd70c352ec0fc967ba221cb6cba523fa6f1`
kubernetes-client-linux-arm.tar.gz	`4a230a6d34e2ffd7df40c5b726fbcbb7ef1373d81733bfb75685b2448ed1`
kubernetes-client-linux-arm64.tar.gz	`87c8d7185df23b3496ceb74606558d895a64daf0c41185c833a233e29216`
kubernetes-client-linux-ppc64le.tar.gz	`16ea5efa2fc29bc7448a609a7118e7994e901ab26462aac52f03b4851d4c`

filename	sha512 hash
kubernetes-client-linux-s390x.tar.gz	7390ad1682227a70550b20425fa5287fecf6a5d413493b03df3a77956142
kubernetes-client-windows-386.tar.gz	88251896dfe38e59699b879f643704c0195e7a5af2cb00078886545f4936
kubernetes-client-windows-amd64.tar.gz	766b2a9bf097e45b2549536682cf25129110bd0562ab0df70e841ff8657d

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	dfd5c2609990c9b9b94249c654931b240dc072f2cc303e1e1d6dec1fddfb0a9e
kubernetes-server-linux-arm.tar.gz	7704c2d3c57950f184322263ac2be1649a0d737d176e7fed1897031d0efb8375
kubernetes-server-linux-arm64.tar.gz	fbbd87cc38cfb6429e3741bfd87ecec4b69b551df6fb7c121900ced4c1cd0bc7
kubernetes-server-linux-ppc64le.tar.gz	cfed5b936eb2fe44df5d0c9c6484bee38ef370fb1258522e8c62fb6a526e9440
kubernetes-server-linux-s390x.tar.gz	31768114173434726 0ad9f918fa4b67e48751f5a7df64a848d2a83c79a4e9dba

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	b3b1013453d35251b8fc4759f6ac26bdeb37f14a98697078535f7f902e8ebca5
kubernetes-node-linux-arm.tar.gz	0bacc1791d260d2863ab768b48daf66f0f7f89eeee70e68dd515b05fc9d7f14b
kubernetes-node-linux-arm64.tar.gz	73bd70cb9d27ce424828a95d715c16fd9dd22396dbe1dfe721eb0aea9e186ec4
kubernetes-node-linux-ppc64le.tar.gz	a865f98838143dc7e1e12d1e258e5f5f2855fcf6e88488fb164ad62cf886d8e2
kubernetes-node-linux-s390x.tar.gz	d2f9f746ed0fe00be982a847a3ae1b6a698d5c506be1d3171156902140fec646
kubernetes-node-windows-amd64.tar.gz	37f48a6d8174f38668bc41c81222615942bfe07e01f319bdfed409f83a3de377

Changelog since v1.15.0

Action Required

- Migrate scheduler to use v1beta1 Event API. action required: any tool targeting scheduler events needs to use v1beta1 Event API (#78447, @yastij)
- scheduler.alpha.kubernetes.io/critical-pod annotation is removed. Pod priority (spec.priorityClassName) should be used instead to mark pods as critical. Action required! (#79554, @draveness)
- hyperkube: the `--make-symlinks` flag, deprecated in v1.14, has been removed. (#80017, @Pothulapati)
- Node labels `beta.kubernetes.io/metadata-proxy-ready`, `beta.kubernetes.io/metadata-proxy-ready` and `beta.kubernetes.io/kube-proxy-ds-ready` are no longer added on new nodes. (#79305, @paivagustavo) * ip-mask-agent addon starts to use the label `node.kubernetes.io/masq-agent-ds-ready` instead of `beta.kubernetes.io/masq-agent-ds-ready` as its node selector. * kube-proxy addon starts to use the label `node.kubernetes.io/kube-proxy-ds-ready` instead of `beta.kubernetes.io/kube-proxy-ds-ready` as its node selector. * metadata-proxy addon starts to use the label `cloud.google.com/metadata-proxy-ready` instead of `beta.kubernetes.io/metadata-proxy-ready` as its node selector. * Kubelet removes the ability to set `kubernetes.io` or `k8s.io` labels via -node-labels other than the specifically allowed labels/prefixes.
- The following APIs are no longer served by default: (#70672, @liggitt) * All resources under `apps/v1beta1` and `apps/v1beta2` - use `apps/v1` instead * `daemonsets`, `deployments`, `replicasets` resources under `extensions/v1beta1` - use `apps/v1` instead * `networkpolicies` resources under `extensions/v1beta1` - use `networking.k8s.io/v1` instead * `podsecuritypolicies` resources under `extensions/v1beta1` - use `policy/v1beta1` instead
  - Serving these resources can be temporarily re-enabled using the `--runtime-config` apiserver flag.
    - `apps/v1beta1=true`
    - `apps/v1beta2=true`
    - `extensions/v1beta1/daemonsets=true,extensions/v1beta1/deployments=true,extensions/v1beta1/replicasets=true,extensions/v1beta1/networkpolicies=true,extensions/v1beta1/podsecuritypolicies=true`
  - The ability to serve these resources will be completely removed in v1.18.
- ACTION REQUIRED: Removed deprecated flag `--resource-container` from kube-proxy. (#78294, @vllry)
  - The deprecated `--resource-container` flag has been removed from kube-proxy, and specifying it will now cause an error. The behavior is now as if you specified `--resource-container=""`. If you previously specified a non-empty `--resource-container`, you can no longer do so as of kubernetes 1.16.

Other notable changes

- When HPAScaleToZero feature gate is enabled HPA supports scaling to zero pods based on object or external metrics. HPA remains active as long as at least one metric value available. (#74526, @DXist)
  - To downgrade the cluster to version that does not support scale-to-zero feature:

    - 1. make sure there are no hpa objects with minReplicas=0. Here is a oneliner to update it to 1:
    - $ kubectl get hpa -all-namespaces -no-headers=true | awk â€˜{if($6==0) printf "kubectl patch hpa/%s -namespace=%s -p \"{\"spec\":{\"minReplicas\":1}}\" ", $2, $1 }' | sh
    - 2. disable HPAScaleToZero feature gate
- Add support for writing out of tree custom scheduler plugins. (#78162, @hex108)
- Remove deprecated github.com/kardianos/osext dependency (#80142, @loqutus)
- Add Bind extension point to the scheduling framework. (#79313, @chenchun)
- On Windows systems, %USERPROFILE% is now preferred over %HOMEDRIVE%\%HOMEPATH% as the home folder if %HOMEDRIVE%\%HOMEPATH% does not contain a .kube* Add -kubernetes-version to "kubeadm init phase certs ca" and "kubeadm init phase kubeconfig" (#80115, @gyuho)
- kubeadm ClusterConfiguration now supports featureGates: IPv6DualStack: true (#80145, @Arvinderpal)
- Fix a bug that ListOptions.AllowWatchBookmarks wasn't propagating correctly in kube-apiserver. (#80157, @wojtek-t)
- Bugfix: csi plugin supporting raw block that does not need attach mounted failed (#79920, @cwdsuzhou)
- Increase log level for graceful termination to v=5 (#80100, @andrewsykim)
- kubeadm: support fetching configuration from the original cluster for â€˜upgrade diff' (#80025, @SataQiu)
- The sample-apiserver gains support for OpenAPI v2 spec serving at `/openapi/v2`. (#79843, @sttts)
    - The `generate-internal-groups.sh` script in k8s.io/code-generator will generate OpenAPI definitions by default in `pkg/generated/openapi`. Additional API group dependencies can be added via `OPENAPI_EXTRA_PACKAGES=<group>/<version> <group2>/<version2>...`.
- Cinder and ScaleIO volume providers have been deprecated and will be removed in a future release. (#80099, @dims)
- kubelet's -containerized flag was deprecated in 1.14. This flag is removed in 1.16. (#80043, @dims)
- Optimize EC2 DescribeInstances API calls in aws cloud provider library by querying instance ID instead of EC2 filters when possible (#78140, @zhan849)
- etcd migration image no longer supports etcd2 version. (#80037, @dims)
- Promote WatchBookmark feature to beta and enable it by default. (#79786, @wojtek-t)
    - With WatchBookmark feature, clients are able to request watch events with BOOKMARK type. Clients should not assume bookmarks are returned at any specific interval, nor may they assume the server will send any BOOKMARK event during a session.
- update to use go 1.12.7 (#79966, @tao12345666333)
- Add -shutdown-delay-duration to kube-apiserver in order to delay a graceful shutdown. `/healthz` will keep returning success during this time and requests are normally served, but `/readyz` will return faillure immediately. This delay can be used to allow the SDN to update iptables on all nodes and stop sending traffic. (#74416, @sttts)
- The `MutatingWebhookConfiguration` and `ValidatingWebhookConfiguration` APIs have been promoted to `admissionregistration.k8s.io/v1`: (#79549, @liggitt) * `failurePolicy` default changed from `Ignore` to `Fail` for v1 * `matchPolicy` default changed from `Exact` to `Equivalent` for v1 * `timeout` default changed from `30s` to `10s` for v1 * `sideEffects` default value is removed and the field made required for v1 * `admissionReviewVersions` default value is removed and the field made required for v1 (supported versions for AdmissionReview are `v1` and `v1beta1`) * The `name` field for specified webhooks must be unique for `Mutati`

ngWebhookConfiguration and ValidatingWebhookConfiguration objects created via admissionregistration.k8s.io/v1
    - The admissionregistration.k8s.io/v1beta1 versions of MutatingWebhookConfiguration and ValidatingWebhookConfiguration are deprecated and will no longer be served in v1.19.
- The garbage collector and generic object quota controller have been updated to use the metadata client which improves memory (#78742, @smarterclayton)
    - and CPU usage of the Kube controller manager.
- SubjectAccessReview requests sent for RBAC escalation, impersonation, and pod security policy authorization checks now populate the version attribute. (#80007, @liggitt)
- na (#79892, @mikebrow)
- Use O_CLOEXEC to ensure file descriptors do not leak to subprocesses. (#74691, @cpuguy83)
- The namespace controller has been updated to use the metadata client which improves memory (#78744, @smarterclayton)
    - and CPU usage of the Kube controller manager.
- NONE (#79933, @mm4tt)
- add kubectl replace --raw and kubectl delete --raw to have parity with create and get (#79724, @deads2k)
- E2E tests no longer add command line flags directly to the command line, test suites that want that need to be updated if they don't use HandleFlags (#75593, @pohly)
    - loading a -viper-config=e2e.yaml with suffix (introduced in 1.13) works again and now has a regression test
- Kubernetes now supports transparent compression of API responses. Clients that send Accept-Encoding: gzip will now receive a GZIP compressed response body if the API call was larger than 128KB. Go clients automatically request gzip-encoding by default and should see reduced transfer times for very large API requests. Clients in other languages may need to make changes to benefit from compression. (#77449, @smarterclayton)
- Resolves an issue serving aggregated APIs backed by services that respond to requests to / with non-2xx HTTP responses (#79895, @deads2k)
- updated fluentd to 1.5.1, elasticsearchs & kibana to 7.1.1 (#79014, @monotek)
- kubeadm: implement support for concurrent add/remove of stacked etcd members (#79677, @neolit123)
- Added a metric â€˜apiserver_watch_events_total' that can be used to understand the number of watch events in the system. (#78732, @mborsz)
- KMS Providers will install a healthz check for the status of kms-pluign in kube-apiservers' encryption config. (#78540, @immutableT)
- Fixes a bug in openapi published for custom resources using x-kubernetes-preserve-unknown-fields extensions, so that kubectl will allow sending unknown fields for that portion of the object. (#79636, @liggitt)
- A new client k8s.io/client-go/metadata.Client has been added for accessing objects generically. This client makes it easier to retrieve only the metadata (the metadata sub-section) from resources on the cluster in an efficient manner for use cases that deal with objects generically, like the garbage collector, quota, or the namespace controller. The client asks the server to return a meta.k8s.io/v1 PartialObjectMetadata object for list, get, delete, watch, and patch operations on both normal APIs and custom resources which can be encoded in protobuf for additional work. If the server does not yet support this API the client will gracefully fall back to JSON and transform the response objects into PartialObjectMetadata. (#77819, @smarterclayton)
- changes timeout value in csi plugin from 15s to 2min which fixes the timeout issue (#79529, @andyzhangx)
- kubeadm: provide "-control-plane-endpoint" flag for controlPlaneEndpoint (#79270, @SataQiu)

- Fixes invalid "time stamp is the future" error when kubectl cp-ing a file (#73982, @tanshanshan)
- Kubelet should now more reliably report the same primary node IP even if the set of node IPs reported by the CloudProvider changes. (#79391, @danwinship)
- To configure controller manager to use ipv6dual stack: (#73977, @khenidak)
    - use -cluster-cidr=",".
    - Notes:
    - 1. Only the first two cidrs are used (soft limits for Alpha, might be lifted later on).
    - 1. Only the "RangeAllocator" (default) is allowed as a value for -cidr-allocator-type . Cloud allocators are not compatible with ipv6dualstack
- When using the conformance test image, a new environment variable E2E_USE_GO_RUNNER will cause the tests to be run with the new Golang-based test runner rather than the current bash wrapper. (#79284, @johnSchnake)
- kubeadm: prevent PSP blocking of upgrade image prepull by using a non-root user (#77792, @neolit123)
- kubelet now accepts a -cni-cache-dir option, which defaults to /var/lib/cni/cache, where CNI stores cache files. (#78908, @dcbw)
- Update Azure API versions (containerregistry -> 2018-09-01, network -> 2018-08-01) (#79583, @justaugustus)
- Fix possible fd leak and closing of dirs in doSafeMakeDir (#79534, @odinuge)
- kubeadm: fix the bug that "-cri-socket" flag does not work for `kubeadm reset` (#79498, @SataQiu)
- kubectl logs -selector will support -tail=-1. (#74943, @JishanXing)
- Introduce a new admission controller for RuntimeClass. Initially, RuntimeClass will be used to apply the pod overhead associated with a given RuntimeClass to the Pod.Spec if a corresponding RuntimeClassName is specified. (#78484, @egernst)
    - PodOverhead is an alpha feature as of Kubernetes 1.16.
- Fix kubelet errors in AArch64 with huge page sizes smaller than 1MiB (#78495, @odinuge)
- The alpha `metadata.initializers` field, deprecated in 1.13, has been removed. (#79504, @yue9944882)
- Fix duplicate error messages in cli commands (#79493, @odinuge)
- Default resourceGroup should be used when the value of annotation azure-load-balancer-resource-group is an empty string. (#79514, @feiskyer)
- Fixes output of `kubectl get --watch-only` when watching a single resource (#79345, @liggitt)

- RateLimiter add a context-aware method, fix client-go request goruntine backlog in async timeout scene. (#79375, @answer1991)
- Fix a bug where kubelet would not retry pod sandbox creation when the restart policy of the pod is Never (#79451, @yujuhong)
- Fix CRD validation error on â€˜items' field. (#76124, @tossmilestone)
- The CRD handler now properly re-creates stale CR storage to reflect CRD update. (#79114, @roycaihw)
- Integrated volume limits for in-tree and CSI volumes into one scheduler predicate. (#77595, @bertinatto)
- Fix a bug in server printer that could cause kube-apiserver to panic. (#79349, @roycaihw)
- Mounts /home/kubernetes/bin/nvidia/vulkan/icd.d on the host to /etc/vulkan/icd.d inside containers requesting GPU. (#78868, @chardch)
- Remove CSIPersistentVolume feature gates (#79309, @draveness)
- Init container resource requests now impact pod QoS class (#75223, @sjenning)
- Correct the maximum allowed insecure bind port for the kube-scheduler and kube-apiserver to 65535. (#79346, @ncdc)
- Fix remove the etcd member from the cluster during a kubeadm reset. (#79326, @bradbeam)
- Remove KubeletPluginsWatcher feature gates (#79310, @draveness)
- Remove HugePages, VolumeScheduling, CustomPodDNS and PodReadinessGates feature flags (#79307, @draveness)
- The GA PodPriority feature gate is now on by default and cannot be disabled. The feature gate will be removed in v1.18. (#79262, @draveness)
- Remove pids cgroup controller requirement when related feature gates are disabled (#79073, @rafatio)
- Add Bind extension point of the scheduling framework (#78513, @chenchun)
- if targetPort is changed that will process by service controller (#77712, @Sn0rt)
- update to use go 1.12.6 (#78958, @tao12345666333)
- kubeadm: fix a potential panic if kubeadm discovers an invalid, existing kubeconfig file (#79165, @neolit123)
- fix kubelet fail to delete orphaned pod directory when the kubelet's pods directory (default is "/var/lib/kubelet/pods") symbolically links to another disk device's directory (#79094, @gaorong)
- Addition of Overhead field to the PodSpec and RuntimeClass types as part of the Pod Overhead KEP (#76968, @egernst)

- fix pod list return value of framework.WaitForPodsWithLabelRunningReady (#78687, @pohly)
- The behavior of the default handler for 404 requests fro the GCE Ingress load balancer is slightly modified in the sense that it now exports metrics using prometheus. The metrics exported include: (#79106, @vbannai)
    - - http_404_request_total (the number of 404 requests handled)
    - - http_404_request_duration_ms (the amount of time the server took to respond in ms)
    - Also includes percentile groupings. The directory for the default 404 handler includes instructions on how to enable prometheus for monitoring and setting alerts.
- The kube-apiserver has improved behavior for both startup and shutdown sequences and also now exposes `eadyz` for readiness checking. Readyz includes all existing healthz checks but also adds a shutdown check. When a cluster admin initiates a shutdown, the kube-apiserver will try to process existing requests (for the duration of request timeout) before killing the apiserver process. (#78458, @logicalhan)
    - The apiserver also now takes an optional flag "-maximum-startup-sequence-duration". This allows you to explicitly define an upper bound on the apiserver startup sequences before healthz begins to fail. By keeping the kubelet liveness initial delay short, this can enable quick kubelet recovery as soon as we have a boot sequence which has not completed in our expected time frame, despite lack of completion from longer boot sequences (like RBAC). Kube-apiserver behavior when the value of this flag is zero is backwards compatible (this is as the defaulted value of the flag).
- fix: make azure disk URI as case insensitive (#79020, @andyzhangx)
- Enable cadvisor ProcessMetrics collecting. (#79002, @jiayingz)
- Fixes a bug where `kubectl set config` hangs and uses 100% CPU on some invalid property names (#79000, @pswica)
- Fix a string comparison bug in IPVS graceful termination where UDP real servers are not deleted. (#78999, @andrewsykim)
- Reflector watchHandler Warning log â€˜The resourceVersion for the provided watch is too old.' is now logged as Info. (#78991, @sallyom)
- fix a bug that pods not be deleted from unmatched nodes by daemon controller (#78974, @DaiHao)
- NONE (#78821, @jhedev)
- Volume expansion is enabled in the default GCE storageclass (#78672, @msau42)
- kubeadm: use the service-cidr flag to pass the desired service CIDR to the kube-controller-manager via its service-cluster-ip-range flag. (#78625, @Arvinderpal)
- kubeadm: introduce deterministic ordering for the certificates generation in the phase command "kubeadm init phase certs" . (#78556, @neolit123)
- Add Pre-filter extension point to the scheduling framework. (#78005, @ahg-g)

- fix pod stuck issue due to corrupt mnt point in flexvol plugin, call Unmount if PathExists returns any error (#75234, @andyzhangx)

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on September 20, 2019 at 7:25 PM PST by Fix typo (#16447) (Page History)

Edit This Page

Installing Kubernetes with Minikube

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

Minikube Features

Minikube supports the following Kubernetes features:

- DNS
- NodePorts
- ConfigMaps and Secrets
- Dashboards

- Container Runtime: Docker, CRI-O, and containerd
- Enabling CNI (Container Network Interface)
- Ingress

Installation

See Installing Minikube.

Quickstart

This brief demo guides you on how to start, use, and delete Minikube locally. Follow the steps given below to start and explore Minikube.

1. Start Minikube and create a cluster:

   ```
   minikube start
   ```

   The output is similar to this:

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

   For more information on starting your cluster on a specific Kubernetes version, VM, or container runtime, see Starting a Cluster.

2. Now, you can interact with your cluster using kubectl. For more information, see Interacting with Your Cluster.

   Let's create a Kubernetes Deployment using an existing image named `echoserver`, which is a simple HTTP server and expose it on port 8080 using `--port`.

   ```
   kubectl run hello-minikube --image=k8s.gcr.io/echoserver:
   1.10 --port=8080
   ```

   The output is similar to this:

   ```
   deployment.apps/hello-minikube created
   ```

3. To access the `hello-minikube` Deployment, expose it as a Service:

   ```
   kubectl expose deployment hello-minikube --type=NodePort
   ```

   The option `--type=NodePort` specifies the type of the Service.

   The output is similar to this:

   ```
   service/hello-minikube exposed
   ```

4. The `hello-minikube` Pod is now launched but you have to wait until the Pod is up before accessing it via the exposed Service.

   Check if the Pod is up and running:

```
kubectl get pod
```

If the output shows the `STATUS` as `ContainerCreating`, the Pod is still being created:

```
NAME                                  READY
STATUS              RESTARTS   AGE
hello-minikube-3383150820-vctvh   0/1
ContainerCreating   0          3s
```

If the output shows the `STATUS` as `Running`, the Pod is now up and running:

```
NAME                                  READY     STATUS
RESTARTS   AGE
hello-minikube-3383150820-vctvh   1/1       Running
0          13s
```

5. Get the URL of the exposed Service to view the Service details:

```
minikube service hello-minikube --url
```

6. To view the details of your local cluster, copy and paste the URL you got as the output, on your browser.

The output is similar to this:

```
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-
```

If you no longer want the Service and cluster to run, you can delete them.

7. Delete the `hello-minikube` Service:

```
kubectl delete services hello-minikube
```

The output is similar to this:

```
service "hello-minikube" deleted
```

8. Delete the `hello-minikube` Deployment:

```
kubectl delete deployment hello-minikube
```

The output is similar to this:

```
deployment.extensions "hello-minikube" deleted
```

9. Stop the local Minikube cluster:

```
minikube stop
```

The output is similar to this:

```
Stopping "minikube"...
"minikube" stopped.
```

For more information, see Stopping a Cluster.

10. Delete the local Minikube cluster:

```
minikube delete
```

The output is similar to this:

```
Deleting "minikube" ...
The "minikube" cluster has been deleted.
```

For more information, see Deleting a cluster.

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.

Note:

If you are behind a web proxy, you need to pass this information to the `minikube start` command:

```
https_proxy=<my proxy> minikube start --docker-env http_p
roxy=<my proxy> --docker-env https_proxy=<my proxy> --
docker-env no_proxy=192.168.99.0/24
```

Unfortunately, setting the environment variables alone does not work.

> Minikube also creates a "minikube" context, and sets it to default in kubectl. To switch back to this context, run this command: `kubectl config use-context minikube`.

Specifying the Kubernetes version

You can specify the version of Kubernetes for Minikube to use by adding the `--kubernetes-version` string to the `minikube start` command. For example, to run version v1.16.0, you would run the following:

```
minikube start --kubernetes-version v1.16.0
```

Specifying the VM driver

You can change the VM driver by adding the `--vm-driver=<enter_driver_name>` flag to `minikube start`. For example the command would be.

```
minikube start --vm-driver=<driver_name>
```

Minikube supports the following drivers:

> **Note:** See DRIVERS for details on supported drivers and how to install plugins.

- virtualbox
- vmwarefusion
- kvm2 (driver installation)
- hyperkit (driver installation)
- hyperv (driver installation) Note that the IP below is dynamic and can change. It can be retrieved with `minikube ip`.
- vmware (driver installation) (VMware unified driver)
- none (Runs the Kubernetes components on the host and not in a VM. Using this driver requires Docker (docker install) and a Linux environment)

Starting a cluster on alternative container runtimes

You can start Minikube on the following container runtimes.

- containerd
- CRI-O

To use containerd as the container runtime, run:

```
minikube start \
    --network-plugin=cni \
    --enable-default-cni \
    --container-runtime=containerd \
    --bootstrapper=kubeadm
```

Or you can use the extended version:

```
minikube start \
    --network-plugin=cni \
    --enable-default-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
```

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

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

Or you can use the extended version:

```
minikube start \
    --network-plugin=cni \
    --enable-default-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
```

Use local images by re-using the Docker daemon

When using a single VM for Kubernetes, it's useful to reuse Minikube's built-in Docker daemon. Reusing the built-in daemon means you don't have to build a Docker registry on your host machine and push the image into it. Instead, you can build inside the same Docker daemon as Minikube, which speeds up local experiments.

> **Note:** Be sure to tag your Docker image with something other than latest and use that tag to pull the image. Because `:latest` is the default value, with a corresponding default image pull policy of `Always`, an image pull error (`ErrImagePull`) eventually results if you do not have the Docker image in the default Docker registry (usually DockerHub).

To work with the Docker daemon on your Mac/Linux host, use the `docker-env command` in your shell:

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

You can now use Docker at the command line of your host Mac/Linux machine to communicate with the Docker daemon inside the Minikube VM:

```
docker ps
```

> **Note:**
>
> 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 or directory
```

You can fix this by updating /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
```

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=true`.

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 its 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 multiple nodes will not work in Minikube.

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 instructions on how to add a new addon for Minikube, see the adding an addon guide.
- **MicroK8s**: Linux users wishing to avoid running a virtual machine may consider MicroK8s as an alternative.

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: ".

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on August 20, 2019 at 4:21 PM PST by Remove references to rkt (#15967) (Page History)

Edit This Page

Running Kubernetes on Multiple Clouds with IBM Cloud Private

- ◦ IBM Cloud Private and Terraform
  ◦ IBM Cloud Private on AWS
  ◦ IBM Cloud Private on Azure
  ◦ IBM Cloud Private with Red Hat OpenShift
  ◦ IBM Cloud Private on VirtualBox
  ◦ IBM Cloud Private on VMware

IBMÂ® Cloud Private is a turnkey cloud solution and an on-premises turnkey cloud solution. IBM Cloud Private delivers pure upstream Kubernetes with the typical management components that are required to run real enterprise workloads. These workloads include health management, log management, audit trails, and metering for tracking usage of workloads on the platform.

IBM Cloud Private is available in a community edition and a fully supported enterprise edition. The community edition is available at no charge from Docker Hub. The enterprise edition supports high availability topologies and includes commercial support from IBM for Kubernetes and the IBM Cloud Private management platform. If you want to try IBM Cloud Private, you can use either the hosted trial, the tutorial, or the self-guided demo. You can also try the free community edition. For details, see Get started with IBM Cloud Private.

For more information, explore the following resources:

- IBM Cloud Private
- Reference architecture for IBM Cloud Private
- IBM Cloud Private documentation

IBM Cloud Private and Terraform

The following modules are available where you can deploy IBM Cloud Private by using Terraform:

- AWS: Deploy IBM Cloud Private to AWS
- Azure: Deploy IBM Cloud Private to Azure
- IBM Cloud: Deploy IBM Cloud Private cluster to IBM Cloud
- OpenStack: Deploy IBM Cloud Private to OpenStack
- Terraform module: Deploy IBM Cloud Private on any supported infrastructure vendor
- VMware: Deploy IBM Cloud Private to VMware

IBM Cloud Private on AWS

You can deploy an IBM Cloud Private cluster on Amazon Web Services (AWS) by using either AWS CloudFormation or Terraform.

IBM Cloud Private has a Quick Start that automatically deploys IBM Cloud Private into a new virtual private cloud (VPC) on the AWS Cloud. A regular deployment takes about 60 minutes, and a high availability (HA) deployment takes about 75 minutes to complete. The Quick Start includes AWS CloudFormation templates and a deployment guide.

This Quick Start is for users who want to explore application modernization and want to accelerate meeting their digital transformation goals, by using IBM Cloud Private and IBM tooling. The Quick Start helps users rapidly deploy a high availability (HA), production-grade, IBM Cloud Private reference architecture on AWS. For all of the details and the deployment guide, see the IBM Cloud Private on AWS Quick Start.

IBM Cloud Private can also run on the AWS cloud platform by using Terraform. To deploy IBM Cloud Private in an AWS EC2 environment, see Installing IBM Cloud Private on AWS.

IBM Cloud Private on Azure

You can enable Microsoft Azure as a cloud provider for IBM Cloud Private deployment and take advantage of all the IBM Cloud Private features on the Azure public cloud. For more information, see IBM Cloud Private on Azure.

IBM Cloud Private with Red Hat OpenShift

You can deploy IBM certified software containers that are running on IBM Cloud Private onto Red Hat OpenShift.

Integration capabilities:

- Supports LinuxÂ® 64-bit platform in offline-only installation mode
- Single-master configuration
- Integrated IBM Cloud Private cluster management console and catalog
- Integrated core platform services, such as monitoring, metering, and logging
- IBM Cloud Private uses the OpenShift image registry

For more information see, IBM Cloud Private on OpenShift.

IBM Cloud Private on VirtualBox

To install IBM Cloud Private to a VirtualBox environment, see Installing IBM Cloud Private on VirtualBox.

IBM Cloud Private on VMware

You can install IBM Cloud Private on VMware with either Ubuntu or RHEL images. For details, see the following projects:

- Installing IBM Cloud Private with Ubuntu
- Installing IBM Cloud Private with Red Hat Enterprise

The IBM Cloud Private Hosted service automatically deploys IBM Cloud Private Hosted on your VMware vCenter Server instances. This service brings the power of microservices and containers to your VMware environment on IBM Cloud. With this service, you can extend the same familiar VMware and IBM Cloud Private operational model and tools from on-premises into the IBM Cloud.

For more information, see IBM Cloud Private Hosted service.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Container runtimes

FEATURE STATE: `Kubernetes v1.6` 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.

To run containers in Pods, Kubernetes uses a container runtime. Here are the installation instructions for various runtimes.

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

> **Caution:**
>
> A flaw was found in the way runc handled system file descriptors when running containers. A malicious container could use this flaw to overwrite contents of the runc binary and consequently run arbitrary commands on the container host system.
>
> Please refer to this link for more information about this issue cve-2019-5736 : runc vulnerability

Applicability

> **Note:** This document is written for users installing CRI onto Linux. For other operating systems, look for documentation specific to your platform.

You should execute all the commands in this guide as `root`. For example, prefix commands with `sudo`, or become `root` and run the commands as that user.

Cgroup drivers

When systemd is chosen as the init system for a Linux distribution, the init process generates and consumes a root control group (`cgroup`) and acts as a cgroup manager. Systemd has a tight integration with cgroups and will allocate cgroups per process. It's possible to configure your container runtime and the kubelet to use `cgroupfs`. Using `cgroupfs` alongside systemd means that there will then be two different cgroup managers.

Control groups are used to constrain resources that are allocated to processes. A single cgroup manager will simplify the view of what resources are being allocated and will by default have a more consistent view of the available and in-use resources. When we have two managers we end up with two views of those resources. We have seen cases in the field where nodes that are configured to use `cgroupfs` for the kubelet and Docker, and `systemd` for the rest of the processes running on the node becomes unstable under resource pressure.

Changing the settings such that your container runtime and kubelet use `systemd` as the cgroup driver stabilized the system. Please note the `native.cgroupdriver=systemd` option in the Docker setup below.

> **Caution:** Changing the cgroup driver of a Node that has joined a cluster is highly unrecommended. If the kubelet has created Pods using the semantics of one cgroup driver, changing the container runtime to another cgroup driver can cause errors when trying to re-create the PodSandbox for such existing Pods. Restarting the kubelet may not solve such errors. The recommendation is to drain the Node from its workloads, remove it from the cluster and re-join it.

Docker

On each of your machines, install Docker. Version 18.06.2 is recommended, but 1.11, 1.12, 1.13, 17.03 and 18.09 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 CE
## Set up the repository:
### Install packages to allow apt to use a repository over HTTPS
apt-get update && apt-get install apt-transport-https ca-
certificates curl software-properties-common

### Add Docker's official 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 CE.
apt-get update && apt-get install docker-ce=18.06.2~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 CE
```

```
## Set up the repository
### Install required packages.
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 CE.
yum update && yum install docker-ce-18.06.2.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-1.13
```

```
# Install prerequisites
yum-config-manager --add-repo=https://cbs.centos.org/repos/paas7-
crio-115-release/x86_64/os/

# 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

```
cat > /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF

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
```

Install containerd

- Ubuntu 16.04
- CentOS/RHEL 7.4+

```
# Install containerd
## Set up the repository
### Install packages to allow apt to use a repository over HTTPS
apt-get update && apt-get install -y apt-transport-https ca-
certificates curl software-properties-common

### Add Docker's official 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 containerd
apt-get update && apt-get install -y containerd.io

# Configure containerd
mkdir -p /etc/containerd
containerd config default > /etc/containerd/config.toml

# Restart containerd
systemctl restart containerd
```

```
# Install containerd
## Set up the repository
### Install required packages
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 containerd
yum update && yum install containerd.io

# Configure containerd
mkdir -p /etc/containerd
containerd config default > /etc/containerd/config.toml

# Restart containerd
systemctl restart containerd
```

systemd

To use the `systemd` cgroup driver, set `plugins.cri.systemd_cgroup = true` in `/etc/containerd/config.toml`. When using kubeadm, manually configure the cgroup driver for kubelet

Other CRI runtimes: frakti

Refer to the Frakti QuickStart guide for more information.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on August 15, 2019 at 8:54 AM PST by Issue with k8s.io/docs/setup/production-environment/container-runtimes/ (#15476) (Page History)

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

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: "cgroupfs"
```

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

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

kubeadm blocks when removing managed containers

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

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

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

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

Inspecting the logs for docker may also be useful:

```
journalctl -ul docker
```

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

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

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

f 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 be 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 because of "crypto/rsa: verification
error" while trying to verify candidate authority certificate
"kubernetes")
```

- 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.
- Unset the KUBECONFIG environment variable using:

```
  unset KUBECONFIG
```

Or set it to the default KUBECONFIG location:

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

- Another workaround is to overwrite the existing `kubeconfig` for the "admin" user:

```
  mv  $HOME/.kube $HOME/.kube.bak
  mkdir $HOME/.kube
  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/mysql: dial tcp 10.19.0.41:10250:
getsockopt: no route to host
```

- 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
```

`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.

etcd pods restart continually

If you encounter the following error:

```
rpc error: code = 2 desc = oci runtime error: exec failed:
container_linux.go:247: starting container process caused
"process_linux.go:110: decoding init error from pipe caused
\"read parent: connection reset by peer\""
```

this issue appears if you run CentOS 7 with Docker 1.13.1.84. This version of Docker can prevent the kubelet from executing into the etcd container.

To work around the issue, choose one of these options:

- Roll back to an earlier version of Docker, such as 1.13.1-75

  ```
  yum downgrade docker-1.13.1-75.git8633870.el7.centos.x86_64
  docker-client-1.13.1-75.git8633870.el7.centos.x86_64 docker-
  common-1.13.1-75.git8633870.el7.centos.x86_64
  ```

- Install one of the more recent recommended versions, such as 18.06:

```
sudo yum-config-manager --add-repo https://
download.docker.com/linux/centos/docker-ce.repo
yum install docker-ce-18.06.1.ce-3.el7.x86_64
```

Not possible to pass a comma separated list of values to arguments inside a `--component-extra-args` flag

`kubeadm init` flags such as `--component-extra-args` allow you to pass custom arguments to a control-plane component like the kube-apiserver. However, this mechanism is limited due to the underlying type used for parsing the values (`mapStringString`).

If you decide to pass an argument that supports multiple, comma-separated values such as `--apiserver-extra-args "enable-admission-plugins=LimitRanger,NamespaceExists"` this flag will fail with `flag: malformed pair, expect string=string`. This happens because the list of arguments for `--apiserver-extra-args` expects `key=value` pairs and in this case `NamespacesExists` is considered as a key that is missing a value.

Alternatively, you can try separating the `key=value` pairs like so: `--apiserver-extra-args "enable-admission-plugins=LimitRanger,enable-admission-plugins=NamespaceExists"` but this will result in the key `enable-admission-plugins` only having the value of `NamespaceExists`.

A known workaround is to use the kubeadm configuration file.

kube-proxy scheduled before node is initialized by cloud-controller-manager

In cloud provider scenarios, kube-proxy can end up being scheduled on new worker nodes before the cloud-controller-manager has initialized the node addresses. This causes kube-proxy to fail to pick up the node's IP address properly and has knock-on effects to the proxy function managing load balancers.

The following error can be seen in kube-proxy Pods:

```
server.go:610] Failed to retrieve node IP: host IP unknown;
known addresses: []
proxier.go:340] invalid nodeIP, initializing kube-proxy with
127.0.0.1 as nodeIP
```

A known solution is to patch the kube-proxy DaemonSet to allow scheduling it on control-plane nodes regardless of their conditions, keeping it off of other nodes until their initial guarding conditions abate:

```
kubectl -n kube-system patch ds kube-proxy -p='{ "spec":
{ "template": { "spec": { "tolerations": [ { "key":
"CriticalAddonsOnly", "operator": "Exists" }, { "effect":
"NoSchedule", "key": "node-role.kubernetes.io/
master" } ] } } } }'
```

The tracking issue for this problem is here.

The NodeRegistration.Taints field is omitted when marshalling kubeadm configuration

Note: This issue only applies to tools that marshal kubeadm types (e.g. to a YAML configuration file). It will be fixed in kubeadm API v1beta2.

By default, kubeadm applies the `role.kubernetes.io/master:NoSchedule` taint to control-plane nodes. If you prefer kubeadm to not taint the control-plane node, and set `InitConfiguration.NodeRegistration.Taints` to an empty slice, the field will be omitted when marshalling. When the field is omitted, kubeadm applies the default taint.

There are at least two workarounds:

1. Use the `role.kubernetes.io/master:PreferNoSchedule` taint instead of an empty slice. Pods will get scheduled on masters, unless other nodes have capacity.

2. Remove the taint after kubeadm init exits:

   ```
   kubectl taint nodes NODE_NAME role.kubernetes.io/
   master:NoSchedule-
   ```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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

- One or more machines running one of:
  - Ubuntu 16.04+
  - Debian 9+
  - CentOS 7
  - Red Hat Enterprise Linux (RHEL) 7
  - Fedora 25+
  - 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.

Ensure iptables tooling does not use the nftables backend

In Linux, nftables is available as a modern replacement for the kernel's iptables subsystem. The `iptables` tooling can act as a compatibility layer, behaving like iptables but actually configuring nftables. This nftables backend is not compatible with the current kubeadm packages: it causes duplicated firewall rules and breaks `kube-proxy`.

If your system's `iptables` tooling uses the nftables backend, you will need to switch the `iptables` tooling to â€˜legacy' mode to avoid these problems. This is the case on at least Debian 10 (Buster), Ubuntu 19.04, Fedora 29 and newer releases of these distributions by default. RHEL 8 does not support switching to legacy mode, and is therefore incompatible with current kubeadm packages.

- Debian or Ubuntu
- Fedora

```
update-alternatives --set iptables /usr/sbin/iptables-legacy
update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy
update-alternatives --set arptables /usr/sbin/arptables-legacy
update-alternatives --set ebtables /usr/sbin/ebtables-legacy
```

```
update-alternatives --set iptables /usr/sbin/iptables-legacy
```

Check required ports

Control-plane 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 control-plane 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.

Since v1.14.0, kubeadm will try to automatically detect the container runtime on Linux nodes by scanning through a list of well known domain sockets. The detectable runtimes and the socket paths, that are used, can be found in the table below.

Runtime	Domain Socket
Docker	/var/run/docker.sock
containerd	/run/containerd/containerd.sock
CRI-O	/var/run/crio/crio.sock

If both Docker and containerd are detected together, Docker takes precedence. This is needed, because Docker 18.09 ships with containerd and both are detectable. If any other two or more runtimes are detected, kubeadm will exit with an appropriate error message.

On non-Linux nodes the container runtime used by default is Docker.

If the container runtime of choice is Docker, it is used 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

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 plane 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.

For information about installing `kubectl`, see Install and set up kubectl.

> **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, see:

- Kubernetes version and version-skew policy

- Kubeadm-specific 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.com/yum/doc/rpm-package-key.gpg
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=kubernet
es

systemctl enable --now 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
  ```

- Make sure that the `br_netfilter` module is loaded before this step. This can be done by running `lsmod | grep br_netfilter`. To load it explicitly call `modprobe br_netfilter`.

Install CNI plugins (required for most pod network):

```
CNI_VERSION="v0.8.2"
mkdir -p /opt/cni/bin
curl -L "https://github.com/containernetworking/plugins/releases/download/${CNI_VERSION}/cni-plugins-linux-amd64-${CNI_VERSION}.tgz" | tar -C /opt/cni/bin -xz
```

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

```
CRICTL_VERSION="v1.16.0"
mkdir -p /opt/bin
curl -L "https://github.com/kubernetes-sigs/cri-tools/releases/download/${CRICTL_VERSION}/crictl-${CRICTL_VERSION}-linux-amd64.tar.gz" | tar -C /opt/bin -xz
```

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/${RELEASE}/bin/linux/amd64/{kubeadm,kubelet,kubectl}
chmod +x {kubeadm,kubelet,kubectl}

curl -sSL "https://raw.githubusercontent.com/kubernetes/kubernetes/${RELEASE}/build/debs/kubelet.service" | sed "s:/usr/bin:/opt/bin:g" > /etc/systemd/system/kubelet.service
mkdir -p /etc/systemd/system/kubelet.service.d
curl -sSL "https://raw.githubusercontent.com/kubernetes/kubernetes/${RELEASE}/build/debs/10-kubeadm.conf" | sed "s:/usr/bin:/opt/bin:g" > /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

Enable and start `kubelet`:

```
systemctl enable --now 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 control-plane 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
```

The automatic detection of cgroup driver for other container runtimes like CRI-O and containerd is work in progress.

Troubleshooting

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

What's next

- Using kubeadm to Create a Cluster

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page
Page last modified on September 16, 2019 at 1:40 PM PST by Add instructions for switching to iptables-legacy (#16271) (Page History)

Edit This Page

Creating a single control-plane 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

kubeadm's overall feature state is **GA**. Some sub-features, like 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
v1.13.x	December 2018	September 2019 Â
v1.14.x	March 2019	December 2019 Â
v1.15.x	June 2019	March 2020 Â
v1.16.x	September 2019	June 2020 Â

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 control-plane node
- Full network connectivity among all machines in the cluster. A public or private network is fine.

Objectives

- Install a single control-plane 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 control-plane, the kubelet runs normally.

Initializing your control-plane node

The control-plane node 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. (Recommended) If you have plans to upgrade this single control-plane kubeadm cluster to high availability you should specify the `--control-plane-endpoint` to set the shared endpoint for all control-plane nodes. Such an endpoint can be either a DNS name or an IP address of a load-balancer.
2. 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.
3. (Optional) Since version 1.14, kubeadm will try to detect the container runtime on Linux by using a list of well known domain socket paths. To use different container runtime or if there are more than one installed on the provisioned node, specify the `--cri-socket` argument to `kubeadm init`. See Installing runtime.
4. (Optional) Unless otherwise specified, kubeadm uses the network interface associated with the default gateway to set the advertise address for this particular control-plane node's API server. 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`
5. (Optional) Run `kubeadm config images pull` prior to `kubeadm init` to verify connectivity to gcr.io registries.

To initialize the control-plane node run:

```
kubeadm init <args>
```

Considerations about apiserver-advertise-address and ControlPlaneEndpoint

While `--apiserver-advertise-address` can be used to set the advertise address for this particular control-plane node's API server, `--control-plane-endpoint` can be used to set the shared endpoint for all control-plane nodes.

`--control-plane-endpoint` allows IP addresses but also DNS names that can map to IP addresses. Please contact your network administrator to evaluate possible solutions with respect to such mapping.

Here is an example mapping:

```
192.168.0.102 cluster-endpoint
```

Where `192.168.0.102` is the IP address of this node and `cluster-endpoint` is a custom DNS name that maps to this IP. This will allow you to pass `--control-plane-endpoint=cluster-endpoint` to `kubeadm init` and pass the same DNS name to `kubeadm join`. Later you can modify `cluster-endpoint` to point to the address of your load-balancer in an high availability scenario.

Turning a single control plane cluster created without `--control-plane-endpoint` into a highly available cluster is not supported by kubeadm.

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, make sure that your deployed DaemonSets have container image support for this 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
[preflight] Pulling images required for setting up a Kubernetes
cluster
[preflight] This might take a minute or two, depending on the
speed of your internet connection
[preflight] You can also perform this action in beforehand using
'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to
file "/var/lib/kubelet/kubeadm-flags.env"
```

```
[kubelet-start] Writing kubelet configuration to file "/var/lib/
kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names
[kubeadm-cp localhost] and IPs [10.138.0.4 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [kubeadm-
cp localhost] and IPs [10.138.0.4 127.0.0.1 ::1]
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubeadm-
cp kubernetes kubernetes.default kubernetes.default.svc
kubernetes.default.svc.cluster.local] and IPs [10.96.0.1
10.138.0.4]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-
controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/
kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the
control plane as static Pods from directory "/etc/kubernetes/
manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after
31.501735 seconds
[uploadconfig] storing the configuration used in ConfigMap
"kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-X.Y" in namespace
kube-system with the configuration for the kubelets in the
cluster
[patchnode] Uploading the CRI Socket information "/var/run/
dockershim.sock" to the Node API object "kubeadm-cp" as an
annotation
[mark-control-plane] Marking the node kubeadm-cp as control-
plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node kubeadm-cp as control-
plane by adding the taints [node-role.kubernetes.io/
master:NoSchedule]
[bootstrap-token] Using token: <token>
```

```
[bootstrap-token] Configuring bootstrap tokens, cluster-info
ConfigMap, RBAC Roles
[bootstraptoken] configured RBAC rules to allow Node Bootstrap
tokens to post CSRs in order for nodes to get long term
certificate credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover
controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate
rotation for all node client certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the
"kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a
regular user:

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

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

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

  kubeadm join <control-plane-host>:<control-plane-port> --token
<token> --discovery-token-ca-cert-hash sha256:<hash>
```

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 control-plane node 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.

Also, beware, that your Pod network must not overlap with any of the host networks as this can cause issues. If you find a collision between your network plugin's preferred Pod network and some of your host networks, you should think of a suitable CIDR replacement and use that during `kubeadm init` with `--pod-network-cidr` and as a replacement in your network plugin's YAML.

You can install a pod network add-on with the following command on the control-plane node or a node that has the kubeconfig credentials:

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

You can install only one pod network per cluster.

- Choose one...
- AWS VPC
- Calico
- Canal
- Cilium
- Contiv-VPP
- Flannel
- JuniperContrail/TungstenFabric
- Kube-router
- Romana
- Weave Net

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

AWS VPC CNI provides native AWS VPC networking to Kubernetes clusters.

For installation, please refer to the AWS VPC CNI setup guide.

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`, `arm64`, and `ppc64le` only.

```
kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/
calico.yaml
```

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/v3.8/manifests/
canal.yaml
```

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

For Cilium to work correctly, you must pass `--pod-network-cidr=10.217.0.0/16` to `kubeadm init`.

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

Note: If you are running kubeadm in a single node please untaint it so that etcd-operator pods can be scheduled in the control-plane node.

```
kubectl taint nodes <node-name> node-role.kubernetes.io/
master:NoSchedule-
```

To deploy Cilium you just need to run:

```
kubectl create -f https://raw.githubusercontent.com/cilium/
cilium/v1.5/examples/kubernetes/1.14/cilium.yaml
```

Once all Cilium pods are marked as `READY`, you start using your cluster.

```
kubectl get pods -n kube-system --selector=k8s-app=cilium
```

The output is similar to this:

```
NAME            READY   STATUS    RESTARTS   AGE
cilium-drxkl    1/1     Running   0          18m
```

Contiv-VPP employs a programmable CNF vSwitch based on FD.io VPP, offering feature-rich & high-performance cloud-native networking and services.

It implements k8s services and network policies in the user space (on VPP).

Please refer to this installation guide: Contiv-VPP Manual Installation

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.

Make sure that your firewall rules allow UDP ports 8285 and 8472 traffic for all hosts participating in the overlay network. see here .

Note that `flannel` works on `amd64`, `arm`, `arm64`, `ppc64le` and `s390x` under Linux. Windows (`amd64`) is claimed as supported in v0.11.0 but the usage is undocumented.

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/32a765fd19ba45b387fdc5e3812c41fff47cfd55/Documentation/kube-flannel.yml
```

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

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

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/romana-kubeadm.yml
```

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 | tr -d '\n')"
```

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, checkout our troubleshooting docs.

Control plane node isolation

By default, your cluster will not schedule pods on the control-plane node for security reasons. If you want to be able to schedule pods on the control-plane node, 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 control-plane 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> <control-plane-host>:<control-plane-
port> --discovery-token-ca-cert-hash sha256:<hash>
```

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

```
kubeadm token list
```

The output is similar to this:

```
TOKEN                     TTL  EXPIRES
USAGES           DESCRIPTION            EXTRA GROUPS
8ewj1p.9r9hcjoqgajrj4gi  23h  2018-06-12T02:51:28Z
```

```
authentication,  The default bootstrap  system:

signing          token generated by     bootstrappers:

   'kubeadm init'.        kubeadm:

                          default-node-token
```

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 control-plane 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 control-plane node:

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

The output is similar to this:

```
8cb2de97839780a412b93877f8507ad6c94f73add17d5d7058e91741c9d5ec78
```

> **Note:** To specify an IPv6 tuple for `<control-plane-host>:<control-plane-ip>`, IPv6 address must be enclosed in square brackets, for example: `[fd0 0::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 control-plane and response
  received.
* Kubelet informed of new secure connection details.

Run 'kubectl get nodes' on control-plane 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 control-plane node.

(Optional) Controlling your cluster from machines other than the control-plane node

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 control-plane node to your workstation like this:

```
scp root@<control-plane-host>:/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 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@<control-plane-host>:/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 control-plane node 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
```

The reset process does not reset or clean up iptables rules or IPVS tables. If you wish to reset iptables, you must do so manually:

```
iptables -F && iptables -t nat -F && iptables -t mangle -F && iptables -X
```

If you want to reset the IPVS tables, you must run the following command:

```
ipvsadm -C
```

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.

These resources provide more information on supported version skew between kubelets and the control plane, and other Kubernetes components:

- Kubernetes version and version-skew policy
- Kubeadm-specific installation guide

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.

Multiplatform container images for the control plane and addons are also supported since v1.12.

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

The cluster created here has a single control-plane node, with a single etcd database running on it. This means that if the control-plane node fails, your cluster may lose data and may need to be recreated from scratch.

Workarounds:

- Regularly back up etcd. The etcd data directory configured by kubeadm is at `/var/lib/etcd` on the control-plane node.
- Use multiple control-plane nodes by completing the HA setup instead.

Troubleshooting

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

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Customizing control plane configuration with kubeadm

FEATURE STATE: `Kubernetes 1.12` 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 kubeadm `ClusterConfiguration` object exposes the field `extraArgs` that can override the default flags passed to control plane components such as the APIServer, ControllerManager and Scheduler. The components are defined using the following fields:

- `apiServer`
- `controllerManager`
- `scheduler`

The `extraArgs` field consist of `key: value` pairs. To override a flag for a control plane component:

1. Add the appropriate fields to your configuration.
2. Add the flags to override to the field.
3. Run `kubeadm init` with `--config <YOUR CONFIG YAML>`.

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

> **Note:** You can generate a `ClusterConfiguration` object with default values by running `kubeadm config print init-defaults` and saving the output to a file of your choice.

- APIServer flags
- ControllerManager flags
- Scheduler flags

APIServer flags

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

Example usage:

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: v1.16.0
apiServer:
  extraArgs:
    advertise-address: 192.168.0.103
    anonymous-auth: "false"
    enable-admission-plugins: AlwaysPullImages,DefaultStorageClas
s
    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/v1beta2
kind: ClusterConfiguration
kubernetesVersion: v1.16.0
controllerManager:
  extraArgs:
    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/v1beta2
kind: ClusterConfiguration
kubernetesVersion: v1.16.0
scheduler:
  extraArgs:
    address: 0.0.0.0
    config: /home/johndoe/schedconfig.yaml
    kubeconfig: /home/johndoe/kubeconfig.yaml
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Options for Highly Available topology

This page explains the two options for configuring the topology of your highly available (HA) Kubernetes clusters.

You can set up an HA cluster:

- With stacked control plane nodes, where etcd nodes are colocated with control plane nodes
- With external etcd nodes, where etcd runs on separate nodes from the control plane

You should carefully consider the advantages and disadvantages of each topology before setting up an HA cluster.

Stacked etcd topology

A stacked HA cluster is a topology where the distributed data storage cluster provided by etcd is stacked on top of the cluster formed by the nodes managed by kubeadm that run control plane components.

Each control plane node runs an instance of the `kube-apiserver`, `kube-scheduler`, and `kube-controller-manager`. The `kube-apiserver` is exposed to worker nodes using a load balancer.

Each control plane node creates a local etcd member and this etcd member communicates only with the `kube-apiserver` of this node. The same applies to the local `kube-controller-manager` and `kube-scheduler` instances.

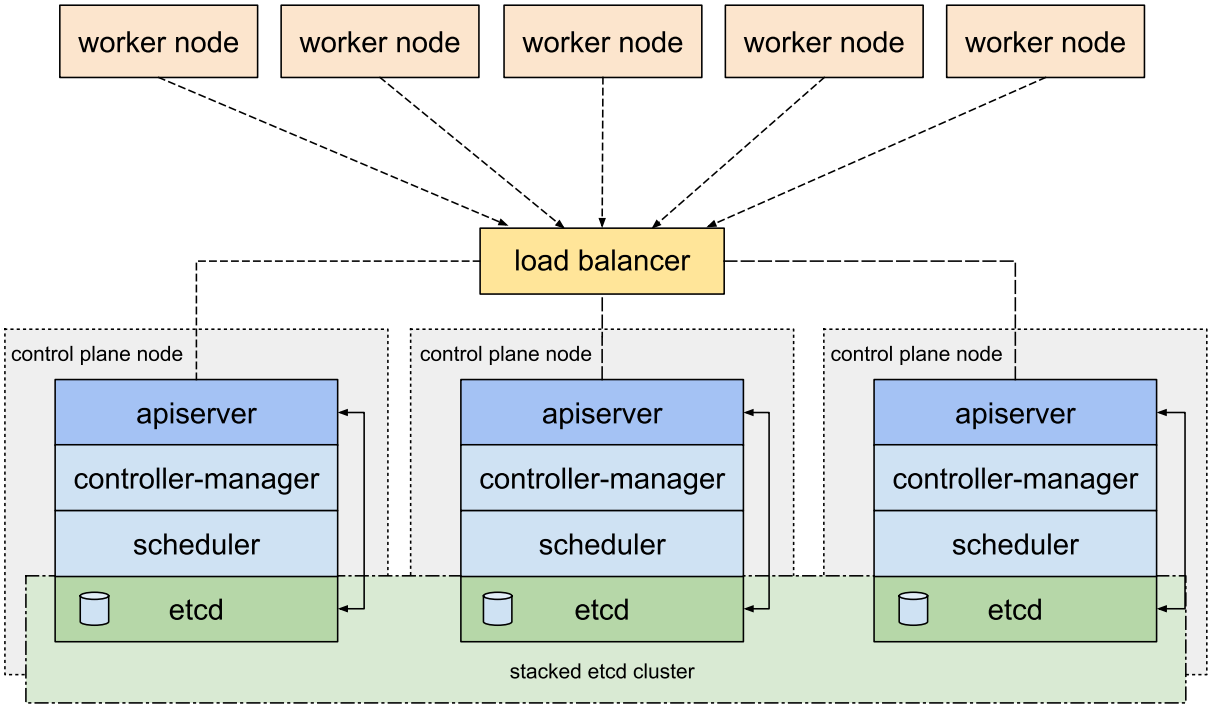
This topology couples the control planes and etcd members on the same nodes. It is simpler to set up than a cluster with external etcd nodes, and simpler to manage for replication.

However, a stacked cluster runs the risk of failed coupling. If one node goes down, both an etcd member and a control plane instance are lost, and redundancy is compromised. You can mitigate this risk by adding more control plane nodes.

You should therefore run a minimum of three stacked control plane nodes for an HA cluster.

This is the default topology in kubeadm. A local etcd member is created automatically on control plane nodes when using `kubeadm init` and `kubeadm join --control-plane`.

kubeadm HA topology - stacked etcd



External etcd topology

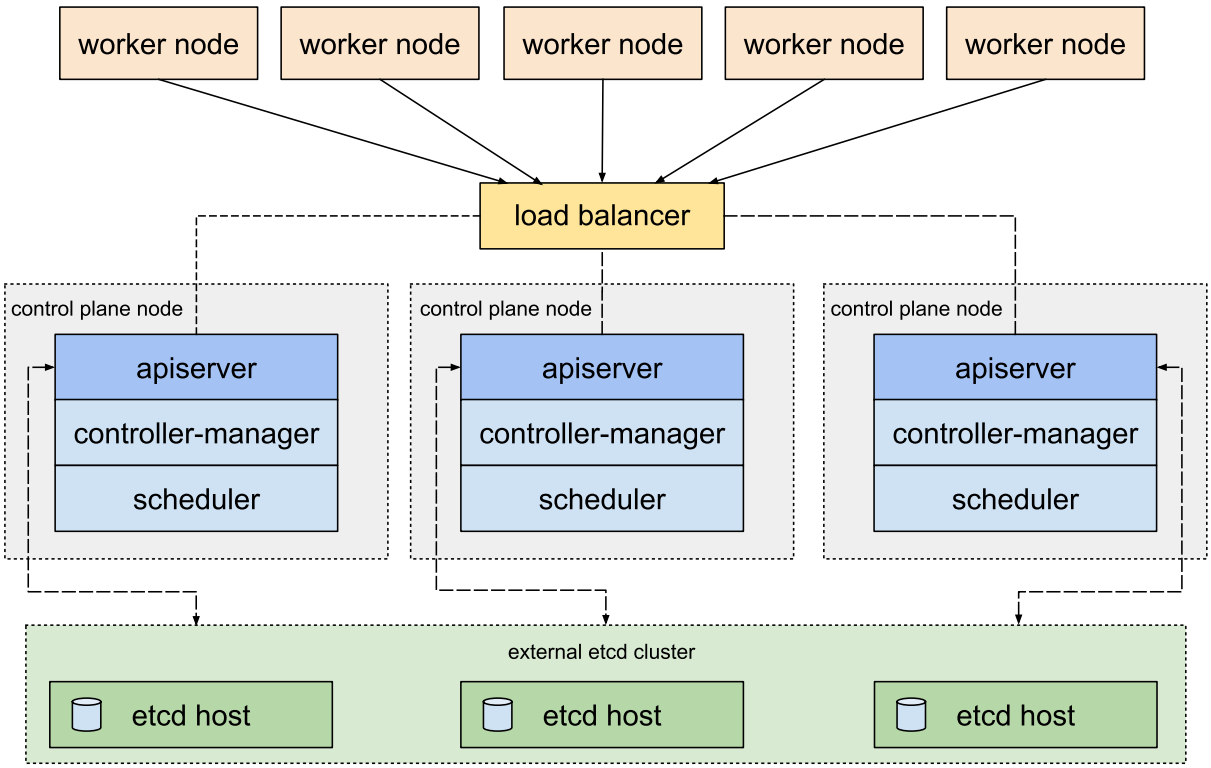
An HA cluster with external etcd is a topology where the distributed data storage cluster provided by etcd is external to the cluster formed by the nodes that run control plane components.

Like the stacked etcd topology, each control plane node in an external etcd topology runs an instance of the `kube-apiserver`, `kube-scheduler`, and `kube-controller-manager`. And the `kube-apiserver` is exposed to worker nodes using a load balancer. However, etcd members run on separate hosts, and each etcd host communicates with the `kube-apiserver` of each control plane node.

This topology decouples the control plane and etcd member. It therefore provides an HA setup where losing a control plane instance or an etcd member has less impact and does not affect the cluster redundancy as much as the stacked HA topology.

However, this topology requires twice the number of hosts as the stacked HA topology. A minimum of three hosts for control plane nodes and three hosts for etcd nodes are required for an HA cluster with this topology.

kubeadm HA topology - external etcd



What's next

- Set up a highly available cluster with kubeadm

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Creating Highly Available clusters with kubeadm

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

- With stacked control plane nodes. This approach requires less infrastructure. The 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.

Before proceeding, you should carefully consider which approach best meets the needs of your applications and environment. This comparison topic outlines the advantages and disadvantages of each.

If you encounter issues with setting up the HA cluster, please provide us with feedback in the kubeadm issue tracker.

See also The upgrade documentation.

> **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)
- sudo privileges on all machines
- SSH access from one device to all nodes in the system
- `kubeadm` and `kubelet` installed on all machines. `kubectl` is optional.

For the external etcd cluster only, you also need:

- Three additional machines for etcd members

First steps for both methods

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.
    - HAProxy can be used as a load balancer.
    - Make sure the address of the load balancer always matches the address of kubeadm's `ControlPlaneEndpoint`.
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 and etcd nodes

Steps for the first control plane node

1. Initialize the control plane:

    ```
    sudo kubeadm init --control-plane-endpoint "LOAD_BALANCER_DNS
    :LOAD_BALANCER_PORT" --upload-certs
    ```

    - You can use the `--kubernetes-version` flag to set the Kubernetes version to use. It is recommended that the versions of kubeadm, kubelet, kubectl and Kubernetes match.
    - The `--control-plane-endpoint` flag should be set to the address or DNS and port of the load balancer.
    - The `--upload-certs` flag is used to upload the certificates that should be shared across all the control-plane instances to the cluster. If instead, you prefer to copy

certs across control-plane nodes manually or using automation tools, please remove this flag and refer to Manual certificate distribution section bellow.

Note: The `kubeadm init` flags `--config` and `--certificate-key` cannot be mixed, therefore if you want to use the kubeadm configuration you must add the `certificateKey` field in the appropriate config locations (under `InitConfiguration` and `JoinConfiguration: controlPlane`).

Note: Some CNI network plugins like Calico require a CIDR such as `192.168.0.0/16` and some like Weave do not. See the CNI network documentation. To add a pod CIDR pass the flag `--pod-network-cidr`, or if you are using a kubeadm configuration file set the `podSubnet` field under the `networking` object of `ClusterConfiguration`.

```
After the command completes you should see something like so:

```sh
...
You can now join any number of control-plane node by running the
following command on each as a root:
  kubeadm join 192.168.0.200:6443 --token
9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26
aec866 --control-plane --certificate-key
f8902e114ef118304e561c3ecd4d0b543adc226b7a07f675f56564185ffe0c07

Please note that the certificate-key gives access to cluster
sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If
necessary, you can use kubeadm init phase upload-certs to reload
certs afterward.

Then you can join any number of worker nodes by running the
following on each as root:
  kubeadm join 192.168.0.200:6443 --token
9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26
aec866
```


- Copy this output to a text file. You will need it later to
join control plane and worker nodes to the cluster.
- When `--upload-certs` is used with `kubeadm init`, the
certificates of the primary control plane
are encrypted and uploaded in the `kubeadm-certs` Secret.
- To re-upload the certificates and generate a new decryption
key, use the following command on a control plane
node that is already joined to the cluster:

  ```sh
  sudo kubeadm init phase upload-certs --upload-certs
  ```


- You can also specify a custom `--certificate-key` during
```

```
`init` that can later be used by `join`.
To generate such a key you can use the following command:

  ```sh
  kubeadm alpha certs certificate-key
  ```
```

Note: The `kubeadm-certs` Secret and decryption key expire after two hours.

Caution: As stated in the command output, the certificate key gives access to cluster sensitive data, keep it secret!

1. Apply the CNI plugin of your choice: Follow these instructions to install the CNI provider. Make sure the configuration corresponds to the Pod CIDR specified in the kubeadm configuration file if applicable.

   In this example we are using Weave Net:

   ```
   kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
   ```

2. Type the following and watch the pods of the control plane components get started:

   ```
   kubectl get pod -n kube-system -w
   ```

Steps for the rest of the control plane nodes

Note: Since kubeadm version 1.15 you can join multiple control-plane nodes in parallel. Prior to this version, you must join new control plane nodes sequentially, only after the first node has finished initializing.

For each additional control plane node you should:

1. Execute the join command that was previously given to you by the `kubeadm init` output on the first node. It should look something like this:

   ```
   sudo kubeadm join 192.168.0.200:6443 --token 9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26aec866 --control-plane --certificate-key f8902e114ef118304e561c3ecd4d0b543adc226b7a07f675f56564185ffe0c07
   ```

   - The `--control-plane` flag tells `kubeadm join` to create a new control plane.
   - The `--certificate-key ...` will cause the control plane certificates to be downloaded from the `kubeadm-certs` Secret in the cluster and be decrypted using the given key.

External etcd nodes

Setting up a cluster with external etcd nodes is similar to the procedure used for stacked etcd with the exception that you should setup etcd first, and you should pass the etcd information in the kubeadm config file.

Set up the etcd cluster

1. Follow these instructions to set up the etcd cluster.
2. Setup SSH as described here.
3. Copy the following files from any etcd node in the cluster to the first control plane node:

```
export CONTROL_PLANE="ubuntu@10.0.0.7"
scp /etc/kubernetes/pki/etcd/ca.crt "${CONTROL_PLANE}":
scp /etc/kubernetes/pki/apiserver-etcd-client.crt "${CONTROL_
PLANE}":
scp /etc/kubernetes/pki/apiserver-etcd-client.key "${CONTROL_
PLANE}":
```

   - Replace the value of `CONTROL_PLANE` with the `user@host` of the first control plane machine.

Set up the first control plane node

1. Create a file called `kubeadm-config.yaml` with the following contents:

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: stable
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
```

   Note: The difference between stacked etcd and external etcd here is that the external etcd setup requires a configuration file with the etcd endpoints under the `external` object for `etcd`. In the case of the stacked etcd topology this is managed automatically.

```
-  Replace the following variables in the config template with
the appropriate values for your cluster:

    - `LOAD_BALANCER_DNS`
    - `LOAD_BALANCER_PORT`
    - `ETCD_0_IP`
    - `ETCD_1_IP`
    - `ETCD_2_IP`
```

The following steps are similar to the stacked etcd setup:

1. Run `sudo kubeadm init --config kubeadm-config.yaml --upload-certs` on this node.
2. Write the output join commands that are returned to a text file for later use.
3. Apply the CNI plugin of your choice. The given example is for Weave Net:

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

Steps for the rest of the control plane nodes

The steps are the same as for the stacked etcd setup:

- Make sure the first control plane node is fully initialized.
- Join each control plane node with the join command you saved to a text file. It's recommended to join the control plane nodes one at a time.
- Don't forget that the decryption key from `--certificate-key` expires after two hours, by default.

Common tasks after bootstrapping control plane

Install workers

Worker nodes can be joined to the cluster with the command you stored previously as the output from the `kubeadm init` command:

```
sudo kubeadm join 192.168.0.200:6443 --token 9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26aec866
```

Manual certificate distribution

If you choose to not use `kubeadm init` with the `--upload-certs` flag this means that you are going to have to manually copy the certificates from the primary control plane node to the joining control plane nodes.

There are many ways to do this. In the following example we are using `ssh` and `scp`:

SSH is required if you want to control all nodes from a single machine.

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
      ```

4. After configuring SSH on all the nodes you should run the following script on the first control plane node after running `kubeadm init`. This script will copy the certificates from the first control plane node to the other control plane nodes:

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

   Caution: Copy only the certificates in the above list. kubeadm will take care of generating the rest of the certificates with the required SANs for the joining control-plane instances. If you copy all the certificates by mistake, the creation of additional nodes could fail due to a lack of required SANs.

1. Then on each joining control plane node you have to run the following script before running `kubeadm join`. This script will move the previously copied certificates from the home directory to `/etc/kubernetes/pki`:

   ```
   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
   ```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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

- 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.

   Since etcd was created first, you must override the service priority by creating a new unit file that has higher precedence than the kubeadm-provided kubelet unit file.

   ```
   cat << EOF > /etc/systemd/system/kubelet.service.d/20-etcd-service-manager.conf
   [Service]
   ExecStart=
   #  Replace "systemd" with the cgroup driver of your container runtime. The default value in the kubelet is "cgroupfs".
   ExecStart=/usr/bin/kubelet --address=127.0.0.1 --pod-manifest-path=/etc/kubernetes/manifests --cgroup-driver=systemd
   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/v1beta2"
   kind: ClusterConfiguration
   etcd:
       local:
           serverCertSANs:
           - "${HOST}"
   ```

```
        peerCertSANs:
        - "${HOST}"
        extraArgs:
            initial-cluster: ${NAMES[0]}=https://$
{ETCDHOSTS[0]}:2380,${NAMES[1]}=https://${ETCDHOSTS[1]}:
2380,${NAMES[2]}=https://${ETCDHOSTS[2]}:2380
            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 init 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 init phase certs etcd-server --config=/tmp/${HOST2}/
   kubeadmcfg.yaml
   kubeadm init phase certs etcd-peer --config=/tmp/${HOST2}/
   kubeadmcfg.yaml
   kubeadm init phase certs etcd-healthcheck-client --config=/
   tmp/${HOST2}/kubeadmcfg.yaml
   kubeadm init 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 init phase certs etcd-server --config=/tmp/${HOST1}/
   kubeadmcfg.yaml
   kubeadm init phase certs etcd-peer --config=/tmp/${HOST1}/
   kubeadmcfg.yaml
   kubeadm init phase certs etcd-healthcheck-client --config=/
   tmp/${HOST1}/kubeadmcfg.yaml
   kubeadm init 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 init phase certs etcd-server --config=/tmp/${HOST0}/
kubeadmcfg.yaml
kubeadm init phase certs etcd-peer --config=/tmp/${HOST0}/
kubeadmcfg.yaml
kubeadm init phase certs etcd-healthcheck-client --config=/
tmp/${HOST0}/kubeadmcfg.yaml
kubeadm init 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}
â””â”€â”€ kubeadmcfg.yaml
---
/etc/kubernetes/pki
â”œâ”€â”€ apiserver-etcd-client.crt
â”œâ”€â”€ apiserver-etcd-client.key
â””â”€â”€ etcd
    â”œâ”€â”€ ca.crt
    â”œâ”€â”€ ca.key
    â”œâ”€â”€ healthcheck-client.crt
    â”œâ”€â”€ healthcheck-client.key
    â”œâ”€â”€ peer.crt
    â”œâ”€â”€ peer.key
    â”œâ”€â”€ server.crt
    â””â”€â”€ server.key
```

   On `$HOST1`:

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

On `$HOST2`

```
$HOME
â””â”€â”€ kubeadmcfg.yaml
---
/etc/kubernetes/pki
â”œâ”€â”€ apiserver-etcd-client.crt
â”œâ”€â”€ apiserver-etcd-client.key
â””â”€â”€ etcd
    â”œâ”€â”€ ca.crt
    â”œâ”€â”€ healthcheck-client.crt
    â”œâ”€â”€ healthcheck-client.key
    â”œâ”€â”€ peer.crt
    â”œâ”€â”€ peer.key
    â”œâ”€â”€ server.crt
    â””â”€â”€ 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 init phase etcd local --config=/tmp/${HO
ST0}/kubeadmcfg.yaml
root@HOST1 $ kubeadm init phase etcd local --config=/home/
ubuntu/kubeadmcfg.yaml
root@HOST2 $ kubeadm init 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:${ETCD
_TAG} 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
```

- Set `${ETCD_TAG}` to the version tag of your etcd image. For example `v3.2.24`.
- Set `${HOST0}`to the IP address of the host you are testing.

What's next

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

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on August 01, 2019 at 11:48 AM PST by Adding --cgroup-driver flag to etcd documentation (#15595) (Page History)

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

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 init-defaults --component-configs 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 to 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

kubeadm ships with configuration for how systemd should run the kubelet. Note that the kubeadm CLI command never touches this drop-in file.

This 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. It augments the basic `kubelet.service` for RPM (resp. `kubelet.service` for DEB)):

```
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/
kubernetes/bootstrap-kubelet.conf
--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" generate
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. Preferably,
#the user should use the .NodeRegistration.KubeletExtraArgs
object in the configuration files instead.
# 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_ARGS $KUBELET_EXTRA_ARGS
```

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/sysconfig/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 for the kubelet.
`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 the cri-tools git repository.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Configuring your kubernetes cluster to self-host the control plane

Self-hosting the Kubernetes control plane

kubeadm allows you to experimentally create a *self-hosted* Kubernetes control plane. This means that key components such as the API server, controller manager, and scheduler run as DaemonSet pods configured via the Kubernetes API instead of static pods configured in the kubelet via static files.

To create a self-hosted cluster see the kubeadm alpha selfhosting pivot command.

Caveats

> **Caution:** This feature pivots your cluster into an unsupported state, rendering kubeadm unable to manage you cluster any longer. This includes `kubeadm upgrade`.

1. Self-hosting in 1.8 and later has some important limitations. In particular, a self-hosted cluster *cannot recover from a reboot of the control-plane node* without manual intervention.

2. By default, self-hosted control plane Pods rely on credentials loaded from `hostPath` volumes. Except for initial creation, these credentials are not managed by kubeadm.

3. The self-hosted portion of the control plane does not include etcd, which still runs as a static Pod.

Process

The self-hosting bootstrap process is documented in the kubeadm design document.

In summary, `kubeadm alpha selfhosting` works as follows:

1. Waits for this bootstrap static control plane to be running and healthy. This is identical to the `kubeadm init` process without self-hosting.

2. Uses the static control plane Pod manifests to construct a set of DaemonSet manifests that will run the self-hosted control plane. It also modifies these manifests where necessary, for example adding new volumes for secrets.

3. Creates DaemonSets in the `kube-system` namespace and waits for the resulting Pods to be running.

4. Once self-hosted Pods are operational, their associated static Pods are deleted and kubeadm moves on to install the next component. This triggers kubelet to stop those static Pods.

5. When the original static control plane stops, the new self-hosted control plane is able to bind to listening ports and become active.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on September 05, 2019 at 7:29 PM PST by kubeadm: update setup docs for 1.16 (#16016) (Page History)

Edit This Page

Installing Kubernetes with 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 Kubernetes 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, CoreOS, RancherOS 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. Both are freely available via the RackN UX or using the upload from catalog feature of the DRPCLI (shown below).

```
drpcli plugin_providers upload certs from catalog:certs-stable
drpcli contents upload catalog:krib-stable
```

(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

Feedback

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 05, 2019 at 7:28 AM PST by fix broken link in krib.md (#15295) (Page History)

Edit This Page

Installing Kubernetes 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

(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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Installing Kubernetes 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

(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/inventory.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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page
Page last modified on June 12, 2019 at 5:27 PM PST by Restructure the left navigation pane of setup (#14826) (Page History)

Edit This Page

Running Kubernetes on AWS EC2

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

- 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.

- KubeOne is an open source cluster lifecycle management tool that creates, upgrades and manages Kubernetes Highly-Available clusters.

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 â€˜Desired' and â€˜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, Community
AWS	KubeOne	Ubuntu, CoreOS, CentOS	canal, weavenet	docs	100%	Commercial, Community

Further reading

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

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Page last modified on June 12, 2019 at 5:27 PM PST by Restructure the left navigation pane of setup (#14826) (Page History)

Edit This Page

Running Kubernetes on Alibaba Cloud

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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 12, 2019 at 5:27 PM PST by Restructure the left navigation pane of setup (#14826) (Page History)

Edit This Page

Running Kubernetes on 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: AKS-Engine

The core of the Azure Kubernetes Service is **open source** and available on GitHub for the community to use and contribute to: **AKS-Engine**. The legacy ACS-Engine codebase has been deprecated in favor of AKS-engine.

AKS-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 AKS-Engine may even become features of the Azure Kubernetes Service.

The input to AKS-Engine is an apimodel JSON file describing the Kubernetes cluster. It is similar to the Azure Resource Manager (ARM) template syntax used to deploy a cluster directly with the Azure Kubernetes Service. The resulting output is an ARM template that can be checked into source control and used to deploy Kubernetes clusters to Azure.

You can get started by following the **AKS-Engine Kubernetes Tutorial**.

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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Edit This Page

Running Kubernetes on CenturyLink Cloud

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 --datacenter=VA1
```

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 --datacenter=VA1 --etcd_separate_cluster=yes
```

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 --datacenter=VA1 -mem=6 -cpu=4
```

Cluster Features and Architecture

We configure the Kubernetes cluster with the following features:

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

We use the following to create the Kubernetes cluster:

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

Optional add-ons

- Logging: We offer an integrated centralized logging ELK platform so that all Kubernetes and docker logs get sent to the ELK stack. To install the ELK stack and configure

Kubernetes to send logs to it, follow the log aggregation documentation. Note: We don't install this by default as the footprint isn't trivial.

Cluster management

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

The script requires that the environment variable `CLC_CLUSTER_NAME` be defined. `install_kubectl.sh` also writes a configuration file which will embed the necessary authentication certificates for the particular cluster. The configuration file is written to the `${CLC_CLUSTER_HOME}/kube` directory

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

Accessing the cluster programmatically

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

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

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

But please note, this *does not* work out of the box with the `curl` binary distributed with 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/services/kubernetes-dashboard/proxy`.

Note on Authentication to the UIs:

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

Configuration files

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

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

`kubectl` usage examples

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

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

```
kubectl get nodes
kubectl get --all-namespaces 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/kubernetes-dashboard/proxy/` 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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Running Kubernetes on Google Compute Engine

The example below creates a Kubernetes cluster with 3 worker node Virtual Machines and a master Virtual Machine (i.e. 4 VMs in your cluster). This cluster is set up and controlled from your workstation (or wherever you find convenient).

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)         AGE
default       kubernetes     ClusterIP         10.0.0.1
<none>            443/TCP         1d
kube-system   kube-dns       ClusterIP         10.0.0.2
<none>            53/TCP,53/UDP   1d
kube-system   kube-ui        ClusterIP         10.0.0.3
```

```
<none>            80/TCP          1d
...
```

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

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

command.

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

```
NAMESPACE     NAME
READY     STATUS    RESTARTS   AGE
kube-system   coredns-5f4fbb68df-mc8z8                       1/
1       Running   0          15m
kube-system   fluentd-cloud-logging-kubernetes-minion-63uo   1/
1       Running   0          14m
kube-system   fluentd-cloud-logging-kubernetes-minion-c1n9   1/
1       Running   0          14m
kube-system   fluentd-cloud-logging-kubernetes-minion-c4og   1/
1       Running   0          14m
kube-system   fluentd-cloud-logging-kubernetes-minion-ngua   1/
1       Running   0          14m
kube-system   kube-ui-v1-curt1                               1/
1       Running   0          15m
kube-system   monitoring-heapster-v5-ex4u3                   1/
1       Running   1          15m
kube-system   monitoring-influx-grafana-v1-piled             2/
2       Running   0          15m
```

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 3 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

Further reading

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

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Running Kubernetes on Tencent Kubernetes Engine

Tencent Kubernetes Engine

Tencent Cloud Tencent Kubernetes Engine (TKE) provides native Kubernetes container management services. You can deploy and manage a Kubernetes cluster with TKE in just a few steps. For detailed directions, see Deploy Tencent Kubernetes Engine.

TKE is a Certified Kubernetes product.It is fully compatible with the native Kubernetes API.

Custom Deployment

The core of Tencent Kubernetes Engine is open source and available on GitHub.

When using TKE to create a Kubernetes cluster, you can choose managed mode or independent deployment mode. In addition, you can customize the deployment as needed; for example, you can choose an existing Cloud Virtual Machine instance for cluster creation or enable Kube-proxy in IPVS mode.

What's Next

To learn more, see the TKE documentation.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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

```
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)

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

---

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.

Screencast

Support Level

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

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page
Page last modified on June 12, 2019 at 5:27 PM PST by Restructure the left navigation pane of setup (#14826) (Page History)

Edit This Page

Guide for scheduling Windows containers in Kubernetes

Windows applications constitute a large portion of the services and applications that run in many organizations. This guide walks you through the steps to configure and deploy a Windows container in Kubernetes.

Objectives

- Configure an example deployment to run Windows containers on the Windows node
- (Optional) Configure an Active Directory Identity for your Pod using Group Managed Service Accounts (GMSA)

Before you begin

- Create a Kubernetes cluster that includes a master and a worker node running Windows Server
- It is important to note that creating and deploying services and workloads on Kubernetes behaves in much the same way for Linux and Windows containers. Kubectl commands to interface with the cluster are identical. The example in the section below is provided simply to jumpstart your experience with Windows containers.

Getting Started: Deploying a Windows container

To deploy a Windows container on Kubernetes, you must first create an example application. The example YAML file below creates a simple webserver application. Create a service spec named `win-webserver.yaml` with the contents below:

```
    apiVersion: v1
    kind: Service
    metadata:
      name: win-webserver
      labels:
        app: win-webserver
    spec:
      ports:
      # the port that this service should serve on
      - port: 80
        targetPort: 80
      selector:
        app: win-webserver
      type: NodePort
    ---
    apiVersion: extensions/v1beta1
    kind: Deployment
    metadata:
      labels:
        app: win-webserver
      name: win-webserver
    spec:
      replicas: 2
      template:
        metadata:
          labels:
            app: win-webserver
          name: win-webserver
        spec:
          containers:
          - name: windowswebserver
            image: mcr.microsoft.com/windows/servercore:ltsc2019
            command:
            - powershell.exe
            - -command
            - "<#code used from https://gist.github.com/
wagnerandrade/5424431#> ; $$listener = New-Object
System.Net.HttpListener ; $$listener.Prefixes.Add('http://*:
80/') ; $$listener.Start() ; $$callerCounts = @{} ; Write-
Host('Listening at http://*:80/') ; while ($
$listener.IsListening) { ;$$context = $$listener.GetContext() ;$
$requestUrl = $$context.Request.Url ;$$clientIP = $
$context.Request.RemoteEndPoint.Address ;$$response = $
$context.Response ;Write-Host '' ;Write-Host('> {0}' -f $
$requestUrl) ;  ;$$count = 1 ;$$k=$$callerCounts.Get_Item($
$clientIP) ;if ($$k -ne $$null) { $$count += $$k } ;$
```

```
$callerCounts.Set_Item($$clientIP, $$count) ;$$ip=(Get-
NetAdapter | Get-NetIpAddress); $
$header='<html><body><H1>Windows Container Web Server</H1>' ;$
$callerCountsString='' ;$$callerCounts.Keys | % { $
$callerCountsString+='<p>IP {0} callerCount {1} ' -f $
$ip[1].IPAddress,$$callerCounts.Item($$_) } ;$$footer='</body></
html>' ;$$content='{0}{1}{2}' -f $$header,$$callerCountsString,$
$footer ;Write-Output $$content ;$$buffer =
[System.Text.Encoding]::UTF8.GetBytes($$content) ;$
$response.ContentLength64 = $$buffer.Length ;$
$response.OutputStream.Write($$buffer, 0, $$buffer.Length) ;$
$response.Close() ;$$responseStatus = $
$response.StatusCode ;Write-Host('< {0}' -f $
$responseStatus)  } ; "
          nodeSelector:
            beta.kubernetes.io/os: windows
```

Note: Port mapping is also supported, but for simplicity in this example the container port 80 is exposed directly to the service.

1. Check that all nodes are healthy:

   ```
   kubectl get nodes
   ```

2. Deploy the service and watch for pod updates:

   ```
   kubectl apply -f win-webserver.yaml
   kubectl get pods -o wide -w
   ```

   When the service is deployed correctly both Pods are marked as Ready. To exit the watch command, press Ctrl+C.

3. Check that the deployment succeeded. To verify:

   - Two containers per pod on the Windows node, use `docker ps`
   - Two pods listed from the Linux master, use `kubectl get pods`
   - Node-to-pod communication across the network, `curl` port 80 of your pod IPs from the Linux master to check for a web server response
   - Pod-to-pod communication, ping between pods (and across hosts, if you have more than one Windows node) using docker exec or kubectl exec
   - Service-to-pod communication, `curl` the virtual service IP (seen under `kubectl get services`) from the Linux master and from individual pods
   - Service discovery, `curl` the service name with the Kubernetes default DNS suffix
   - Inbound connectivity, `curl` the NodePort from the Linux master or machines outside of the cluster
   - Outbound connectivity, `curl` external IPs from inside the pod using kubectl exec

   Note: Windows container hosts are not able to access the IP of services scheduled on them due to current platform limitations of the Windows networking stack. Only Windows pods are able to access service IPs.

Using configurable Container usernames

Starting with Kubernetes v1.16, Windows containers can be configured to run their entrypoints and processes with different usernames than the image defaults. The way this is achieved is a bit different from the way it is done for Linux containers. Learn more about it here.

Managing Workload Identity with Group Managed Service Accounts

Starting with Kubernetes v1.14, Windows container workloads can be configured to use Group Managed Service Accounts (GMSA). Group Managed Service Accounts are a specific type of Active Directory account that provides automatic password management, simplified service principal name (SPN) management, and the ability to delegate the management to other administrators across multiple servers. Containers configured with a GMSA can access external Active Directory Domain resources while carrying the identity configured with the GMSA. Learn more about configuring and using GMSA for Windows containers here.

Taints and Tolerations

Users today need to use some combination of taints and node selectors in order to keep Linux and Windows workloads on their respective OS-specific nodes. This likely imposes a burden only on Windows users. The recommended approach is outlined below, with one of its main goals being that this approach should not break compatibility for existing Linux workloads.

Ensuring OS-specific workloads land on the appropriate container host

Users can ensure Windows containers can be scheduled on the appropriate host using Taints and Tolerations. All Kubernetes nodes today have the following default labels:

- beta.kubernetes.io/os = [windows|linux]
- beta.kubernetes.io/arch = [amd64|arm64|â€¦]

If a Pod specification does not specify a nodeSelector like `"beta.kubernetes.io/os": windows`, it is possible the Pod can be scheduled on any host, Windows or Linux. This can be problematic since a Windows container can only run on Windows and a Linux container can only run on Linux. The best practice is to use a nodeSelector.

However, we understand that in many cases users have a pre-existing large number of deployments for Linux containers, as well as an ecosystem of off-the-shelf configurations, such as community Helm charts, and programmatic Pod generation cases, such as with Operators. In those situations, you may be hesitant to make the configuration change to add nodeSelectors. The alternative is to use Taints. Because the kubelet can set Taints during registration, it could easily be modified to automatically add a taint when running on Windows only.

For example: `--register-with-taints='os=Win1809:NoSchedule'`

By adding a taint to all Windows nodes, nothing will be scheduled on them (that includes existing Linux Pods). In order for a Windows Pod to be scheduled on a Windows node, it would need both the nodeSelector to choose Windows, and the appropriate matching toleration.

```
nodeSelector:
    "beta.kubernetes.io/os": windows
tolerations:
    - key: "os"
      operator: "Equal"
      value: "Win1809"
      effect: "NoSchedule"
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page
Page last modified on August 19, 2019 at 4:37 PM PST by Windows: Adds RunAsUserName related documentation (#15378) (Page History)

Edit This Page

Intro to Windows support in Kubernetes

Windows applications constitute a large portion of the services and applications that run in many organizations. Windows containers provide a modern way to encapsulate processes and package dependencies, making it easier to use DevOps practices and follow cloud native patterns for Windows applications. Kubernetes has become the defacto standard container orchestrator, and the release of Kubernetes 1.14 includes production support for scheduling Windows containers on Windows nodes in a Kubernetes cluster, enabling a vast ecosystem of Windows applications to leverage the power of Kubernetes. Organizations with investments in Windows-based applications and Linux-based applications don't have to look for separate orchestrators to manage their workloads, leading to increased operational efficiencies across their deployments, regardless of operating system.

Windows containers in Kubernetes

To enable the orchestration of Windows containers in Kubernetes, simply include Windows nodes in your existing Linux cluster. Scheduling Windows containers in Pods on Kubernetes is as simple and easy as scheduling Linux-based containers.

In order to run Windows containers, your Kubernetes cluster must include multiple operating systems, with control plane nodes running Linux and workers running either Windows or Linux depending on your workload needs. Windows Server 2019 is the only Windows operating system supported, enabling Kubernetes Node on Windows (including kubelet, container runtime, and kube-proxy). For a detailed explanation of Windows distribution channels see the Microsoft documentation.

> **Note:** The Kubernetes control plane, including the master components, continues to run on Linux. There are no plans to have a Windows-only Kubernetes cluster.

> **Note:** In this document, when we talk about Windows containers we mean Windows containers with process isolation. Windows containers with Hyper-V isolation is planned for a future release.

Supported Functionality and Limitations

Supported Functionality

Compute

From an API and kubectl perspective, Windows containers behave in much the same way as Linux-based containers. However, there are some notable differences in key functionality which are outlined in the limitation section.

Let's start with the operating system version. Refer to the following table for Windows operating system support in Kubernetes. A single heterogeneous Kubernetes cluster can have both Windows and Linux worker nodes. Windows containers have to be scheduled on Windows nodes and Linux containers on Linux nodes.

Kubernetes version	**Host OS version (Kubernetes Node)**		
	Windows Server 1709	*Windows Server 1803*	*Windows Server 1809/Windows Server 2019*
Kubernetes v1.14	Not Supported	Not Supported	Supported for Windows Server containers Builds 17763.* with Docker EE-basic 18.09

> **Note:** We don't expect all Windows customers to update the operating system for their apps frequently. Upgrading your applications is what dictates and necessitates upgrading or introducing new nodes to the cluster. For the customers that chose to upgrade their operating system for containers running on Kubernetes, we will offer guidance and step-by-step instructions when we add support for a new operating system version. This guidance will include recommended upgrade procedures for upgrading user applications together with cluster nodes. Windows nodes adhere to Kubernetes version-skew policy (node to control plane versioning) the same way as Linux nodes do today.

> **Note:** The Windows Server Host Operating System is subject to the Windows Server licensing. The Windows Container images are subject to the Supplemental License Terms for Windows containers.

Note: Windows containers with process isolation have strict compatibility rules, where the host OS version must match the container base image OS version. Once we support Windows containers with Hyper-V isolation in Kubernetes, the limitation and compatibility rules will change.

Key Kubernetes elements work the same way in Windows as they do in Linux. In this section, we talk about some of the key workload enablers and how they map to Windows.

- Pods

  A Pod is the basic building block of Kubernetes-the smallest and simplest unit in the Kubernetes object model that you create or deploy. The following Pod capabilities, properties and events are supported with Windows containers:

  - Single or multiple containers per Pod with process isolation and volume sharing
  - Pod status fields
  - Readiness and Liveness probes
  - postStart & preStop container lifecycle events
  - ConfigMap, Secrets: as environment variables or volumes
  - EmptyDir
  - Named pipe host mounts
  - Resource limits

- Controllers

  Kubernetes controllers handle the desired state of Pods. The following workload controllers are supported with Windows containers:

  - ReplicaSet
  - ReplicationController
  - Deployments
  - StatefulSets
  - DaemonSet
  - Job
  - CronJob

- Services

  A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. You can use services for cross-operating system connectivity. In Windows, services can utilize the following types, properties and capabilities:

  - Service Environment variables
  - NodePort
  - ClusterIP
  - LoadBalancer
  - ExternalName
  - Headless services

Pods, Controllers and Services are critical elements to managing Windows workloads on Kubernetes. However, on their own they are not enough to enable the proper lifecycle

management of Windows workloads in a dynamic cloud native environment. We added support for the following features:

- Pod and container metrics
- Horizontal Pod Autoscaler support
- kubectl Exec
- Resource Quotas
- Scheduler preemption

Container Runtime

Docker EE-basic 18.09 is required on Windows Server 2019 / 1809 nodes for Kubernetes. This works with the dockershim code included in the kubelet. Additional runtimes such as CRI-ContainerD may be supported in later Kubernetes versions.

Persistent Storage

Kubernetes volumes enable complex applications, with data persistence and Pod volume sharing requirements, to be deployed on Kubernetes. Management of persistent volumes associated with a specific storage back-end or protocol includes actions such as: provisioning/de-provisioning/ resizing of volumes, attaching/detaching a volume to/from a Kubernetes node and mounting/ dismounting a volume to/from individual containers in a pod that needs to persist data. The code implementing these volume management actions for a specific storage back-end or protocol is shipped in the form of a Kubernetes volume plugin. The following broad classes of Kubernetes volume plugins are supported on Windows:

In-tree Volume Plugins

Code associated with in-tree volume plugins ship as part of the core Kubernetes code base. Deployment of in-tree volume plugins do not require installation of additional scripts or deployment of separate containerized plugin components. These plugins can handle: provisioning/de-provisioning and resizing of volumes in the storage backend, attaching/detaching of volumes to/from a Kubernetes node and mounting/dismounting a volume to/from individual containers in a pod. The following in-tree plugins support Windows nodes:

- awsElasticBlockStore
- azureDisk
- azureFile
- gcePersistentDisk
- awsElasticBlockStore
- vsphereVolume

FlexVolume Plugins

Code associated with FlexVolume plugins ship as out-of-tree scripts or binaries that need to be deployed directly on the host. FlexVolume plugins handle attaching/detaching of volumes to/from a Kubernetes node and mounting/dismounting a volume to/from individual containers in a pod. Provisioning/De-provisioning of persistent volumes associated with FlexVolume plugins may be handled through an external provisioner that is typically separate from the FlexVolume plugins. The following FlexVolume plugins, deployed as powershell scripts on the host, support Windows nodes:

- SMB

- iSCSI

CSI Plugins

FEATURE STATE: `Kubernetes v1.16` alpha
This feature is currently in a *alpha* state, meaning:

- The version names contain alpha (e.g. v1alpha1).
- Might be buggy. Enabling the feature may expose bugs. Disabled by default.
- Support for feature may be dropped at any time without notice.
- The API may change in incompatible ways in a later software release without notice.
- Recommended for use only in short-lived testing clusters, due to increased risk of bugs and lack of long-term support.

Code associated with CSIThe Container Storage Interface (CSI) defines a standard interface to expose storage systems to containers. plugins ship as out-of-tree scripts and binaries that are typically distributed as container images and deployed using standard Kubernetes constructs like DaemonSets and StatefulSets. CSI plugins handle a wide range of volume management actions in Kubernetes: provisioning/de-provisioning/resizing of volumes, attaching/detaching of volumes to/from a Kubernetes node and mounting/dismounting a volume to/from individual containers in a pod, backup/restore of persistent data using snapshots and cloning. CSI plugins typically consist of node plugins (that run on each node as a DaemonSet) and controller plugins.

CSI node plugins (especially those associated with persistent volumes exposed as either block devices or over a shared file-system) need to perform various privileged operations like scanning of disk devices, mounting of file systems, etc. These operations differ for each host operating system. For Linux worker nodes, containerized CSI node plugins are typically deployed as privileged containers. For Windows worker nodes, privileged operations for containerized CSI node plugins is supported using csi-proxy, a community-managed, stand-alone binary that needs to be pre-installed on each Windows node. Please refer to the deployment guide of the CSI plugin you wish to deploy for further details.

Networking

Networking for Windows containers is exposed through CNI plugins. Windows containers function similarly to virtual machines in regards to networking. Each container has a virtual network adapter (vNIC) which is connected to a Hyper-V virtual switch (vSwitch). The Host Networking Service (HNS) and the Host Compute Service (HCS) work together to create containers and attach container vNICs to networks. HCS is responsible for the management of containers whereas HNS is responsible for the management of networking resources such as:

- Virtual networks (including creation of vSwitches)
- Endpoints / vNICs
- Namespaces
- Policies (Packet encapsulations, Load-balancing rules, ACLs, NAT'ing rules, etc.)

The following service spec types are supported:

- NodePort
- ClusterIP
- LoadBalancer
- ExternalName

Windows supports five different networking drivers/modes: L2bridge, L2tunnel, Overlay, Transparent, and NAT. In a heterogeneous cluster with Windows and Linux worker nodes, you need to select a networking solution that is compatible on both Windows and Linux. The following out-of-tree plugins are supported on Windows, with recommendations on when to use each CNI:

Network Driver	**Description**	**Container Packet Modifications**	**Network Plugins**	**Network Plugin Characteristics**
L2bridge	Containers are attached to an external vSwitch. Containers are attached to the underlay network, although the physical network doesn't need to learn the container MACs because they are rewritten on ingress/ egress. Inter-container traffic is bridged inside the container host.	MAC is rewritten to host MAC, IP remains the same.	win-bridge, Azure-CNI, Flannel host-gateway uses win-bridge	win-bridge uses L2bridge network mode, connects containers to the underlay of hosts, offering best performance. Requires user-defined routes (UDR) for inter-node connectivity.
L2Tunnel	This is a special case of l2bridge, but only used on Azure. All packets are sent to the virtualization host where SDN policy is applied.	MAC rewritten, IP visible on the underlay network	Azure-CNI	Azure-CNI allows integration of containers with Azure vNET, and allows them to leverage the set of capabilities that Azure Virtual Network provides. For example, securely connect to Azure services or use Azure NSGs. See azure-cni for some examples
Overlay (Overlay networking for Windows in Kubernetes is in *alpha* stage)	Containers are given a vNIC connected to an external vSwitch. Each overlay network gets its own IP subnet, defined by a custom IP prefix.The overlay network driver uses VXLAN encapsulation.	Encapsulated with an outer header.	Win-overlay, Flannel VXLAN (uses win-overlay)	win-overlay should be used when virtual container networks are desired to be isolated from underlay of hosts (e.g. for security reasons). Allows for IPs to be re-used for different overlay networks (which have different VNID tags) if you are restricted on IPs in your datacenter. This option requires KB4489899 on Windows Server 2019.

Network Driver	Description	Container Packet Modifications	Network Plugins	Network Plugin Characteristics
Transparent (special use case for ovn-kubernetes)	Requires an external vSwitch. Containers are attached to an external vSwitch which enables intra-pod communication via logical networks (logical switches and routers).	Packet is encapsulated either via GENEVE or STT tunneling to reach pods which are not on the same host. Packets are forwarded or dropped via the tunnel metadata information supplied by the ovn network controller. NAT is done for north-south communication.	ovn-kubernetes	Deploy via ansible. Distributed ACLs can be applied via Kubernetes policies. IPAM support. Load-balancing can be achieved without kube-proxy. NATing is done without using iptables/ netsh.
NAT (*not used in Kubernetes*)	Containers are given a vNIC connected to an internal vSwitch. DNS/DHCP is provided using an internal component called WinNAT	MAC and IP is rewritten to host MAC/IP.	nat	Included here for completeness

As outlined above, the Flannel CNI meta plugin is also supported on Windows via the VXLAN network backend (**alpha support** ; delegates to win-overlay) and host-gateway network backend (stable support; delegates to win-bridge). This plugin supports delegating to one of the reference CNI plugins (win-overlay, win-bridge), to work in conjunction with Flannel daemon on Windows (Flanneld) for automatic node subnet lease assignment and HNS network creation. This plugin reads in its own configuration file (cni.conf), and aggregates it with the environment variables from the FlannelD generated subnet.env file. It then delegates to one of the reference CNI plugins for network plumbing, and sends the correct configuration containing the node-assigned subnet to the IPAM plugin (e.g. host-local).

For the node, pod, and service objects, the following network flows are supported for TCP/UDP traffic:

- Pod -> Pod (IP)
- Pod -> Pod (Name)
- Pod -> Service (Cluster IP)
- Pod -> Service (PQDN, but only if there are no ".")
- Pod -> Service (FQDN)
- Pod -> External (IP)
- Pod -> External (DNS)
- Node -> Pod
- Pod -> Node

The following IPAM options are supported on Windows:

- Host-local
- HNS IPAM (Inbox platform IPAM, this is a fallback when no IPAM is set)
- Azure-vnet-ipam (for azure-cni only)

Limitations

Control Plane

Windows is only supported as a worker node in the Kubernetes architecture and component matrix. This means that a Kubernetes cluster must always include Linux master nodes, zero or more Linux worker nodes, and zero or more Windows worker nodes.

Compute

Resource management and process isolation

Linux cgroups are used as a pod boundary for resource controls in Linux. Containers are created within that boundary for network, process and file system isolation. The cgroups APIs can be used to gather cpu/io/memory stats. In contrast, Windows uses a Job object per container with a system namespace filter to contain all processes in a container and provide logical isolation from the host. There is no way to run a Windows container without the namespace filtering in place. This means that system privileges cannot be asserted in the context of the host, and thus privileged containers are not available on Windows. Containers cannot assume an identity from the host because the Security Account Manager (SAM) is separate.

Operating System Restrictions

Windows has strict compatibility rules, where the host OS version must match the container base image OS version. Only Windows containers with a container operating system of Windows Server 2019 are supported. Hyper-V isolation of containers, enabling some backward compatibility of Windows container image versions, is planned for a future release.

Feature Restrictions

- TerminationGracePeriod: not implemented
- Single file mapping: to be implemented with CRI-ContainerD
- Termination message: to be implemented with CRI-ContainerD
- Privileged Containers: not currently supported in Windows containers
- HugePages: not currently supported in Windows containers
- The existing node problem detector is Linux-only and requires privileged containers. In general, we don't expect this to be used on Windows because privileged containers are not supported
- Not all features of shared namespaces are supported (see API section for more details)

Memory Reservations and Handling

Windows does not have an out-of-memory process killer as Linux does. Windows always treats all user-mode memory allocations as virtual, and pagefiles are mandatory. The net effect is that Windows won't reach out of memory conditions the same way Linux does, and processes page to

disk instead of being subject to out of memory (OOM) termination. If memory is over-provisioned and all physical memory is exhausted, then paging can slow down performance.

Keeping memory usage within reasonable bounds is possible with a two-step process. First, use the kubelet parameters `--kubelet-reserve` and/or `--system-reserve` to account for memory usage on the node (outside of containers). This reduces NodeAllocatable). As you deploy workloads, use resource limits (must set only limits or limits must equal requests) on containers. This also subtracts from NodeAllocatable and prevents the scheduler from adding more pods once a node is full.

A best practice to avoid over-provisioning is to configure the kubelet with a system reserved memory of at least 2GB to account for Windows, Docker, and Kubernetes processes.

The behavior of the flags behave differently as described below:

- `--kubelet-reserve`, `--system-reserve` , and `--eviction-hard` flags update Node Allocatable
- Eviction by using `--enforce-node-allocable` is not implemented
- Eviction by using `--eviction-hard` and `--eviction-soft` are not implemented
- MemoryPressure Condition is not implemented
- There are no OOM eviction actions taken by the kubelet
- Kubelet running on the windows node does not have memory restrictions. `--kubelet-reserve` and `--system-reserve` do not set limits on kubelet or processes running on the host. This means kubelet or a process on the host could cause memory resource starvation outside the node-allocatable and scheduler

Storage

Windows has a layered filesystem driver to mount container layers and create a copy filesystem based on NTFS. All file paths in the container are resolved only within the context of that container.

- Volume mounts can only target a directory in the container, and not an individual file
- Volume mounts cannot project files or directories back to the host filesystem
- Read-only filesystems are not supported because write access is always required for the Windows registry and SAM database. However, read-only volumes are supported
- Volume user-masks and permissions are not available. Because the SAM is not shared between the host & container, there's no mapping between them. All permissions are resolved within the context of the container

As a result, the following storage functionality is not supported on Windows nodes

- Volume subpath mounts. Only the entire volume can be mounted in a Windows container.
- Subpath volume mounting for Secrets
- Host mount projection
- DefaultMode (due to UID/GID dependency)
- Read-only root filesystem. Mapped volumes still support readOnly
- Block device mapping
- Memory as the storage medium
- File system features like uui/guid, per-user Linux filesystem permissions
- NFS based storage/volume support
- Expanding the mounted volume (resizefs)

Networking

Windows Container Networking differs in some important ways from Linux networking. The Microsoft documentation for Windows Container Networking contains additional details and background.

The Windows host networking networking service and virtual switch implement namespacing and can create virtual NICs as needed for a pod or container. However, many configurations such as DNS, routes, and metrics are stored in the Windows registry database rather than /etc/â€¦ files as they are on Linux. The Windows registry for the container is separate from that of the host, so concepts like mapping /etc/resolv.conf from the host into a container don't have the same effect they would on Linux. These must be configured using Windows APIs run in the context of that container. Therefore CNI implementations need to call the HNS instead of relying on file mappings to pass network details into the pod or container.

The following networking functionality is not supported on Windows nodes

- Host networking mode is not available for Windows pods
- Local NodePort access from the node itself fails (works for other nodes or external clients)
- Accessing service VIPs from nodes will be available with a future release of Windows Server
- Overlay networking support in kube-proxy is an alpha release. In addition, it requires KB4482887 to be installed on Windows Server 2019
- Local Traffic Policy and DSR mode
- Windows containers connected to l2bridge, l2tunnel, or overlay networks do not support communicating over the IPv6 stack. There is outstanding Windows platform work required to enable these network drivers to consume IPv6 addresses and subsequent Kubernetes work in kubelet, kube-proxy, and CNI plugins.
- Outbound communication using the ICMP protocol via the win-overlay, win-bridge, and Azure-CNI plugin. Specifically, the Windows data plane (VFP) doesn't support ICMP packet transpositions. This means:
    - ICMP packets directed to destinations within the same network (e.g. pod to pod communication via ping) work as expected and without any limitations
    - TCP/UDP packets work as expected and without any limitations
    - ICMP packets directed to pass through a remote network (e.g. pod to external internet communication via ping) cannot be transposed and thus will not be routed back to their source
    - Since TCP/UDP packets can still be transposed, one can substitute `ping <destination>` with `curl <destination>` to be able to debug connectivity to the outside world.

These features were added in Kubernetes v1.15:

- `kubectl port-forward`

CNI Plugins

- Windows reference network plugins win-bridge and win-overlay do not currently implement CNI spec v0.4.0 due to missing "CHECK" implementation.
- The Flannel VXLAN CNI has the following limitations on Windows:

1. Node-pod connectivity isn't possible by design. It's only possible for local pods with Flannel PR 1096

2. We are restricted to using VNI 4096 and UDP port 4789. The VNI limitation is being worked on and will be overcome in a future release (open-source flannel changes). See the official Flannel VXLAN backend docs for more details on these parameters.

DNS

- ClusterFirstWithHostNet is not supported for DNS. Windows treats all names with a â€˜.' as a FQDN and skips PQDN resolution
- On Linux, you have a DNS suffix list, which is used when trying to resolve PQDNs. On Windows, we only have 1 DNS suffix, which is the DNS suffix associated with that pod's namespace (mydns.svc.cluster.local for example). Windows can resolve FQDNs and services or names resolvable with just that suffix. For example, a pod spawned in the default namespace, will have the DNS suffix **default.svc.cluster.local**. On a Windows pod, you can resolve both **kubernetes.default.svc.cluster.local** and **kubernetes**, but not the in-betweens, like **kubernetes.default** or **kubernetes.default.svc**.
- On Windows, there are multiple DNS resolvers that can be used. As these come with slightly different behaviors, using the `Resolve-DNSName` utility for name query resolutions is recommended.

Security

Secrets are written in clear text on the node's volume (as compared to tmpfs/in-memory on linux). This means customers have to do two things

1. Use file ACLs to secure the secrets file location
2. Use volume-level encryption using BitLocker

RunAsUser is not currently supported on Windows. The workaround is to create local accounts before packaging the container. The RunAsUsername capability may be added in a future release.

Linux specific pod security context privileges such as SELinux, AppArmor, Seccomp, Capabilities (POSIX Capabilities), and others are not supported.

In addition, as mentioned already, privileged containers are not supported on Windows.

API

There are no differences in how most of the Kubernetes APIs work for Windows. The subtleties around what's different come down to differences in the OS and container runtime. In certain situations, some properties on workload APIs such as Pod or Container were designed with an assumption that they are implemented on Linux, failing to run on Windows.

At a high level, these OS concepts are different:

- Identity - Linux uses userID (UID) and groupID (GID) which are represented as integer types. User and group names are not canonical - they are just an alias in `/etc/groups` or `/etc/passwd` back to UID+GID. Windows uses a larger binary security identifier (SID) which is stored in the Windows Security Access Manager (SAM) database. This database is not shared between the host and containers, or between containers.
- File permissions - Windows uses an access control list based on SIDs, rather than a bitmask of permissions and UID+GID
- File paths - convention on Windows is to use `\` instead of `/`. The Go IO libraries typically accept both and just make it work, but when you're setting a path or command line that's interpreted inside a container, `\` may be needed.

- Signals - Windows interactive apps handle termination differently, and can implement one or more of these:
    - A UI thread handles well-defined messages including WM_CLOSE
    - Console apps handle ctrl-c or ctrl-break using a Control Handler
    - Services register a Service Control Handler function that can accept SERVICE_CONTROL_STOP control codes

Exit Codes follow the same convention where 0 is success, nonzero is failure. The specific error codes may differ across Windows and Linux. However, exit codes passed from the Kubernetes components (kubelet, kube-proxy) are unchanged.

V1.Container

- V1.Container.ResourceRequirements.limits.cpu and V1.Container.ResourceRequirements.limits.memory - Windows doesn't use hard limits for CPU allocations. Instead, a share system is used. The existing fields based on millicores are scaled into relative shares that are followed by the Windows scheduler. see: kuberuntime/helpers_windows.go, see: resource controls in Microsoft docs
    - Huge pages are not implemented in the Windows container runtime, and are not available. They require asserting a user privilege that's not configurable for containers.
- V1.Container.ResourceRequirements.requests.cpu and V1.Container.ResourceRequirements.requests.memory - Requests are subtracted from node available resources, so they can be used to avoid overprovisioning a node. However, they cannot be used to guarantee resources in an overprovisioned node. They should be applied to all containers as a best practice if the operator wants to avoid overprovisioning entirely.
- V1.Container.SecurityContext.allowPrivilegeEscalation - not possible on Windows, none of the capabilities are hooked up
- V1.Container.SecurityContext.Capabilities - POSIX capabilities are not implemented on Windows
- V1.Container.SecurityContext.privileged - Windows doesn't support privileged containers
- V1.Container.SecurityContext.procMount - Windows doesn't have a /proc filesystem
- V1.Container.SecurityContext.readOnlyRootFilesystem - not possible on Windows, write access is required for registry & system processes to run inside the container
- V1.Container.SecurityContext.runAsGroup - not possible on Windows, no GID support
- V1.Container.SecurityContext.runAsNonRoot - Windows does not have a root user. The closest equivalent is ContainerAdministrator which is an identity that doesn't exist on the node.
- V1.Container.SecurityContext.runAsUser - not possible on Windows, no UID support as int.
- V1.Container.SecurityContext.seLinuxOptions - not possible on Windows, no SELinux
- V1.Container.terminationMessagePath - this has some limitations in that Windows doesn't support mapping single files. The default value is /dev/termination-log, which does work because it does not exist on Windows by default.

V1.Pod

- V1.Pod.hostIPC, v1.pod.hostpid - host namespace sharing is not possible on Windows
- V1.Pod.hostNetwork - There is no Windows OS support to share the host network
- V1.Pod.dnsPolicy - ClusterFirstWithHostNet - is not supported because Host Networking is not supported on Windows.
- V1.Pod.podSecurityContext - see V1.PodSecurityContext below

- V1.Pod.shareProcessNamespace - this is a beta feature, and depends on Linux namespaces which are not implemented on Windows. Windows cannot share process namespaces or the container's root filesystem. Only the network can be shared.
- V1.Pod.terminationGracePeriodSeconds - this is not fully implemented in Docker on Windows, see: reference. The behavior today is that the ENTRYPOINT process is sent CTRL_SHUTDOWN_EVENT, then Windows waits 5 seconds by default, and finally shuts down all processes using the normal Windows shutdown behavior. The 5 second default is actually in the Windows registry inside the container, so it can be overridden when the container is built.
- V1.Pod.volumeDevices - this is a beta feature, and is not implemented on Windows. Windows cannot attach raw block devices to pods.
- V1.Pod.volumes - EmptyDir, Secret, ConfigMap, HostPath - all work and have tests in TestGrid
    - V1.emptyDirVolumeSource - the Node default medium is disk on Windows. Memory is not supported, as Windows does not have a built-in RAM disk.
- V1.VolumeMount.mountPropagation - mount propagation is not supported on Windows.

V1.PodSecurityContext

None of the PodSecurityContext fields work on Windows. They're listed here for reference.

- V1.PodSecurityContext.SELinuxOptions - SELinux is not available on Windows
- V1.PodSecurityContext.RunAsUser - provides a UID, not available on Windows
- V1.PodSecurityContext.RunAsGroup - provides a GID, not available on Windows
- V1.PodSecurityContext.RunAsNonRoot - Windows does not have a root user. The closest equivalent is ContainerAdministrator which is an identity that doesn't exist on the node.
- V1.PodSecurityContext.SupplementalGroups - provides GID, not available on Windows
- V1.PodSecurityContext.Sysctls - these are part of the Linux sysctl interface. There's no equivalent on Windows.

Getting Help and Troubleshooting

Your main source of help for troubleshooting your Kubernetes cluster should start with this section. Some additional, Windows-specific troubleshooting help is included in this section. Logs are an important element of troubleshooting issues in Kubernetes. Make sure to include them any time you seek troubleshooting assistance from other contributors. Follow the instructions in the SIG-Windows contributing guide on gathering logs.

1. How do I know start.ps1 completed successfully?

   You should see kubelet, kube-proxy, and (if you chose Flannel as your networking solution) flanneld host-agent processes running on your node, with running logs being displayed in separate PowerShell windows. In addition to this, your Windows node should be listed as "Ready" in your Kubernetes cluster.

2. Can I configure the Kubernetes node processes to run in the background as services?

   Kubelet and kube-proxy are already configured to run as native Windows Services, offering resiliency by re-starting the services automatically in the event of failure (for example a process crash). You have two options for configuring these node components as services.

    1. As native Windows Services

       Kubelet & kube-proxy can be run as native Windows Services using `sc.exe`.

```
# Create the services for kubelet and kube-proxy in two
separate commands
sc.exe create <component_name> binPath=
"<path_to_binary> --service <other_args>"

# Please note that if the arguments contain spaces, they
must be escaped.
sc.exe create kubelet binPath= "C:\kubelet.exe --service
--hostname-override 'minion' <other_args>"

# Start the services
Start-Service kubelet
Start-Service kube-proxy

# Stop the service
Stop-Service kubelet (-Force)
Stop-Service kube-proxy (-Force)

# Query the service status
Get-Service kubelet
Get-Service kube-proxy
```

2. Using nssm.exe

   You can also always use alternative service managers like nssm.exe to run these processes (flanneld, kubelet & kube-proxy) in the background for you. You can use this sample script, leveraging nssm.exe to register kubelet, kube-proxy, and flanneld.exe to run as Windows services in the background.

```
register-svc.ps1 -NetworkMode <Network mode> -
ManagementIP <Windows Node IP> -ClusterCIDR <Cluster
subnet> -KubeDnsServiceIP <Kube-dns Service IP> -LogDir
<Directory to place logs>

# NetworkMode      = The network mode l2bridge (flannel
host-gw, also the default value) or overlay (flannel
vxlan) chosen as a network solution
# ManagementIP     = The IP address assigned to the
Windows node. You can use ipconfig to find this
# ClusterCIDR      = The cluster subnet range. (Default
value 10.244.0.0/16)
# KubeDnsServiceIP = The Kubernetes DNS service IP
(Default value 10.96.0.10)
# LogDir           = The directory where kubelet and
kube-proxy logs are redirected into their respective
output files (Default value C:\k)
```

   If the above referenced script is not suitable, you can manually configure nssm.exe using the following examples.

```
# Register flanneld.exe
nssm install flanneld C:\flannel\flanneld.exe
```

```
nssm set flanneld AppParameters --kubeconfig-file=c:\k\co
nfig --iface=<ManagementIP> --ip-masq=1 --kube-subnet-
mgr=1
nssm set flanneld AppEnvironmentExtra
NODE_NAME=<hostname>
nssm set flanneld AppDirectory C:\flannel
nssm start flanneld

# Register kubelet.exe
# Microsoft releases the pause infrastructure container
at mcr.microsoft.com/k8s/core/pause:1.2.0
# For more info search for "pause" in the "Guide for
adding Windows Nodes in Kubernetes"
nssm install kubelet C:\k\kubelet.exe
nssm set kubelet AppParameters --hostname-
override=<hostname> --v=6 --pod-infra-container-
image=mcr.microsoft.com/k8s/core/pause:1.2.0 --resolv-
conf="" --allow-privileged=true --enable-debugging-
handlers --cluster-dns=<DNS-service-IP> --cluster-
domain=cluster.local --kubeconfig=c:\k\config --hairpin-
mode=promiscuous-bridge --image-pull-progress-
deadline=20m --cgroups-per-qos=false  --log-dir=<log
directory> --logtostderr=false --enforce-node-
allocatable="" --network-plugin=cni --cni-bin-dir=c:\k\cn
i --cni-conf-dir=c:\k\cni\config
nssm set kubelet AppDirectory C:\k
nssm start kubelet

# Register kube-proxy.exe (l2bridge / host-gw)
nssm install kube-proxy C:\k\kube-proxy.exe
nssm set kube-proxy AppDirectory c:\k
nssm set kube-proxy AppParameters --v=4 --proxy-
mode=kernelspace --hostname-override=<hostname>--
kubeconfig=c:\k\config --enable-dsr=false --log-dir=<log
directory> --logtostderr=false
nssm.exe set kube-proxy AppEnvironmentExtra
KUBE_NETWORK=cbr0
nssm set kube-proxy DependOnService kubelet
nssm start kube-proxy

# Register kube-proxy.exe (overlay / vxlan)
nssm install kube-proxy C:\k\kube-proxy.exe
nssm set kube-proxy AppDirectory c:\k
nssm set kube-proxy AppParameters --v=4 --proxy-
mode=kernelspace --feature-gates="WinOverlay=true" --
hostname-override=<hostname> --kubeconfig=c:\k\config --
network-name=vxlan0 --source-vip=<source-vip> --enable-
dsr=false --log-dir=<log directory> --logtostderr=false
nssm set kube-proxy DependOnService kubelet
nssm start kube-proxy
```

For initial troubleshooting, you can use the following flags in nssm.exe to redirect stdout and stderr to a output file:

```
nssm set <Service Name> AppStdout C:\k\mysvc.log
nssm set <Service Name> AppStderr C:\k\mysvc.log
```

For additional details, see official nssm usage docs.

3. My Windows Pods do not have network connectivity

   If you are using virtual machines, ensure that MAC spoofing is enabled on all the VM network adapter(s).

4. My Windows Pods cannot ping external resources

   Windows Pods do not have outbound rules programmed for the ICMP protocol today. However, TCP/UDP is supported. When trying to demonstrate connectivity to resources outside of the cluster, please substitute `ping <IP>` with corresponding `curl <IP>` commands.

   If you are still facing problems, most likely your network configuration in cni.conf deserves some extra attention. You can always edit this static file. The configuration update will apply to any newly created Kubernetes resources.

   One of the Kubernetes networking requirements (see Kubernetes model) is for cluster communication to occur without NAT internally. To honor this requirement, there is an ExceptionList for all the communication where we do not want outbound NAT to occur. However, this also means that you need to exclude the external IP you are trying to query from the ExceptionList. Only then will the traffic originating from your Windows pods be SNAT'ed correctly to receive a response from the outside world. In this regard, your ExceptionList in `cni.conf` should look as follows:

   ```
   "ExceptionList": [
                   "10.244.0.0/16",  # Cluster subnet
                   "10.96.0.0/12",   # Service subnet
                   "10.127.130.0/24" # Management (host) subnet
               ]
   ```

5. My Windows node cannot access NodePort service

   Local NodePort access from the node itself fails. This is a known limitation. NodePort access works from other nodes or external clients.

6. vNICs and HNS endpoints of containers are being deleted

   This issue can be caused when the `hostname-override` parameter is not passed to kube-proxy. To resolve it, users need to pass the hostname to kube-proxy as follows:

   ```
   C:\k\kube-proxy.exe --hostname-override=$(hostname)
   ```

7. With flannel my nodes are having issues after rejoining a cluster

   Whenever a previously deleted node is being re-joined to the cluster, flannelD tries to assign a new pod subnet to the node. Users should remove the old pod subnet configuration files in the following paths:

   ```
   Remove-Item C:\k\SourceVip.json
   Remove-Item C:\k\SourceVipRequest.json
   ```

8. After launching `start.ps1`, flanneld is stuck in "Waiting for the Network to be created"

   There are numerous reports of this issue which are being investigated; most likely it is a timing issue for when the management IP of the flannel network is set. A workaround is to simply relaunch start.ps1 or relaunch it manually as follows:

   ```
   PS C:> [Environment]::SetEnvironmentVariable("NODE_NAME", "<Windows_Worker_Hostname>")
   PS C:> C:\flannel\flanneld.exe --kubeconfig-file=c:\k\config --iface=<Windows_Worker_Node_IP> --ip-masq=1 --kube-subnet-mgr=1
   ```

9. My Windows Pods cannot launch because of missing `/run/flannel/subnet.env`

   This indicates that Flannel didn't launch correctly. You can either try to restart flanneld.exe or you can copy the files over manually from `/run/flannel/subnet.env` on the Kubernetes master to`C:\run\flannel\subnet.env` on the Windows worker node and modify the `FLANNEL_SUBNET` row to a different number. For example, if node subnet 10.244.4.1/24 is desired:

   ```
   FLANNEL_NETWORK=10.244.0.0/16
   FLANNEL_SUBNET=10.244.4.1/24
   FLANNEL_MTU=1500
   FLANNEL_IPMASQ=true
   ```

10. My Windows node cannot access my services using the service IP

    This is a known limitation of the current networking stack on Windows. Windows Pods are able to access the service IP however.

11. No network adapter is found when starting kubelet

    The Windows networking stack needs a virtual adapter for Kubernetes networking to work. If the following commands return no results (in an admin shell), virtual network creation â€" a necessary prerequisite for Kubelet to work â€" has failed:

    ```
    Get-HnsNetwork | ? Name -ieq "cbr0"
    Get-NetAdapter | ? Name -Like "vEthernet (Ethernet*"
    ```

    Often it is worthwhile to modify the InterfaceName parameter of the start.ps1 script, in cases where the host's network adapter isn't "Ethernet". Otherwise, consult the output of the `start-kubelet.ps1` script to see if there are errors during virtual network creation.

12. My Pods are stuck at "Container Creating" or restarting over and over

    Check that your pause image is compatible with your OS version. The instructions assume that both the OS and the containers are version 1803. If you have a later version of Windows, such as an Insider build, you need to adjust the images accordingly. Please refer to the Microsoft's Docker repository for images. Regardless, both the pause image Dockerfile and the sample service expect the image to be tagged as :latest.

    Starting with Kubernetes v1.14, Microsoft releases the pause infrastructure container at `mcr.microsoft.com/k8s/core/pause:1.2.0`. For more information search for "pause" in the Guide for adding Windows Nodes in Kubernetes.

13. DNS resolution is not properly working

    Check the DNS limitations for Windows in this section.

14. `kubectl port-forward` fails with "unable to do port forwarding: wincat not found"

    This was implemented in Kubernetes 1.15, and the pause infrastructure container `mcr.microsoft.com/k8s/core/pause:1.2.0`. Be sure to use these versions or newer ones. If you would like to build your own pause infrastructure container, be sure to include wincat

15. My Kubernetes installation is failing because my Windows Server node is behind a proxy

    If you are behind a proxy, the following PowerShell environment variables must be defined:

    ```
    [Environment]::SetEnvironmentVariable("HTTP_PROXY", "http://proxy.example.com:80/", [EnvironmentVariableTarget]::Machine)
    [Environment]::SetEnvironmentVariable("HTTPS_PROXY", "http://proxy.example.com:443/", [EnvironmentVariableTarget]::Machine)
    ```

16. What is a `pause` container?

    In a Kubernetes Pod, an infrastructure or "pause" container is first created to host the container endpoint. Containers that belong to the same pod, including infrastructure and worker containers, share a common network namespace and endpoint (same IP and port space). Pause containers are needed to accomodate worker containers crashing or restarting without losing any of the networking configuration.

    The "pause" (infrastructure) image is hosted on Microsoft Container Registry (MCR). You can access it using `docker pull mcr.microsoft.com/k8s/core/pause:1.2.0`. For more details, see the DOCKERFILE.

Further investigation

If these steps don't resolve your problem, you can get help running Windows containers on Windows nodes in Kubernetes through:

- StackOverflow Windows Server Container topic
- Kubernetes Official Forum discuss.kubernetes.io
- Kubernetes Slack #SIG-Windows Channel

Reporting Issues and Feature Requests

If you have what looks like a bug, or you would like to make a feature request, please use the GitHub issue tracking system. You can open issues on GitHub and assign them to SIG-Windows. You should first search the list of issues in case it was reported previously and comment with your experience on the issue and add additional logs. SIG-Windows Slack is also a great avenue to get some initial support and troubleshooting ideas prior to creating a ticket.

If filing a bug, please include detailed information about how to reproduce the problem, such as:

- Kubernetes version: kubectl version
- Environment details: Cloud provider, OS distro, networking choice and configuration, and Docker version
- Detailed steps to reproduce the problem
- Relevant logs
- Tag the issue sig/windows by commenting on the issue with `/sig windows` to bring it to a SIG-Windows member's attention

What's next

We have a lot of features in our roadmap. An abbreviated high level list is included below, but we encourage you to view our roadmap project and help us make Windows support better by contributing.

CRI-ContainerD

containerdA container runtime with an emphasis on simplicity, robustness and portability is another OCI-compliant runtime that recently graduated as a CNCFCloud Native Computing Foundation project. It's currently tested on Linux, but 1.3 will bring support for Windows and Hyper-V. [reference]

The CRI-ContainerD interface will be able to manage sandboxes based on Hyper-V. This provides a foundation where RuntimeClass could be implemented for new use cases including:

- Hypervisor-based isolation between pods for additional security
- Backwards compatibility allowing a node to run a newer Windows Server version without requiring containers to be rebuilt
- Specific CPU/NUMA settings for a pod
- Memory isolation and reservations

Hyper-V isolation

The existing Hyper-V isolation support, an experimental feature as of v1.10, will be deprecated in the future in favor of the CRI-ContainerD and RuntimeClass features mentioned above. To use the current features and create a Hyper-V isolated container, the kubelet should be started with feature gates `HyperVContainer=true` and the Pod should include the annotation `experimental.windows.kubernetes.io/isolation-type=hyperv`. In the experiemental release, this feature is limited to 1 container per Pod.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iis
spec:
  selector:
    matchLabels:
      app: iis
  replicas: 3
  template:
    metadata:
```

```
    labels:
      app: iis
    annotations:
      experimental.windows.kubernetes.io/isolation-type: hyperv
  spec:
    containers:
    - name: iis
      image: microsoft/iis
      ports:
      - containerPort: 80
```

Deployment with kubeadm and cluster API

Kubeadm is becoming the de facto standard for users to deploy a Kubernetes cluster. Windows node support in kubeadm will come in a future release. We are also making investments in cluster API to ensure Windows nodes are properly provisioned.

A few other key features

- Beta support for Group Managed Service Accounts
- More CNIs
- More Storage Plugins

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on September 12, 2019 at 11:20 PM PST by Update window storage driver list with EBS (#16300) (Page History)

Edit This Page

Guide for adding Windows Nodes in Kubernetes

The Kubernetes platform can now be used to run both Linux and Windows containers. This page shows how one or more Windows nodes can be registered to a cluster.

Objectives

- Register a Windows node to the cluster
- Configure networking so Pods and Services on Linux and Windows can communicate with each other

Before you begin

- Obtain a Windows Server 2019 license (or higher) in order to configure the Windows node that hosts Windows containers. You can use your organization's licenses for the cluster, or acquire one from Microsoft, a reseller, or via the major cloud providers such as GCP, AWS, and Azure by provisioning a virtual machine running Windows Server through their marketplaces. A time-limited trial is also available.

- Build a Linux-based Kubernetes cluster in which you have access to the control-plane (some examples include Creating a single control-plane cluster with kubeadm, AKS Engine, GCE, AWS.

Getting Started: Adding a Windows Node to Your Cluster

Plan IP Addressing

Kubernetes cluster management requires careful planning of your IP addresses so that you do not inadvertently cause network collision. This guide assumes that you are familiar with the Kubernetes networking concepts.

In order to deploy your cluster you need the following address spaces:

Subnet / address range	Description	Default value
Service Subnet	A non-routable, purely virtual subnet that is used by pods to uniformly access services without caring about the network topology. It is translated to/from routable address space by `kube-proxy` running on the nodes.	10.96.0.0/12
Cluster Subnet	This is a global subnet that is used by all pods in the cluster. Each node is assigned a smaller /24 subnet from this for their pods to use. It must be large enough to accommodate all pods used in your cluster. To calculate *minimumsubnet* size: `(number of nodes) + (number of nodes * maximum pods per node that you configure)`. Example: for a 5 node cluster for 100 pods per node: `(5) + (5 * 100) = 505.`	10.244.0.0/16
Kubernetes DNS Service IP	IP address of `kube-dns` service that is used for DNS resolution & cluster service discovery.	10.96.0.10

Review the networking options supported in â€˜Intro to Windows containers in Kubernetes: Supported Functionality: Networking' to determine how you need to allocate IP addresses for your cluster.

Components that run on Windows

While the Kubernetes control-plane runs on your Linux node(s), the following components are configured and run on your Windows node(s).

1. kubelet
2. kube-proxy
3. kubectl (optional)
4. Container runtime

Get the latest binaries from https://github.com/kubernetes/kubernetes/releases, starting with v1.14 or later. The Windows-amd64 binaries for kubeadm, kubectl, kubelet, and kube-proxy can be found under the CHANGELOG link.

Networking Configuration

Once you have a Linux-based Kubernetes control-plane ("Master") node you are ready to choose a networking solution. This guide illustrates using Flannel in VXLAN mode for simplicity.

Configuring Flannel in VXLAN mode on the Linux control-plane

1. Prepare Kubernetes master for Flannel

   Some minor preparation is recommended on the Kubernetes master in our cluster. It is recommended to enable bridged IPv4 traffic to iptables chains when using Flannel. This can be done using the following command:

   ```
   sudo sysctl net.bridge.bridge-nf-call-iptables=1
   ```

2. Download & configure Flannel

   Download the most recent Flannel manifest:

   ```
   wget https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
   ```

   There are two sections you should modify to enable the vxlan networking backend:

   After applying the steps below, the `net-conf.json` section of `kube-flannel.yml` should look as follows:

   ```
   net-conf.json: |
       {
         "Network": "10.244.0.0/16",
         "Backend": {
           "Type": "vxlan",
           "VNI" : 4096,
           "Port": 4789
         }
       }
   ```

Note: The VNI must be set to 4096 and port 4789 for Flannel on Linux to interoperate with Flannel on Windows. Support for other VNIs is coming soon. See the VXLAN documentation for an explanation of these fields.

3. In the `net-conf.json` section of your `kube-flannel.yml`, double-check:

    1. The cluster subnet (e.g. "10.244.0.0/16") is set as per your IP plan.
        - VNI 4096 is set in the backend
        - Port 4789 is set in the backend
    2. In the `cni-conf.json` section of your `kube-flannel.yml`, change the network name to `vxlan0`.

    Your `cni-conf.json` should look as follows:

```
cni-conf.json: |
    {
      "name": "vxlan0",
      "plugins": [
        {
          "type": "flannel",
          "delegate": {
            "hairpinMode": true,
            "isDefaultGateway": true
          }
        },
        {
          "type": "portmap",
          "capabilities": {
            "portMappings": true
          }
        }
      ]
    }
```

4. Apply the Flannel manifest and validate

    Let's apply the Flannel configuration:

```
kubectl apply -f kube-flannel.yml
```

    Next, since the Flannel pods are Linux-based, apply a NodeSelector patch, which can be found here, to the Flannel DaemonSet pod:

```
kubectl patch ds/kube-flannel-ds-amd64 --patch "$(cat node-
selector-patch.yml)" -n=kube-system
```

    After a few minutes, you should see all the pods as running if the Flannel pod network was deployed.

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

    The output looks like as follows:

```
NAMESPACE     NAME
READY         STATUS    RESTARTS   AGE
kube-system   etcd-flannel-master
1/1           Running   0          1m
kube-system   kube-apiserver-flannel-master
1/1           Running   0          1m
kube-system   kube-controller-manager-flannel-master
1/1           Running   0          1m
kube-system   kube-dns-86f4d74b45-hcx8x
3/3           Running   0          12m
kube-system   kube-flannel-ds-54954
1/1           Running   0          1m
kube-system   kube-proxy-Zjlxz
1/1           Running   0          1m
kube-system   kube-scheduler-flannel-master
1/1           Running   0          1m
```

Verify that the Flannel DaemonSet has the NodeSelector applied.

```
kubectl get ds -n kube-system
```

The output looks like as follows. The NodeSelector `beta.kubernetes.io/os=linux` is applied.

```
NAME              DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE
SELECTOR                                                  AGE
kube-flannel-ds   2         2         2       2
2           beta.kubernetes.io/arch=amd64,beta.kubernetes.io/
os=linux   21d
kube-proxy        2         2         2       2
2           beta.kubernetes.io/
os=linux                                     26d
```

Join Windows Worker Node

In this section we'll cover configuring a Windows node from scratch to join a cluster on-prem. If your cluster is on a cloud you'll likely want to follow the cloud specific guides in the public cloud providers section.

Preparing a Windows Node

> **Note:** All code snippets in Windows sections are to be run in a PowerShell environment with elevated permissions (Administrator) on the Windows worker node.

1. Download the SIG Windows tools repository containing install and join scripts

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityPr
otocolType]::Tls12
Start-BitsTransfer https://github.com/kubernetes-sigs/sig-
windows-tools/archive/master.zip
tar -xvf .\master.zip --strip-components 3 sig-windows-tools-
```

```
master/kubeadm/v1.15.0/*
Remove-Item .\master.zip
```

2. Customize the Kubernetes configuration file

```
{
"Cri" : {  // Contains values for container runtime and base
container setup
    "Name" : "dockerd", // Container runtime name
    "Images" : {
        "Pause" : "mcr.microsoft.com/k8s/core/pause:
1.2.0",  // Infrastructure container image
        "Nanoserver" : "mcr.microsoft.com/windows/nanoserver:
1809",  // Base Nanoserver container image
        "ServerCore" : "mcr.microsoft.com/windows/
servercore:ltsc2019"  // Base ServerCore container image
    }
},
"Cni" : {  // Contains values for networking executables
    "Name" : "flannel",  // Name of network fabric
    "Source" : [{ // Contains array of objects containing
values for network daemon(s)
        "Name" : "flanneld",  // Name of network daemon
        "Url" : "https://github.com/coreos/flannel/releases/
download/v0.11.0/flanneld.exe"  // Direct URL pointing to
network daemon executable
        }
    ],
    "Plugin" : {  // Contains values for CNI network plugin
        "Name": "vxlan" // Backend network mechanism to use:
["vxlan" | "bridge"]
    },
    "InterfaceName" : "Ethernet"  // Designated network
interface name on Windows node to use as container network
},
"Kubernetes" : {  // Contains values for Kubernetes node
binaries
    "Source" : {  // Contains values for Kubernetes node
binaries
        "Release" : "1.15.0",  // Version of Kubernetes node
binaries
        "Url" : "https://dl.k8s.io/v1.15.0/kubernetes-node-
windows-amd64.tar.gz"  // Direct URL pointing to Kubernetes
node binaries tarball
    },
    "ControlPlane" : {  // Contains values associated with
Kubernetes control-plane ("Master") node
        "IpAddress" : "kubemasterIP",  // IP address of
control-plane ("Master") node
        "Username" : "localadmin",  // Username on control-
plane ("Master") node with remote SSH access
        "KubeadmToken" : "token",  // Kubeadm bootstrap token
        "KubeadmCAHash" : "discovery-token-ca-cert-hash"  //
Kubeadm CA key hash
```

```
    },
    "KubeProxy" : {  // Contains values for Kubernetes
network proxy configuration
        "Gates" : "WinOverlay=true"  // Comma-separated key-
value pairs passed to kube-proxy feature gate flag
    },
    "Network" : {  // Contains values for IP ranges in CIDR
notation for Kubernetes networking
        "ServiceCidr" : "10.96.0.0/12",  // Service IP
subnet used by Services in CIDR notation
        "ClusterCidr" : "10.244.0.0/16"  // Cluster IP
subnet used by Pods in CIDR notation
    }
},
"Install" : {  // Contains values and configurations for
Windows node installation
    "Destination" : "C:\\ProgramData\\Kubernetes"  //
Absolute DOS path where Kubernetes will be installed on the
Windows node
}
}
```

> **Note:** Users can generate values for the `ControlPlane.KubeadmToken` and `ControlPlane.KubeadmCAHash` fields by running `kubeadm token create --print-join-command` on the Kubernetes control-plane ("Master") node.

1. Install containers and Kubernetes (requires a system reboot)

Use the previously downloaded KubeCluster.ps1 script to install Kubernetes on the Windows Server container host:

```
    .\KubeCluster.ps1 -ConfigFile .\Kubeclustervxlan.json -
install
```

where `-ConfigFile` points to the path of the Kubernetes configuration file.

> **Note:** In the example below, we are using overlay networking mode. This requires Windows Server version 2019 with KB4489899 and at least Kubernetes v1.14 or above. Users that cannot meet this requirement must use `L2bridge` networking instead by selecting `bridge` as the plugin in the configuration file.

alt_text

On the Windows node you target, this step will:

1. Enable Windows Server containers role (and reboot)
2. Download and install the chosen container runtime
3. Download all needed container images
4. Download Kubernetes binaries and add them to the `$PATH` environment variable
5. Download CNI plugins based on the selection made in the Kubernetes Configuration file

6. (Optionally) Generate a new SSH key which is required to connect to the control-plane ("Master") node during joining

> **Note:** For the SSH key generation step, you also need to add the generated public SSH key to the `authorized_keys` file on your (Linux) control-plane node. You only need to do this once. The script prints out the steps you can follow to do this, at the end of its output.

Once installation is complete, any of the generated configuration files or binaries can be modified before joining the Windows node.

Join the Windows Node to the Kubernetes cluster

This section covers how to join a Windows node with Kubernetes installed with an existing (Linux) control-plane, to form a cluster.

Use the previously downloaded KubeCluster.ps1 script to join the Windows node to the cluster:

```
    .\KubeCluster.ps1 -ConfigFile .\Kubeclustervxlan.json -join
```

where `-ConfigFile` points to the path of the Kubernetes configuration file.

alt_text

> **Note:** Should the script fail during the bootstrap or joining procedure for whatever reason, start a new PowerShell session before starting each consecutive join attempt.

This step will perform the following actions:

1. Connect to the control-plane ("Master") node via SSH, to retrieve the Kubeconfig file file.
2. Register kubelet as a Windows service
3. Configure CNI network plugins
4. Create an HNS network on top of the chosen network interface

   > **Note:** This may cause a network blip for a few seconds while the vSwitch is being created.

5. (If vxlan plugin is selected) Open up inbound firewall UDP port 4789 for overlay traffic
6. Register flanneld as a Windows service
7. Register kube-proxy as a Windows service

Now you can view the Windows nodes in your cluster by running the following:

```
kubectl get nodes
```

Remove the Windows Node from the Kubernetes cluster

In this section we'll cover how to remove a Windows node from a Kubernetes cluster.

Use the previously downloaded KubeCluster.ps1 script to remove the Windows node from the cluster:

```
    .\KubeCluster.ps1 -ConfigFile .\Kubeclustervxlan.json -reset
```

where `-ConfigFile` points to the path of the Kubernetes configuration file.

alt_text

This step will perform the following actions on the targeted Windows node:

1. Delete the Windows node from the Kubernetes cluster
2. Stop all running containers
3. Remove all container networking (HNS) resources
4. Unregister all Kubernetes services (flanneld, kubelet, kube-proxy)
5. Delete all Kubernetes binaries (kube-proxy.exe, kubelet.exe, flanneld.exe, kubeadm.exe)
6. Delete all CNI network plugins binaries
7. Delete Kubeconfig file used to access the Kubernetes cluster

Public Cloud Providers

Azure

AKS-Engine can deploy a complete, customizable Kubernetes cluster with both Linux & Windows nodes. There is a step-by-step walkthrough available in the docs on GitHub.

GCP

Users can easily deploy a complete Kubernetes cluster on GCE following this step-by-step walkthrough on GitHub

Deployment with kubeadm and cluster API

Kubeadm is becoming the de facto standard for users to deploy a Kubernetes cluster. Windows node support in kubeadm is an alpha feature since Kubernetes release v1.16. We are also making investments in cluster API to ensure Windows nodes are properly provisioned. For more details, please consult the kubeadm for Windows KEP.

Next Steps

Now that you've configured a Windows worker in your cluster to run Windows containers you may want to add one or more Linux nodes as well to run Linux containers. You are now ready to schedule Windows containers on your cluster.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page
Page last modified on September 08, 2019 at 6:11 AM PST by kubeadm deployment instructions for Windows (#15750) (Page History)

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

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	hosts (SAN)
kube-etcd	etcd-ca		server, client	`localhost`, `127.0.0.1`
kube-etcd-peer	etcd-ca		server, client	`<hostname>`, `<Host_IP>`, `localhost`, `127.0.0.1`
kube-etcd-healthcheck-client	etcd-ca		client	
kube-apiserver-etcd-client	etcd-ca	system:masters	client	
kube-apiserver	kubernetes-ca		server	`<hostname>`, `<Host_IP>`, `<advertise_IP>`, [1]
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	recommended key path	recommended cert path	command	key argument	cert argument
etcd-ca	etcd/ca.key	etcd/ca.crt	kube-apiserver		-etcd-cafile

Default CN	recommended key path	recommended cert path	command	key argument	cert argument
etcd-client	apiserver-etcd-client.key	apiserver-etcd-client.crt	kube-apiserver	-etcd-keyfile	-etcd-certfile
kubernetes-ca	ca.key	ca.crt	kube-apiserver		-client-ca-file
kube-apiserver	apiserver.key	apiserver.crt	kube-apiserver	-tls-private-key-file	-tls-cert-file
apiserver-kubelet-client	apiserver-kubelet-client.key	apiserver-kubelet-client.crt	kube-apiserver		-kubelet-client-certificate
front-proxy-ca	front-proxy-ca.key	front-proxy-ca.crt	kube-apiserver		-requestheader-client-ca-file
front-proxy-client	front-proxy-client.key	front-proxy-client.crt	kube-apiserver	-proxy-client-key-file	-proxy-client-cert-file
etcd-ca	etcd/ca.key	etcd/ca.crt	etcd		-trusted-ca-file, -peer-trusted-ca-file
kube-etcd	etcd/server.key	etcd/server.crt	etcd	-key-file	-cert-file
kube-etcd-peer	etcd/peer.key	etcd/peer.crt	etcd	-peer-key-file	-peer-cert-file
etcd-ca		etcd/ca.crt	etcdctl[2]		-cacert
kube-etcd-healthcheck-client	etcd/healthcheck-client.key	etcd/healthcheck-client.crt	etcdctl[2]	-key	-cert

[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>` (see note)	system:nodes
controller-manager.conf	default-controller-manager	system:kube-controller-manager	
scheduler.conf	default-manager	system:kube-scheduler	

> **Note:** The value of `<nodeName>` for `kubelet.conf` **must** match precisely the value of the node name provided by the kubelet as it registers with the apiserver. For further details, read the Node Authorization.

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>:6443 --certificate-authority <path-to-kubernetes-ca> --embed-certs
```

```
KUBECONFIG=<filename> kubectl config set-credentials <credential-name> --client-key <path-to-key>.pem --client-certificate <path-to-cert>.pem --embed-certs
KUBECONFIG=<filename> kubectl config set-context default-system --cluster default-cluster --user <credential-name>
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-controller-manager.yaml`
scheduler.conf	kube-scheduler	Must be added to manifest in `manifests/kube-scheduler.yaml`

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 12, 2019 at 5:27 PM PST by Restructure the left navigation pane of setup (#14826) (Page History)

Edit This Page

Running in multiple zones

This page describes how to run a cluster in multiple zones.

Introduction

Kubernetes 1.2 adds support for running a single cluster in multiple failure zones (GCE calls them simply "zones", AWS calls them "availability zones", here we'll refer to them as "zones"). This is a lightweight version of a broader Cluster Federation feature (previously referred to by the affectionate nickname "Ubernetes"). Full Cluster Federation allows combining separate

Kubernetes clusters running in different regions or cloud providers (or on-premises data centers). However, many users simply want to run a more available Kubernetes cluster in multiple zones of their single cloud provider, and this is what the multizone support in 1.2 allows (this previously went by the nickname "Ubernetes Lite").

Multizone support is deliberately limited: a single Kubernetes cluster can run in multiple zones, but only within the same region (and cloud provider). Only GCE and AWS are currently supported automatically (though it is easy to add similar support for other clouds or even bare metal, by simply arranging for the appropriate labels to be added to nodes and volumes).

Functionality

When nodes are started, the kubelet automatically adds labels to them with zone information.

Kubernetes will automatically spread the pods in a replication controller or service across nodes in a single-zone cluster (to reduce the impact of failures.) With multiple-zone clusters, this spreading behavior is extended across zones (to reduce the impact of zone failures.) (This is achieved via `SelectorSpreadPriority`). This is a best-effort placement, and so if the zones in your cluster are heterogeneous (e.g. different numbers of nodes, different types of nodes, or different pod resource requirements), this might prevent perfectly even spreading of your pods across zones. If desired, you can use homogeneous zones (same number and types of nodes) to reduce the probability of unequal spreading.

When persistent volumes are created, the `PersistentVolumeLabel` admission controller automatically adds zone labels to them. The scheduler (via the `VolumeZonePredicate` predicate) will then ensure that pods that claim a given volume are only placed into the same zone as that volume, as volumes cannot be attached across zones.

Limitations

There are some important limitations of the multizone support:

- We assume that the different zones are located close to each other in the network, so we don't perform any zone-aware routing. In particular, traffic that goes via services might cross zones (even if 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-central1-a NUM_NODES=3
bash
```

AWS:

```
curl -sS https://get.k8s.io | MULTIZONE=true
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2a NUM_NODES=3 bash
```

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
```

The output is similar to this:

```
NAME                     STATUS                     ROLES
AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-1,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-master
kubernetes-minion-87j9   Ready                      <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-87j9
kubernetes-minion-9vlv   Ready                      <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-9vlv
kubernetes-minion-a12q   Ready                      <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-a12q
```

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-central1-b NUM_NODES=3
kubernetes/cluster/kube-up.sh
```

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-2b NUM_NODES=3 KUBE
_SUBNET_CIDR=172.20.1.0/24 MASTER_INTERNAL_IP=172.20.0.9
kubernetes/cluster/kube-up.sh
```

View the nodes again; 3 more nodes should have launched and be tagged in us-central1-b:

```
kubectl get nodes --show-labels
```

The output is similar to this:

```
NAME                     STATUS                     ROLES
AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>
```

```
16m   v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-1,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-master
kubernetes-minion-281d   Ready                      <none>
2m    v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-281d
kubernetes-minion-87j9   Ready                      <none>
16m   v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-87j9
kubernetes-minion-9vlv   Ready                      <none>
16m   v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-9vlv
kubernetes-minion-a12q   Ready                      <none>
17m   v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-a12q
kubernetes-minion-pp2f   Ready                      <none>
2m    v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-pp2f
kubernetes-minion-wf8i   Ready                      <none>
2m    v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-wf8i
```

Volume affinity

Create a volume using the dynamic volume creation (only PersistentVolumes are supported for zone affinity):

```
kubectl apply -f - <<EOF
{
  "apiVersion": "v1",
  "kind": "PersistentVolumeClaim",
  "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 let's validate that Kubernetes automatically labeled the zone & region the PV was created in.

```
kubectl get pv --show-labels
```

The output is similar to this:

```
NAME           CAPACITY   ACCESSMODES   RECLAIM POLICY
STATUS    CLAIM             STORAGECLASS    REASON    AGE
LABELS
pv-gce-mj4gm   5Gi        RWO           Retain
Bound     default/claim1   manual                    46s
failure-domain.beta.kubernetes.io/region=us-central1,failure-
domain.beta.kubernetes.io/zone=us-central1-a
```

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 apply -f - <<EOF
apiVersion: v1
kind: Pod
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
```

And check node labels:

```
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=n1-standard-2,failure-
domain.beta.kubernetes.io/region=us-central1,failure-
domain.beta.kubernetes.io/zone=us-central1-a,kubernetes.io/
hostname=kubernetes-minion-9vlv
```

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-central1-f NUM_NODES=3
kubernetes/cluster/kube-up.sh
```

AWS:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2c NUM_NODES=3 KUBE
_SUBNET_CIDR=172.20.2.0/24 MASTER_INTERNAL_IP=172.20.0.9
kubernetes/cluster/kube-up.sh
```

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 apply -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 --show-labels
```

```
NAME                     STATUS    ROLES     AGE
VERSION          LABELS
kubernetes-minion-9vlv   Ready     <none>    34m
```

```
v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-9vlv
kubernetes-minion-281d   Ready     <none>   20m
v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-281d
kubernetes-minion-olsh   Ready     <none>   3m
v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
f,kubernetes.io/hostname=kubernetes-minion-olsh
```

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
```

The output is similar to this:

```
LoadBalancer Ingress:   130.211.126.21
```

Set the above IP:

```
export IP=130.211.126.21
```

Explore with curl via IP:

```
curl -s http://${IP}:3000/env | grep HOSTNAME
```

The output is similar to this:

```
  "HOSTNAME": "guestbook-44sep",
```

Again, explore multiple times:

```
(for i in `seq 20`; do curl -s http://${IP}:3000/env | grep
HOSTNAME; done)  | sort | uniq
```

The output is similar to this:

```
  "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_ZO
NE=us-central1-f kubernetes/cluster/kube-down.sh
KUBERNETES_PROVIDER=gce KUBE_USE_EXISTING_MASTER=true KUBE_GCE_ZO
NE=us-central1-b kubernetes/cluster/kube-down.sh
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_ZO
NE=us-west-2c kubernetes/cluster/kube-down.sh
KUBERNETES_PROVIDER=aws KUBE_USE_EXISTING_MASTER=true KUBE_AWS_ZO
NE=us-west-2b kubernetes/cluster/kube-down.sh
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2a kubernetes/
cluster/kube-down.sh
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Building large clusters

Support

At v1.16, 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

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`.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page
Page last modified on June 12, 2019 at 5:27 PM PST by Restructure the left navigation pane of setup (#14826) (Page History)