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
Kubeadm-dind	Docker Desktop
Kubernetes IN Docker	Minishift
	MicroK8s
	IBM Cloud Private-CE (Community Edition)
	IBM Cloud Private-CE (Community Edition) on Linux Containers
	k3s
	Ubuntu on LXD

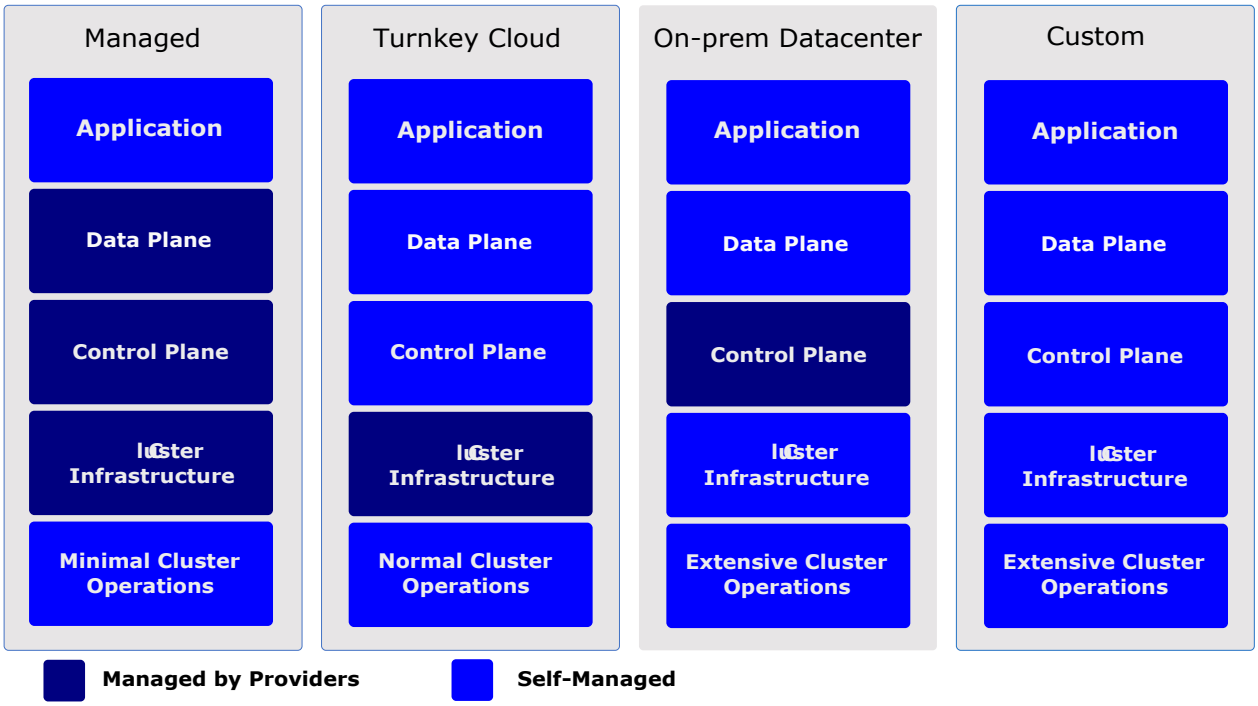
Production environment

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”			
CenturyLink Cloud		âœ”				
Cisco Container Platform			âœ”			
Cloud Foundry Container Runtime (CFCR)				âœ”	âœ”	
CloudStack					âœ”	
Canonical		âœ”		âœ”	âœ”	âœ”
Containership	âœ”	âœ”				
Digital Rebar						âœ”
DigitalOcean	âœ”					
Docker Enterprise		âœ”	âœ”			âœ”
Fedora (Multi Node)Â					âœ”	âœ”

Providers	Managed	Turnkey cloud	On-prem datacenter	Custom (cloud)	Custom (On-premises VMs)	Custom (Bare Metal)
Fedora (Single Node)Â						âœ”
Gardener		âœ”		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			
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”	
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”				

Providers	Managed	Turnkey cloud	On-prem datacenter	Custom (cloud)	Custom (On-premises VMs)	Custom (Bare Metal)
Supergiant		âœ”				
SUSE		âœ”				
SysEleven	âœ”					
VEXXHOST	âœ”	âœ”				
VMware	VMware Cloud PKS	VMware Enterprise PKS	VMware Enterprise PKS	VMware Essential PKS		VMware Essential PKS

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 June 21, 2019 at 6:48 PM PST by Fix typo (#15046) (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

v1.15 Release Notes

v1.15.0

Documentation

Downloads for v1.15.0

filename	sha512 hash
kubernetes.tar.gz	cb03adc8bee094b93652a19cb77ca4b7b0b2ec201cf9c09958128eb93b4c71
kubernetes-src.tar.gz	a682c88539b46741f6f3b2fa27017d52e88149e0cf0fe49c5a84ff30018cfa

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	bb14d564f5c2e4da964f6dcaf4026ac7371b35ecf5d651d226fb7cc0c3f1
kubernetes-client-darwin-amd64.tar.gz	8c218437588d960f6782576038bc63af5623e66291d37029653d4bdbba5e
kubernetes-client-linux-386.tar.gz	6a17e7215d0eb9ca18d4b55ee179a13f1f111ac995aad12bf2613b9dbee1
kubernetes-client-linux-amd64.tar.gz	0906a8f7de1e5c5efd124385fdee376893733f343d3e8113e4f0f02dfae6
kubernetes-client-linux-arm.tar.gz	1d3418665b4998d6fff1c137424eb60302129098321052d7c5cee5a0e2a5
kubernetes-client-linux-arm64.tar.gz	986d6bec386b3bb427e49cd7e41390c7dc5361da4f2f7fc2a823507f8357
kubernetes-client-linux-ppc64le.tar.gz	81315af33bc21f9f8808b125e1f4c7a1f797c70f01098fe1fe8dba73d05d
kubernetes-client-linux-s390x.tar.gz	485978a24ba97a2a2cac162a6984d4b5c32dbe95882cf18d2fd2bf74477f
kubernetes-client-windows-386.tar.gz	9a1b5d0f6fbfc85269e9bd7e08be95eeb9a11f43ea38325b8a736e768f3e
kubernetes-client-windows-amd64.tar.gz	f2f0221c7d364e3e71b2d9747628298422441c43b731d58c14d7a0ed292e

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	`fee0200887c7616e3706394b0540b471ad24d57bb587a3a7154adfcd212c7a25`
kubernetes-server-linux-arm.tar.gz	`2d329ec0e231dbd4ec750317fc45fb8a966b9a81b45f1af0dde3ca0d1ae66a5a`
kubernetes-server-linux-arm64.tar.gz	`0fb64d934d82c17eee15e1f97fc5eeeb4af6e042c30abe41a4d245cde1d9d81e`
kubernetes-server-linux-ppc64le.tar.gz	`5cac4b5951692921389db280ec587037eb3bb7ec4ccf08599ecee2fa39c2a598`
kubernetes-server-linux-s390x.tar.gz	`39a33f0bb0e06b34779d741e6758b6f7d385e0b933ab799b233e3d4e317f76b5`

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	`73abf50e44319763be3124891a1db36d7f7b38124854a1f223ebd91dce8e848a`
kubernetes-node-linux-arm.tar.gz	`b7ddb82efa39ba5fce5b4124d83279357397a1eb60be24aa19ccbd8263e5e614`
kubernetes-node-linux-arm64.tar.gz	`458f20f7e9ca2ebddef8738de6a2baa8b8d958b22a935e4d7ac099b07bed91fe`
kubernetes-node-linux-ppc64le.tar.gz	`d4d5bfe9b9d56495b00322f62aed0f76029d774bff5004d68e85a0db4fb3b4ce`
kubernetes-node-linux-s390x.tar.gz	`b967034c8db871a7f503407d5a096fcd6811771c9a294747b0a028659af582fb`
kubernetes-node-windows-amd64.tar.gz	`dd021d8f2a3d9ddff6e88bce678c28cc0f38165a5d7a388df952d900dcfd1dca`

Kubernetes v1.15 Release Notes

1.15 What's New

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

Kubernetes 1.15 consists of **25 enhancements**: 2 moving to stable, 13 in beta, and 10 in alpha. The main themes of this release are:

Continuous Improvement

- Project sustainability is not just about features. Many SIGs have been working on improving test coverage, ensuring the basics stay reliable, and stability of the core feature set and working on maturing existing features and cleaning up the backlog.

Extensibility

- The community has been asking for continuing support of extensibility, so this cycle features more work around CRDs and API Machinery. Most of the enhancements in this cycle were from SIG API Machinery and related areas.

Extensibility around core Kubernetes APIs

CustomResourceDefinitions Pruning

To enforce both data consistency and security, Kubernetes performs pruning, or the automatic removal of unknown fields in objects sent to a Kubernetes API. An "unknown" field is one that is not specified in the OpenAPI validation schema. This behavior is already in place for native resources and ensures only data structures specified by the CRD developer are persisted to etcd. It will be available as a beta feature in Kubernetes 1.15.

Pruning is activated by setting `spec.preserveUnknownFields: false` in the CustomResourceDefinition. A future apiextensions.k8s.io/v1 variant of CRDs will enforce pruning.

Pruning requires that CRD developer provides complete, structural validation schemas, either at the top-level or for all versions of the CRD.

CustomResourceDefinition Defaulting

CustomResourceDefinitions also have new support for defaulting, with defaults specified using the `default` keyword in the OpenAPI validation schema. Defaults are set for unspecified fields in an object sent to the API, and when reading from etcd.

Defaulting will be available as alpha in Kubernetes 1.15 and requires structural schemas.

CustomResourceDefinition OpenAPI Publishing

OpenAPI specs for native types have long been served at /openapi/v2, and they are consumed by a number of components, notably kubectl client-side validation, kubectl explain and OpenAPI based client generators.

With Kubernetes 1.15 as beta, OpenAPI schemas are also published for CRDs, as long as their schemas are structural.

These changes are reflected in the following Kubernetes enhancements: (#383), (#575), (#492), (#598), (#692), (#95), (#995), (#956)

Cluster Lifecycle Stability and Usability Improvements

Work on making Kubernetes installation, upgrade and configuration even more robust has been a major focus for this cycle for SIG Cluster Lifecycle (see the May 6, 2019 Community Update). Bug fixes across bare metal tooling and production-ready user stories, such as the high availability use cases have been given priority for 1.15.

kubeadm, the cluster lifecycle building block, continues to receive features and stability work required for bootstrapping production clusters efficiently. kubeadm has promoted high availability (HA) capability to beta, allowing users to use the familiar `kubeadm init` and `kubeadm join` commands to configure and deploy an HA control plane. An entire new test suite has been created specifically for ensuring these features will stay stable over time.

Certificate management has become more robust in 1.15, with kubeadm now seamlessly rotating all your certificates (on upgrades) before they expire. Check the kubeadm documentation for information on how to manage your certificates.

The kubeadm configuration file API is moving from v1beta1 to v1beta2 in 1.15.

These changes are reflected in the following Kubernetes enhancements: (#357), (#970)

Continued improvement of CSI

In Kubernetes v1.15, SIG Storage continued work to enable migration of in-tree volume plugins to the Container Storage Interface (CSI). SIG Storage worked on bringing CSI to feature parity with in-tree functionality, including functionality like resizing, inline volumes, and more. SIG Storage introduces new alpha functionality in CSI that doesn't exist in the Kubernetes Storage subsystem yet, like volume cloning.

Volume cloning enables users to specify another PVC as a "DataSource" when provisioning a new volume. If the underlying storage system supports this functionality and implements the "CLONE_VOLUME" capability in its CSI driver, then the new volume becomes a clone of the source volume.

These changes are reflected in the following Kubernetes enhancements: (#625)

Additional Notable Feature Updates

- Support for go modules in Kubernetes Core.
- Continued preparation for cloud provider extraction and code organization. The cloud provider code has been moved to kubernetes/legacy-cloud-providers for easier removal later and external consumption.
- Kubectl get and describe now works with extensions
- Nodes now support third party monitoring plugins.
- A new Scheduling Framework for schedule plugins is now Alpha.
- ExecutionHook API designed to trigger hook commands in containers is now Alpha.
- Continued deprecation of extensions/v1beta1, apps/v1beta1, and apps/v1beta2 APIs; these extensions will be retired in 1.16!

Check the release notes website for the complete changelog of notable features and fixes.

Known Issues

- Concurrently joining control-plane nodes does not work as expected in kubeadm 1.15.0. The feature was planned for release in 1.15.0, but a fix may come in a follow up patch release.

- Using `--log-file` is known to be problematic in 1.15. This presents as things being logged multiple times to the same file. The behaviour and details of this issue, as well as some preliminary attempts at fixing it are documented here

Urgent Upgrade Notes

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

API Machinery

- `k8s.io/kubernetes` and published components (such as `k8s.io/client-go` and `k8s.io/api`) now contain go module files including dependency version information. See go-modules for details on consuming `k8s.io/client-go` using go modules. (#74877, @liggitt)

Apps

- Hyperkube short aliases have been removed from source code, because hyperkube docker image currently creates these aliases. (#76953, @Rand01ph)

Auth

- The Rancher credential provider has now been removed. This only affects you if you are using the downstream Rancher distro. (#77099, @dims)

AWS

- The `system:aws-cloud-provider` cluster role, deprecated in v1.13, is no longer auto-created. Deployments using the AWS cloud provider should grant required permissions to the `aws-cloud-provider` service account in the `kube-system` namespace as part of deployment. (#66635, @wgliang)

Azure

- Kubelet can now run without identity on Azure. A sample cloud provider configuration is: `{"vmType": "vmss", "useInstanceMetadata": true, "subscriptionId": "<subscriptionId>"}` (#77906, @feiskyer)
- Multiple Kubernetes clusters can now share the same resource group
  - When upgrading from previous releases, issues will arise with public IPs if multiple clusters share the same resource group. To solve these problems, make the following changes to the cluster: Recreate the relevant LoadBalancer services, or add a new tag â€˜kubernetes-cluster-name: ' manually for existing public IPs. Configure each cluster with a different cluster name using `kube-controller-manager --cluster-name=<cluster-name>` (#77630, @feiskyer)

- The cloud config for Azure cloud provider can now be initialized from Kubernetes secret azure-cloud-provider in kube-system namespace
  - the secret is a serialized version of `azure.json` file with key cloud-config. And the secret name is azure-cloud-provider.
  - A new option cloudConfigType has been added to the cloud-config file. Supported values are: `file`, `secret` and `merge` (`merge` is the default value).
  - To allow Azure cloud provider to read secrets, the RBAC rules should be configured.

CLI

- `kubectl scale job`, deprecated since 1.10, has been removed. (#78445, @soltysh)
- The deprecated `--pod/-p` flag for `kubectl exec` has been removed. The flag has been marked as deprecated since k8s version v1.12. (#76713, @prksu)

Lifecycle

- Support for deprecated old kubeadm v1alpha3 config has been totally removed. (#75179, @rosti)
- kube-up.sh no longer supports "centos" and "local" providers. (#76711, @dims)

Network

- The deprecated flag `--conntrack-max` has been removed from kube-proxy. Users of this flag should switch to `--conntrack-min` and `--conntrack-max-per-core` instead. (#78399, @rikatz)
- The deprecated kube-proxy flag `--cleanup-iptables` has been removed. (#78344, @aramase)

Node

- The deprecated kubelet security controls `AllowPrivileged`, `HostNetworkSources`, `HostPIDSources`, and `HostIPCSources` have been removed. Enforcement of these restrictions should be done through admission control (such as `PodSecurityPolicy`) instead. (#77820, @dims)
- The deprecated Kubelet flag `--allow-privileged` has been removed. Remove any use of the flag from your kubelet scripts or manifests. (#77820, @dims)
- The kubelet now only collects cgroups metrics for the node, container runtime, kubelet, pods, and containers. (#72787, @dashpole)

Storage

- The `Node.Status.Volumes.Attached.DevicePath` field is now unset for CSI volumes. You must update any external controllers that depend on this field. (#75799, @msau42)
- CSI alpha CRDs have been removed (#75747, @msau42)
- The `StorageObjectInUseProtection` admission plugin is enabled by default, so the default enabled admission plugins are now `NamespaceLifecycle,LimitRanger,ServiceAccount,PersistentVolumeLabel,DefaultStorageClass,DefaultTolerationSeconds,MutatingAdmissionWebhook,ValidatingAdmissionWebhook,ResourceQuota,StorageObjectInUseProtection`. Please note that if you previously had not set the `--admission-control` flag, your cluster behavior may change (to be more standard). (#74610, @oomichi)

Deprecations and Removals

- 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.
  - The `--pod`/`-p` flag for `kubectl exec`, deprecated since 1.12, has been removed.
  - `kubectl scale job`, deprecated since 1.10, has been removed. (#78445, @soltysh)
- kubelet
  - The `beta.kubernetes.io/os` and `beta.kubernetes.io/arch` labels, deprecated since v1.14, are targeted for removal in v1.18.
  - The `--containerized` flag, deprecated since v1.14, will be removed in a future release.
  - cAdvisor json endpoints have been deprecated. (#78504, @dashpole)
- kube-apiserver
  - The `--enable-logs-handler` flag and log-serving functionality is deprecated, and scheduled to be removed in v1.19. (#77611, @rohitsardesai83)
- kube-proxy
  - The deprecated `--cleanup-iptables` has been removed,. (#78344, @aramase)
- API
  - Ingress resources will no longer be served from `extensions/v1beta1` in v1.19. 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.
  - NetworkPolicy resources will no longer be served from `extensions/v1beta1` in v1.16. Migrate use to the `networking.k8s.io/v1` API, available since v1.8. Existing persisted data can be retrieved via the `networking.k8s.io/v1` API.
  - PodSecurityPolicy resources will no longer be served from `extensions/v1beta1` in v1.16. Migrate to the `policy/v1beta1` API, available since v1.10. Existing persisted data can be retrieved via the `policy/v1beta1` API.
  - DaemonSet, Deployment, and ReplicaSet resources will no longer be served from `extensions/v1beta1`, `apps/v1beta1`, or `apps/v1beta2` in v1.16. Migrate to the `apps/v1` API, available since v1.9. Existing persisted data can be retrieved via the `apps/v1` API.
  - PriorityClass resources will no longer be served from `scheduling.k8s.io/v1beta1` and `scheduling.k8s.io/v1alpha1` in v1.17. Migrate use 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)

- kubeadm
    - The `kubeadm upgrade node config` and `kubeadm upgrade node experimental-control-plane` commands are deprecated in favor of `kubeadm upgrade node`, and will be removed in a future release. (#78408, @fabriziopandini)
    - The flag `--experimental-control-plane` is now deprecated in favor of `--control-plane`. The flag `--experimental-upload-certs` is now deprecated in favor of `--upload-certs` (#78452, @fabriziopandini)
    - `kubeadm config upload` has been deprecated, as its replacement is now graduated. Please use `kubeadm init phase upload-config` instead. (#77946, @Klaven)

- The following features are now GA, and the associated feature gates are deprecated and will be removed in v1.17:
    - `GCERegionalPersistentDisk`

Metrics Changes

Added metrics

- The metric `kube_proxy_sync_proxy_rules_last_timestamp_seconds` is now available, indicating the last time that kube-proxy successfully applied proxying rules. (#74027, @squeed)
- `process_start_time_seconds` has been added to kubelet's â€˜/metrics/probes' endpoint (#77975, @logicalhan)
- Scheduler: added metrics to record the number of pending pods in different queues (#75501, @Huang-Wei)
- Exposed CSI volume stats via kubelet volume metrics (#76188, @humblec)
- Added a new `storage_operation_status_count` metric for kube-controller-manager and kubelet to count success and error statues. (#75750, @msau42)

Deprecated/changed metrics

- kubelet probe metrics are now of the counter type rather than the gauge type, and the `prober_probe_result` has been replaced by `prober_probe_total`. (#76074, @danielqsj)
- The `transformer_failures_total` metric is deprecated in favor of `transformation_operation_total`. The old metric will continue to be populated but will be removed in a future release. (#70715, @immutableT)

- Introducing new semantic for metric `volume_operation_total_seconds` to be the end to end latency of volume provisioning/deletion. Existing metric "storage_operation_duration_seconds" will remain untouched, however it is exposed to the following potential issues:

    1. For volumes provisioned/deleted via external provisioner/deleter, `storage_operation_duration_seconds` will NOT wait for the external operation to be done before reporting latency metric (effectively close to 0). This will be fixed by using `volume_operation_total_seconds` instead

2. if there's a transient error happened during "provisioning/deletion", i.e., a volume is still in-use while a deleteVolume has been called, original `storage_operation_duration_seconds` will NOT wait until a volume has been finally deleted before reporting an inaccurate latency metric. The newly implemented metric `volume_operation_total_seconds`, however, waits until a provisioning/deletion operation has been fully executed.

   Potential impacts: If an SLO/alert has been defined based on `volume_operation_total_seconds`, it might get violated because of the more accurate metric might be significantly larger than previously reported. The metric is defined to be a histogram and the new semantic could change the distribution. (#78061, @yuxiangqian)

- Implement the scheduling framework with `Reserve`, `Prebind`, `Permit`, `Post-bind`, `Queue sort` and `Unreserve` extension points. (#77567, @wgliang) (#77559, @ahg-g) (#77529, @draveness) (#77598, @danielqsj) (#77501, @JieJhih) (#77457, @danielqsj)

- Replaced *_admission_latencies_milliseconds_summary and *_admission_latencies_milliseconds metrics because they were reporting seconds rather than milliseconds. They were also subject to multiple naming guideline violations (units should be in base units and "duration" is the best practice labelling to measure the time a request takes). Please convert to use *_admission_duration_seconds and *_admission_duration_seconds_summary, as these now report the unit as described, and follow the instrumentation best practices. (#75279, @danielqsj)

- Fixed admission metrics histogram bucket sizes to cover 25ms to ~2.5 seconds. (#78608, @jpbetz)

- Fixed incorrect prometheus azure metrics. (#77722, @andyzhangx)

- `kubectl scale job`, deprecated since 1.10, has been removed. (#78445, @soltysh)

Notable Features

Stable

- You can now create a non-preempting Pod priority. If set on a class, the pod will continue to be prioritized above queued pods of a lesser class, but will not preempt running pods. (#74614, @denkensk)

- Third party device monitoring is now enabled by default (KubeletPodResources). (#77274, @RenaudWasTaken)

- The kube-apiserver's `watch` can now be enabled for events using the `--watch-cache-sizes` flag. (#74321, @yastij)

Beta

- Admission webhooks can now register for a single version of a resource (for example, `apps/v1 deployments`) and be called when any other version of that resource is modified (for example `extensions/v1beta1 deployments`). This allows new versions of a resource to be handled by admission webhooks without needing to update every webhook to understand the new version. See the API documentation for the `matchPolicy: Equivalent` option in MutatingWebhookConfiguration and ValidatingWebhookConfiguration types. (#78135, @liggitt)

- The CustomResourcePublishOpenAPI feature is now beta and enabled by default. CustomResourceDefinitions with structural schemas now publish schemas in the OpenAPI document served at `/openapi/v2`. CustomResourceDefinitions with non-structural schemas have a `NonStructuralSchema` condition added with details about what needs to be corrected in the validation schema. (#77825, @roycaihw)
- Online volume expansion (ExpandInUsePersistentVolumes) is now a beta feature. As such, it is enabled by default. (#77755, @gnufied)
- The `SupportNodePidsLimit` feature is now beta, and enabled by default. It is no longer necessary to set the feature gate `SupportNodePidsLimit=true`. (#76221, @RobertKrawitz)
- kubeadm now includes the ability to specify certificate encryption and decryption keys for the upload and download certificate phases as part of the new v1beta2 kubeadm config format. (#77012, @rosti)
- You can now use kubeadm's `InitConfiguration` and `JoinConfiguration` to define which preflight errors will be ignored. (#75499, @marccarre)
- CustomResourcesDefinition conversion via Web Hooks is promoted to beta. Note that you must set `spec.preserveUnknownFields` to `false`. (#78426, @sttts)
- Group Managed Service Account support has moved to a new API for beta. Special annotations for Windows GMSA support have been deprecated. (#75459, @wk8)
- The `storageVersionHash` feature is now beta. `StorageVersionHash` is a field in the discovery document of each resource. It enables clients to detect whether the storage version of that resource has changed. Its value must be treated as opaque by clients. Only equality comparison on the value is valid. (#78325, @caesarxuchao)
- Ingress objects are now persisted in etcd using the `networking.k8s.io/v1beta1` version (#77139, @cmluciano)
- NodeLocal DNSCache graduating to beta. (#77887, @prameshj)

Alpha

- kubelet now allows the use of XFS quotas (on XFS and suitably configured ext4fs filesystems) to monitor storage consumption for ephemeral storage. This method of monitoring consumption, which is currently available only for `emptyDir` volumes, is faster and more accurate than the old method of walking the filesystem tree. Note that it does not enforce limits, it only monitors consumption. To utilize this functionality, set the feature gate `LocalStorageCapacityIsolationFSQuotaMonitoring=true`. For ext4fs filesystems, create the filesystem with `mkfs.ext4 -O project <block_device>` and run `tune2fs -Q prjquotablock device; XFS filesystems need no additional preparation. The filesystem must be mounted with optionprojectin/etc/fstab. If the primary partition is the root filesystem, addrootflags=pquota` to the GRUB config file. (#66928, @RobertKrawitz)
- Finalizer Protection for Service LoadBalancers (ServiceLoadBalancerFinalizer) has been added as an Alpha feature, which is disabled by default. This feature ensures the Service resource is not fully deleted until the correlating load balancer resources are deleted. (#78262, @MrHohn)
- Inline CSI ephemeral volumes can now be controlled with PodSecurityPolicy when the CSIInlineVolume alpha feature is enabled. (#76915, @vladimirvivien)
- Kubernetes now includes an alpha field, `AllowWatchBookmarks`, in ListOptions for requesting the watching of bookmarks from apiserver. The implementation in apiserver is hidden behind the feature gate `WatchBookmark`. (#74074, @wojtek-t)

Staging Repositories

- The CRI API is now available in the `k8s.io/cri-api` staging repository. (#75531, @dims)
- Support for the Azure File plugin has been added to `csi-translation-lib` (CSIMigrationAzureFile). (#78356, @andyzhangx)
- Added support for Azure Disk plugin to csi-translation-lib (CSIMigrationAzureDisk) (#78330, @andyzhangx)

CLI Improvements

- Added `kubeadm upgrade node`. This command can be used to upgrade both secondary control-plane nodes and worker nodes. The `kubeadm upgrade node config` and `kubeadm upgrade node experimental-control-plane` commands are now deprecated. (#78408, @fabriziopandini)
- The `kubectl top` command now includes a `--sort-by` option to sort by `memory` or `cpu`. (#75920, @artmello)
- `kubectl rollout restart` now works for DaemonSets and StatefulSets. (#77423, @apelisse)
- `kubectl get --watch=true` now prints custom resource definitions with custom print columns. (#76161, @liggitt)
- Added `kubeadm alpha certs certificate-key` command to generate secure random key to use on `kubeadm init --experimental-upload-certs` (#77848, @yagonobre)
- Kubernetes now supports printing the `volumeMode` using `kubectl get pv/pvc -o wide` (#76646, @cwdsuzhou)
- Created a new `kubectl rollout restart` command that does a rolling restart of a deployment. (#76062, @apelisse)
- `kubectl exec` now allows using the resource name to select a matching pod and `--pod-running-timeout` flag to wait till at least one pod is running. (#73664, @prksu)
- `kubeadm alpha certs renew` and `kubeadm upgrade` now supports renewal of certificates embedded in KubeConfig files managed by kubeadm; this does not apply to certificates signed by external CAs. (#77180, @fabriziopandini)
- Kubeadm: a new command `kubeadm alpha certs check-expiration` was created in order to help users in managing expiration for local PKI certificates (#77863, @fabriziopandini)

Misc

- Service account controller clients to now use the TokenRequest API, and tokens are periodically rotated. (#72179, @WanLinghao)
- Added `ListPager.EachListItem` utility function to client-go to enable incremental processing of chunked list responses (#75849, @jpbetz)
- Object count quota is now supported for namespaced custom resources using the `count/<resource>.<group>` syntax. (#72384, @zhouhaibing089)
- Added completed job status in Cron Job event. (#75712, @danielqsj)
- Pod disruption budgets can now be updated and patched. (#69867, @davidmccormick)
- Add CRD spec.preserveUnknownFields boolean, defaulting to true in v1beta1 and to false in v1 CRDs. If false, fields not specified in the validation schema will be removed when sent to the API server or when read from etcd. (#77333, @sttts)
- Added RuntimeClass restrictions and defaulting to PodSecurityPolicy. (#73795, @tallclair)

- Kubelet plugin registration now has retry and exponential backoff logic for when registration of plugins (such as CSI or device plugin) fail. (#73891, @taragu)
- proxy/transport now supports Content-Encoding: deflate (#76551, @JieJhih)
- Admission webhooks are now properly called for `scale` and `deployments/rollback` subresources. (#76849, @liggitt)

API Changes

- CRDs get support for x-kubernetes-int-or-string to allow faithful representation of IntOrString types in CustomResources.(#78815, @sttts)
- Introduced the `v1beta2` config format to kubeadm. (#76710, @rosti)
- Resource list requests for `PartialObjectMetadata` now correctly return list metadata like the resourceVersion and the continue token. (#75971, @smarterclayton)
- Added a condition `NonStructuralSchema` to `CustomResourceDefinition` listing Structural Schema violations as defined in the KEP. CRD authors should update their validation schemas to be structural in order to participate in future CRD features. (#77207, @sttts)
- Promoted meta.k8s.io/v1beta1 Table and PartialObjectMetadata to v1. (#77136, @smarterclayton)
- Introduced the flag `--ipvs-strict-arp` to configure stricter ARP sysctls, defaulting to false to preserve existing behaviors. This was enabled by default in 1.13.0, which impacted a few CNI plugins. (#75295, @lbernail)
- CRD validation schemas should not specify `metadata` fields other than `name` and `generateName`. A schema will not be considered structural (and therefore ready for future features) if `metadata` is specified in any other way. (#77653, @sttts)

Other notable changes

API Machinery

- Added port configuration to Admission webhook configuration service reference.
- Added port configuration to AuditSink webhook configuration service reference.
- Added port configuration to CRD Conversion webhook configuration service reference.
- Added port configuration to kube-aggregator service reference. (#74855, @mbohlool)
- Implemented deduplication logic for v1beta1.Event API (#65782, @yastij)
- Added `objectSelector` to admission webhook configurations. `objectSelector` is evaluated the oldObject and newObject that would be sent to the webhook, and is considered to match if either object matches the selector. A null object (oldObject in the case of create, or newObject in the case of delete) or an object that cannot have labels (like a DeploymentRollback or a PodProxyOptions object) is not considered to match. Use the object selector only if the webhook is opt-in, because end users may skip the admission webhook by setting the labels. (#78505, @caesarxuchao)
- Watch will now support converting response objects into Table or PartialObjectMetadata forms. (#71548, @smarterclayton)
- In CRD webhook conversion, Kubernetes will now ignore changes to metadata other than for labels and annotations. (#77743, @sttts)
- Added ListMeta.RemainingItemCount. When responding to a LIST request, if the server has more data available, and if the request does not contain label selectors or field selectors, the server sets the ListOptions.RemainingItemCount to the number of remaining objects. (#75993, @caesarxuchao)

- Clients may now request that API objects are converted to the `v1.Table` and `v1.PartialObjectMetadata` forms for generic access to objects. (#77448, @smarterclayton)

- Fixed a spurious error where update requests to the status subresource of multi-version custom resources would complain about an incorrect API version. (#78713, @liggitt)
- Fixed a bug in apiserver storage that could cause just-added finalizers to be ignored immediately following a delete request, leading to premature deletion. (#77619, @caesarxuchao)
- API requests rejected by admission webhooks which specify an http status code < 400 are now assigned a 400 status code. (#77022, @liggitt)
- Fixed a transient error API requests for custom resources could encounter while changes to the CustomResourceDefinition were being applied. (#77816, @liggitt) @smarterclayton)
- Added name validation for dynamic client methods in client-go (#75072, @lblackstone)
- CustomResourceDefinition with invalid regular expression in the pattern field of OpenAPI v3 validation schemas are no longer considered structural. (#78453, @sttts)
- API paging is now enabled by default in k8s.io/apiserver recommended options, and in k8s.io/sample-apiserver (#77278, @liggitt)
- Increased verbose level for local openapi aggregation logs to avoid flooding the log during normal operation (#75781, @roycaihw)
- k8s.io/client-go/dynamic/dynamicinformer.NewFilteredDynamicSharedInformerFactory now honours the `namespace` argument. (#77945, @michaelfig)
- client-go and kubectl no longer write cached discovery files with world-accessible file permissions. (#77874, @yuchengwu)
- Fixed an error with stuck informers when an etcd watch receives update or delete events with missing data. (#76675, @ryanmcnamara)
- `DelayingQueue.ShutDown()` can now be invoked multiple times without causing a closed channel panic. (#77170, @smarterclayton)
- When specifying an invalid value for a label, it was not always clear which label the value was specified for. Starting with this release, the label's key is included in such error messages, which makes debugging easier. (#77144, @kenegozi)
- Fixed a regression error when proxying responses from aggregated API servers, which could cause watch requests to hang until the first event was received. (#75887, @liggitt)
- Fixed a bug where dry-run is not honored for pod/eviction sub-resource. (#76969, @apelisse)
- DeleteOptions parameters for deletecollection endpoints are now published in the OpenAPI spec. (#77843, @roycaihw)
- Active watches of custom resources now terminate properly if the CRD is modified. (#78029, @liggitt)
- Fixed a potential deadlock in the resource quota controller. Enabled recording partial usage info for quota objects specifying multiple resources, when only some of the resources' usage can be determined. (#74747, @liggitt)

- Updates that remove remaining `metadata.finalizers` from an object that is pending deletion (non-nil metadata.deletionTimestamp) and has no graceful deletion pending (nil or 0 metadata.deletionGracePeriodSeconds) now results in immediate deletion of the object. (#77952, @liggitt)

- client-go: The `rest.AnonymousClientConfig(*rest.Config)` `*rest.Config` helper method no longer copies custom `Transport` and `WrapTrans port` fields, because those can be used to inject user credentials. (#75771, @liggitt)

- Validating admission webhooks are now properly called for CREATE operations on the following resources: pods/binding, pods/eviction, bindings (#76910, @liggitt)

- Removed the function Parallelize, please convert to use the function ParallelizeUntil. (#76595, @danielqsj)

Apps

- Users can now specify a DataSource/Kind of type `PersistentVolumeClaim` in their PVC spec. This can then be detected by the external csi-provisioner and plugins if capable. (#76913, @j-griffith)
- Fixed bug in DaemonSetController causing it to stop processing some DaemonSets for 5 minutes after node removal. (#76060, @krzysztof-jastrzebski)
- StatefulSet controllers no longer force a resync every 30 seconds when nothing has changed. (#75622, @jonsabo)
- Enhanced the daemonset sync logic to avoid a problem where pods are thought to be unavailable when the controller's clock is slower than the node's clock. (#77208, @DaiHao)
- Fixed a bug that caused a DaemonSet rolling update to hang when its pod gets stuck at terminating. (#77773, @DaiHao)
- Route controller now respects rate limiting to the cloud provider on deletion; previously it was only for create. (#78581, @andrewsykim)
- Removed extra pod creation expectations when daemonset fails to create pods in batches. (#74856, @draveness)
- Resolved spurious rollouts of workload controllers when upgrading the API server, due to incorrect defaulting of an alpha procMount field in pods. (#78885, @liggitt)

Auth

- Fixed OpenID Connect (OIDC) token refresh when the client secret contains a special character. (#76914, @tsuna)
- Improved `kubectl auth can-i` command by warning users when they try to access a resource out of scope. (#76014, @WanLinghao)
- Validating admission webhooks are now properly called for CREATE operations on the following resources: tokenreviews, subjectaccessreviews, localsubjectaccessreviews, selfsubjectaccessreviews, selfsubjectrulesreviews (#76959, @sbezverk)

Autoscaling

- Horizontal Pod Autoscaling can now scale targets up even when one or more metrics are invalid/unavailable, as long as one metric indicates a scale up should occur. (#78503, @gjtempleton)

AWS

- Kubernetes will now use the zone from the node for topology aware aws-ebs volume creation to reduce unnecessary cloud provider calls. (#78276, @zhan849)
- Kubernetes now supports configure accessLogs for AWS NLB. (#78497, @M00nF1sh)
- Kubernetes now supports update LoadBalancerSourceRanges for AWS NLB(#74692, @M00nF1sh)
- Kubernetes now supports configure TLS termination for AWS NLB(#74910, @M00nF1sh)
- Kubernetes will now consume the AWS region list from the AWS SDK instead of a hard-coded list in the cloud provider. (#75990, @mcrute)
- Limit use of tags when calling EC2 API to prevent API throttling for very large clusters. (#76749, @mcrute)
- The AWS credential provider can now obtain ECR credentials even without the AWS cloud provider or being on an EC2 instance. Additionally, AWS credential provider caching has been improved to honor the ECR credential timeout. (#75587, @tiffanyfay)

Azure

- Kubernetes now supports specifying the Resource Group of the Route Table when updating the Pod network route on Azure. (#75580, @suker200)
- Kubernetes now uses instance-level update APIs for Azure VMSS loadbalancer operations. (#76656, @feiskyer)
- Users can now specify azure file share name in the azure file plugin, making it possible to use existing shares or specify a new share name. (#76988, @andyzhangx)
- You can now run kubelet with no Azure identity. A sample cloud provider configuration is: `{"vmType": "vmss", "useInstanceMetadata": true, "subscriptionId": "<subscriptionId>"}` (#77906, @feiskyer)
- Fixed some service tags not supported issues for Azure LoadBalancer service. (#77719, @feiskyer)
- Fixed an issue where `pull image` fails from a cross-subscription Azure Container Registry when using MSI to authenticate. (#77245, @norshtein)
- Azure cloud provider can now be configured by Kubernetes secrets and a new option `cloudConfigType` has been introduced. Candidate values are `file`, `secret` or `merge` (default is `merge`). Note that the secret is a serialized version of `azure.json` file with key cloud-config. And the secret name is azure-cloud-provider in kube-system namespace. (#78242, @feiskyer)

CLI

- Fixed `kubectl exec` usage string to correctly reflect flag placement. (#77589, @soltysh)
- Fixed `kubectl describe cronjobs` error of `Successful Job History Limit`. (#77347, @danielqsj)
- In the `kubectl describe` output, the fields with names containing special characters are now displayed as-is without any pretty formatting, avoiding awkward outputs. (#75483, @gsadhani)
- Fixed incorrect handling by kubectl of custom resources whose Kind is "Status". (#77368, @liggitt)
- Report cp errors consistently, providing full message whether copying to or from a pod. (#77010, @soltysh)
- Preserved existing namespace information in manifests when running `set ... --local` commands. (#77267, @liggitt)

- Support for parsing more v1.Taint forms has been added. For example, `key:effect`, `key=:effect-` are now accepted. (#74159, @dlipovetsky)

Cloud Provider

- The GCE-only flag `cloud-provider-gce-lb-src-cidrs` is now optional for external cloud providers. (#76627, @timoreimann)
- Fixed a bug where cloud-controller-manager initializes nodes multiple times. (#75405, @tghartland)

Cluster Lifecycle

- `kubeadm upgrade` now renews all the certificates used by a component before upgrading the component itself, with the exception of certificates signed by external CAs. User can eventually opt-out of certificate renewal during upgrades by setting the new flag `--certificate-renewal` to false. (#76862, @fabriziopandini)
- kubeadm still generates RSA keys when deploying a node, but also accepts ECDSA keys if they already exist in the directory specified in the `--cert-dir` option. (#76390, @rojkov)
- kubeadm now implements CRI detection for Windows worker nodes (#78053, @ksubrmnn)

- Added `--image-repository` flag to `kubeadm config images`. (#75866, @jmkeyes)

- kubeadm: The kubeadm reset command has now been exposed as phases. (#77847, @yagonobre)

- kubeadm: Improved resiliency when it comes to updating the `kubeadm-config` configmap upon new control plane joins or resets. This allows for safe multiple control plane joins and/or resets. (#76821, @ereslibre)

- kubeadm: Bumped the minimum supported Docker version to 1.13.1 (#77051, @chenzhiwei)

- Reverted the CoreDNS version to 1.3.1 for kubeadm (#78545, @neolit123)

- kubeadm: Fixed the machine readability of `kubeadm token create --print-join-command` (#75487, @displague)

- `kubeadm alpha certs renew --csr-only` now reads the current certificates as the authoritative source for certificates attributes (same as kubeadm alpha certs renew). (#77780, @fabriziopandini)

- kubeadm: You can now delete multiple bootstrap tokens at once. (#75646, @bart0sh)

- util/initsystem: Added support for the OpenRC init system (#73101, @oz123)

- Default TTL for DNS records in kubernetes zone has been changed from 5s to 30s to keep consistent with old dnsmasq based kube-dns. The TTL can be customized with command `kubectl edit -n kube-system configmap/coredns`. (#76238, @Dieken)

- Communication between the etcd server and kube-apiserver on master is now overridden to use HTTPS instead of HTTP when mTLS is enabled in GCE. (#74690, @wenjiaswe)

GCP

- [stackdriver addon] Bumped prometheus-to-sd to v0.5.0 to pick up security fixes. [fluentd-gcp addon] Bumped fluentd-gcp-scaler to v0.5.1 to pick up security fixes. [fluentd-gcp addon] Bumped event-exporter to v0.2.4 to pick up security fixes. [fluentd-gcp addon] Bumped prometheus-to-sd to v0.5.0 to pick up security fixes. [metatada-proxy addon] Bumped prometheus-to-sd v0.5.0 to pick up security fixes. (#75362, @serathius)
- [fluentd-gcp addon] Bump fluentd-gcp-scaler to v0.5.2 to pick up security fixes. (#76762, @serathius)
- The GCERegionalPersistentDisk feature gate (GA in 1.13) can no longer be disabled. The feature gate will be removed in v1.17. (#77412, @liggitt)
- GCE/Windows: When the service cannot be stopped Stackdriver logging processes are now force killed (#77378, @yujuhong)
- Reduced GCE log rotation check from 1 hour to every 5 minutes. Rotation policy is unchanged (new day starts, log file size > 100MB). (#76352, @jpbetz)
- GCE/Windows: disabled stackdriver logging agent to prevent node startup failures (#76099, @yujuhong)
- API servers using the default Google Compute Engine bootstrapping scripts will have their insecure port (`:8080`) disabled by default. To enable the insecure port, set `ENABLE_APISERVER_INSECURE_PORT=true` in kube-env or as an environment variable. (#77447, @dekkagaijin)
- Fixed a NPD bug on GCI, so that it disables glog writing to files for log-counter. (#76211, @wangzhen127)
- Windows nodes on GCE now have the Windows firewall enabled by default. (#78507, @pjh)
- Added `CNI_VERSION` and `CNI_SHA1` environment variables in `kube-up.sh` to configure CNI versions on GCE. (#76353, @Random-Liu)
- GCE clusters will include some IP ranges that are not used on the public Internet in the list of non-masq IPs. Bumped ip-masq-agent version to v2.3.0 with flag `nomasq-all-reserved-ranges` turned on. (#77458, @grayluck)
- GCE/Windows: added support for the stackdriver logging agent (#76850, @yujuhong)
- GCE Windows nodes will rely solely on kubernetes and kube-proxy (and not the GCE agent) for network address management. (#75855, @pjh)
- Ensured that the `node-role.kubernetes.io/master` taint is applied to the master with NoSchedule on GCE. (#78183, @cheftako)
- Windows nodes on GCE now use a known-working 1809 image rather than the latest 1809 image. (#76722, @pjh)
- kube-up.sh scripts now disable the KubeletPodResources feature for Windows nodes, due to issue #78628. (#78668, @mtaufen)

Instrumentation

- [metrics-server addon] Restored the ability to connect to nodes via IP addresses. (#76819, @serathius)
- If a pod has a running instance, the stats of its previously terminated instances will not show up in the kubelet summary stats any more for CRI runtimes such as containerd and cri-o. This keeps the behavior consistent with Docker integration, and fixes an issue that some container Prometheus metrics don't work when there are summary stats for multiple instances of the same pod. (#77426, @Random-Liu)

Network

- Ingress objects are now persisted in etcd using the networking.k8s.io/v1beta1 version (#77139, @cmluciano)
- Transparent kube-proxy restarts when using IPVS are now allowed. (#75283, @lbernail)
- Packets considered INVALID by conntrack are now dropped. In particular, this fixes a problem where spurious retransmits in a long-running TCP connection to a service IP could result in the connection being closed with the error "Connection reset by peer" (#74840, @anfernee)
- kube-proxy no longer automatically cleans up network rules created by running kube-proxy in other modes. If you are switching the kube-proxy mode (EG: iptables to IPVS), you will need to run `kube-proxy --cleanup`, or restart the worker node (recommended) before restarting kube-proxy. If you are not switching kube-proxy between different modes, this change should not require any action. (#76109, @vllry)
- kube-proxy: HealthzBindAddress and MetricsBindAddress now support ipv6 addresses. (#76320, @JieJhih)

- The userspace proxy now respects the IPTables proxy's minSyncInterval parameter. (#71735, @dcbw)

- iptables proxier: now routes local traffic to LB IPs to service chain (#77523, @andrewsykim)

- IPVS: Disabled graceful termination for UDP traffic to solve issues with high number of UDP connections (DNS / syslog in particular) (#77802, @lbernail)

- Fixed a bug where kube-proxy returns error due to existing ipset rules using a different hash type. (#77371, @andrewsykim)

- Fixed spurious error messages about failing to clean up iptables rules when using iptables 1.8. (#77303, @danwinship)

- Increased log level to 2 for IPVS graceful termination (#78395, @andrewsykim)

- kube-proxy: os exit when CleanupAndExit is set to true (#76732, @JieJhih)

- Kubernetes will now allow trailing dots in the externalName of Services of type ExternalName. (#78385, @thz)

Node

- The dockershim container runtime now accepts the `docker` runtime handler from a RuntimeClass. (#78323, @tallclair)
- The init container can now get its own field value as environment variable values using downwardAPI support. (#75109, @yuchengwu)
- UpdateContainerResources is no longer recorded as a `container_status` operation. It now uses the label `update_container`. (#75278, @Nessex)
- kubelet: fix fail to close kubelet->API connections on heartbeat failure when bootstrapping or client certificate rotation is disabled (#78016, @gaorong)
- Set selinux label at plugin socket directory (#73241, @vikaschoudhary16)
- Fixed detection of non-root image user ID.(#78261, @tallclair)
- Signal handling is now initialized within hyperkube commands that require it, such as apiserver and kubelet. (#76659, @S-Chan)

- The Kubelet now properly requests protobuf objects where they are supported from the apiserver, reducing load in large clusters. (#75602, @smarterclayton)

OpenStack

- You can now define a kubeconfig file for the OpenStack cloud provider. (#77415, @Fedosin)
- OpenStack user credentials can now be read from a secret instead of a local config file. (#75062, @Fedosin)

Release

- Removed hyperkube short aliases from source code because hyperkube docker image currently create these aliases. (#76953, @Rand01ph)

Scheduling

- Tolerations with the same key and effect will be merged into one that has the value of the latest toleration for best effort pods. (#75985, @ravisantoshgudimetla)
- Achieved 2X performance improvement on both required and preferred PodAffinity. (#76243, @Huang-Wei)
- Fixed a scheduler racing issue to ensure low priority pods are unschedulable on the node(s) where high priority pods have `NominatedNodeName` set to the node(s). (#77990, @Huang-Wei)

Storage

- Fixed issue with kubelet waiting on invalid devicepath on AWS (#78595, @gnufied)
- StorageOS volumes now show correct mount information (node and mount time) in the StorageOS administration CLI and UI. (#78522, @croomes)
- Fixed issue in Portworx volume driver causing controller manager to crash. (#76341, @harsh-px)
- For an empty regular file, `stat --printf %F` will now display `regular empty file` instead of `regular file`. (#62159, @dixudx)
- You can now have different operation names for different storage operations. This still prevents two operations on same volume from happening concurrently but if the operation changes, it resets the exponential backoff. (#75213, @gnufied)
- Reduced event spam for `AttachVolume` storage operation. (#75986, @mucahitkurt)
- Until this release, the iscsi plugin was waiting 10 seconds for a path to appear in the device list. However this timeout is not enough, or is less than the default device discovery timeout in most systems, which prevents certain devices from being discovered. This timeout has been raised to 30 seconds, which should help to avoid mount issues due to device discovery. (#78475, @humblec)
- Added a field to store CSI volume expansion secrets (#77516, @gnufied)
- Fixed a bug in block volume expansion. (#77317, @gnufied)
- Count PVCs that are unbound towards attach limit. (#73863, @gnufied)

VMware

- SAML token delegation (required for Zones support in vSphere) is now supported (#78876, @dougm)
- vSphere SAML token auth is now supported when using Zones (#75515, @dougm)

Windows

- Kubectl port-forward for Windows containers was added in v1.15. To use it, you'll need to build a new pause image including WinCAT. (#75479, @benmoss)
- We're working to simplify the Windows node join experience with better scripts and kubeadm. Scripts and doc updates are still in the works, but some of the needed improvements are included in 1.15. These include:
  - Windows kube-proxy will wait for HNS network creation on start (#78612, @ksubrmnn)
  - kubeadm: implemented CRI detection for Windows worker nodes (#78053, @ksubrmnn)
- Worked toward support for Windows Server version 1903, including adding Windows support for preserving the destination IP as the VIP when loadbalancing with DSR. (#74825, @ksubrmnn)
- Bug fix: Windows Kubelet nodes will now correctly search the default location for Docker credentials (`%USERPROFILE%\.docker\config.json`) when pulling images from a private registry. (https://kubernetes.io/docs/concepts/containers/images/#configuring-nodes-to-authenticate-to-a-private-registry) (#78528, @bclau)

Dependencies

Changed

- The default Go version was updated to 1.12.5. (#78528)
- cri-tools has been updated to v1.14.0. (#75658)
- Cluster Autoscaler has been updated to v1.15.0. (#78866)
- Kibana has been upgraded to v6.6.1. (#71251)
- CAdvisor has been updated to v0.33.2. (#76291)
- Fluentd-gcp-scaler has been upgraded to v0.5.2. (#76762)
- Fluentd in fluentd-elasticsearch has been upgraded to v1.4.2. (#76854)
- fluentd-elasticsearch has been updated to v2.5.2. (#76854)
- event-exporter has been updated to v0.2.5. (#77815)
- es-image has been updated to Elasticsearch 6.7.2. (#77765)
- metrics-server has been updated to v0.3.3. (#77950)
- ip-masq-agent has been updated to v2.4.1. (#77844)
- addon-manager has been updated to v9.0.1 (#77282)
- go-autorest has been updated to v11.1.2 (#77070)
- klog has been updated to 0.3.0 (#76474)
- k8s-dns-node-cache image has been updated to v1.15.1 (#76640, @george-angel)

Unchanged

- Default etcd server version remains unchanged at v3.3.10. The etcd client version was updated to v3.3.10. (#71615, #70168, #76917)
- 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)
- 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)
- crictl on GCE was updated to v1.14.0. (#75658)
- CoreDNS is unchanged at v1.3.1 as of Kubernetes 1.14. (#78691)
- 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)
- v1.15.0-rc.1
- v1.15.0-beta.2
- v1.15.0-beta.1
- v1.15.0-alpha.3
- v1.15.0-alpha.2
- v1.15.0-alpha.1

v1.15.0-rc.1

Documentation

Downloads for v1.15.0-rc.1

filename	sha512 hash
kubernetes.tar.gz	45733de20d0e46a0937577912d945434fa12604bd507f7a6df9a28b9c60b76
kubernetes-src.tar.gz	63394dee48a5c69cecd26c2a8e54e6ed5c422a239b78a267c47b640f7c6774

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	6af05492d75b4e2b510381dd7947afd104bf412cfcfff86ccf5ec1f10719
kubernetes-client-darwin-amd64.tar.gz	72e4ac298a6fc0b64673243fd0e02fe8d51d534dca6361690f204d43ae87
kubernetes-client-linux-386.tar.gz	06f96a3b48a92ec45125fbcff64ed13466be9c0aa418dfe64e158b7a122d
kubernetes-client-linux-amd64.tar.gz	ba97ccad5c572e264bccf97c69d93d49f4da02512a7e3fbfa01d5569e15c
kubernetes-client-linux-arm.tar.gz	6155c5775ebe937dabcfeb53983358e269fb43396b15a170214be0b3f682
kubernetes-client-linux-arm64.tar.gz	ff6ef9f14be3c01f700546d949cfb2da91400f93bc4c8d0dc82cea442bf2
kubernetes-client-linux-ppc64le.tar.gz	09dbec3378130acd52aee71ba0ac7ad3942ac1b05f17886868bb499c32ab
kubernetes-client-linux-s390x.tar.gz	8f1c211ef5764c57965d3ca197c93f8dcd768f7eb0ee9d5524f0867a8650
kubernetes-client-windows-386.tar.gz	4bea6bd88eb41c7c1f0d495da6d0c7f39b55f2ccbbc0939ccd97a470aeff

filename	sha512 hash
kubernetes-client-windows-amd64.tar.gz	235e83e4bcf9535fb41a5d18dae145545ca4a7703ec6f7d6b3d0c3887c698

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	7de5aa86903ae91e97ce3017d815ab944b2ce36b2a64b0d8222e49887013596d
kubernetes-server-linux-arm.tar.gz	05d42c2a72c7ec54adc4e61bccae842fbab3e6f4f06ac3123eb6449fe7828698
kubernetes-server-linux-arm64.tar.gz	143152305c6b9a99d95da4e6ed479ab33b1c4a58f5386496f9b680bf7d601d87
kubernetes-server-linux-ppc64le.tar.gz	7cf9084939319cf9ab67989151dd3384ffb4eb2c2575c8654c3afac65cabe27f
kubernetes-server-linux-s390x.tar.gz	aaca5140e6bfeb67259d47e28da75da9a8f335ed4b61580d9f13061c4010a773

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	ec53dc1eb78be6e80470c5606b515e6859a245136e6b19a6bbb1f18dbc0aa192
kubernetes-node-linux-arm.tar.gz	369e6a6f1f989af3863bc645019448964f0f1f28ace15680a888bc6e8b919237
kubernetes-node-linux-arm64.tar.gz	c3ffd6c293feec6739881bf932c4fb5d49c01698b16bf950d63185883fcadacc
kubernetes-node-linux-ppc64le.tar.gz	edeafe6bf1deeee4dd0174bdd3a09ece5a9a895667fcf60691a8b81ba5f99ec9
kubernetes-node-linux-s390x.tar.gz	3d10142101327ee9a6d754488c3e9e4fd0b5f3a43f3ef4a19c5d9da993fbab63
kubernetes-node-windows-amd64.tar.gz	514d09f3936af68746fc11e3b83f82c744ddab1c8160b59cb1b42ea8417dc098

Changelog since v1.15.0-beta.2

Other notable changes

- Resolves spurious rollouts of workload controllers when upgrading the API server, due to incorrect defaulting of an alpha procMount field in pods (#78885, @liggitt)
- vSphere: allow SAML token delegation (required for Zones support) (#78876, @dougm)
- Update Cluster Autoscaler to 1.15.0; changelog: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.15.0 (#78866, @losipiuk)
- Revert the CoreDNS version to 1.3.1 (#78691, @rajansandeep)
- CRDs get support for x-kuberntes-int-or-string to allow faithful representation of IntOrString types in CustomResources. (#78815, @sttts)
- fix: retry detach azure disk issue (#78700, @andyzhangx)
    - try to only update vm if detach a non-existing disk when got <200, error> after detach disk operation
- Fix issue with kubelet waiting on invalid devicepath on AWS (#78595, @gnufied)
- Fixed a spurious error where update requests to the status subresource of multi-version custom resources would complain about an incorrect API version. (#78713, @liggitt)
- Fix admission metrics histogram bucket sizes to cover 25ms to ~2.5 seconds. (#78608, @jpbetz)
- Revert Promotion of resource quota scope selector to GA (#78696, @ravisantoshgudimetla)

v1.15.0-beta.2

Documentation

Downloads for v1.15.0-beta.2

filename	sha512 hash
kubernetes.tar.gz	e6c98ae93c710bb655e9b55d5ae60c56001fefb0fce74c624c18a032b94798
kubernetes-src.tar.gz	c9666ddb858631721f15e988bb5c30e222f0db1c38a6d67721b9ddcfac870d

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	084e37b2d5d06aab37b34aba012eb6c2bb4d33bef433bef0340e306def8f
kubernetes-client-darwin-amd64.tar.gz	7319108bb6e7b28575d64dadc3f397de30eb6f4f3ae1bef2001a2e84f98c
kubernetes-client-linux-386.tar.gz	5c4c8993c3a57f08cf08232ce5f3ecd5a2acffe9f5bc779fd00a4042a2d2
kubernetes-client-linux-amd64.tar.gz	607cd737c944d186c096d38bc256656b6226534c36ffcaab981df0a755e6
kubernetes-client-linux-arm.tar.gz	9a0aac4210c453311d432fab0925cb9b275efa2d01335443795c35e4d7dd

filename	sha512 hash
kubernetes-client-linux-arm64.tar.gz	6f416001e9fb42e1720302a6a46cee94952a2a825281ac7c5d6cce549f81
kubernetes-client-linux-ppc64le.tar.gz	4c0e4451b6bfd08cdb851ef8e68d5206cbd55c60a65bb95e2951ab22f2f2
kubernetes-client-linux-s390x.tar.gz	d5c47fe6e79e73b426881e9ee00291952d70c65bfbdb69216e84b86ddaf2
kubernetes-client-windows-386.tar.gz	d906d737a90ca0287156e42569479c9918f89f9a02e6fb800ea250a8c2a7
kubernetes-client-windows-amd64.tar.gz	7b0c9f14600bdfb77dc2935ba0c3407f7d5720a3a0b7ca9a18fe3fabb87a

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	636ebe9044f0033e3eff310e781d395f31a871a53e322932f331d2496975148a
kubernetes-server-linux-arm.tar.gz	ff656458f1d19345538a4145b97821403f418a06503ef94f6c0d0662f671b54b
kubernetes-server-linux-arm64.tar.gz	a95199a2b2f81c38c6c14791668598986595bedd41c9e9b2e94add0e93c5d013
kubernetes-server-linux-ppc64le.tar.gz	856d949df9494576e2dbd3b99d8097e97e8c4d2d195404f8307285303ff94ab7
kubernetes-server-linux-s390x.tar.gz	7215091725f742977120f2ee4f4bc504dcff75d7258b7e90fcb4e41a2527d6cf

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	47b8c18afaa5f81b82a42309e95cf6b3f849db18bc2e8aeaaaa54ee219b5c412
kubernetes-node-linux-arm.tar.gz	64d5ad334f9448c3444cd90b0a6a7f07d83f4fb307e850686eb14b13f8926f83
kubernetes-node-linux-arm64.tar.gz	62d1e7fb2f1f271ca349d29bc43f683e7025107d893e974131063403746bb58c
kubernetes-node-linux-ppc64le.tar.gz	215a2e3a40c88922427d73af3d38b6a2827c2a699a76fa7acf1a171814d36c0a
kubernetes-node-linux-s390x.tar.gz	d75f2a2fb430e7e7368f456590698fe04930c623269ffba88dd546a45ac9dd1f

filename	sha512 hash
kubernetes-node-windows-amd64.tar.gz	`c8eeb1d9ada781a97dc368d308fb040124f644225579f18bb41bff0f354d65ea`

Changelog since v1.15.0-beta.1

Action Required

- ACTION REQUIRED The deprecated flag -conntrack-max has been removed from kube-proxy. Users of this flag should switch to -conntrack-min and -conntrack-max-per-core instead. (#78399, @rikatz)
- ACTION REQUIRED: kubeadm: the mixture of "-config" and "-certificate-key" is no longer allowed. The InitConfiguration and JoinConfiguration objects now support the "certificateKey" field and this field should be used instead of the command line argument in case a configuration file is already passed. (#78542, @neolit123)
- Azure cloud provider could now be configured by Kubernetes secrets and a new option `cloudConfigType` is introduced, whose candicate values are `file`, `secret` and `merge` (default is `merge`). (#78242, @feiskyer)
    - action required:
    - Since Azure cloud provider would read Kubernetes secrets, the following RBAC should be configured:
    - â€”
    - apiVersion: rbac.authorization.k8s.io/v1beta1
    - kind: ClusterRole
    - metadata:
    - labels:
    - kubernetes.io/cluster-service: "true"
    - name: system:azure-cloud-provider-secret-getter
    - rules:
    - - apiGroups: [""]
    - resources: ["secrets"]
    - verbs:
    - - get
    - â€”
    - apiVersion: rbac.authorization.k8s.io/v1beta1
    - kind: ClusterRoleBinding
    - metadata:
    - labels:
    - kubernetes.io/cluster-service: "true"
    - name: system:azure-cloud-provider-secret-getter
    - roleRef:
    - apiGroup: rbac.authorization.k8s.io
    - kind: ClusterRole
    - name: system:azure-cloud-provider-secret-getter
    - subjects:
    - - kind: ServiceAccount
    - name: azure-cloud-provider
    - namespace: kube-system

Other notable changes

- kube-up.sh scripts now disable the KubeletPodResources feature for Windows nodes, due to issue #78628. (#78668, @mtaufen)
- StorageOS volumes now show correct mount information (node and mount time) in the StorageOS administration CLI and UI. (#78522, @croomes)
- Horizontal Pod Autoscaling can now scale targets up even when one or more metrics are invalid/unavailable as long as one metric indicates a scale up should occur. (#78503, @gjtempleton)
- kubeadm: revert the CoreDNS version to 1.3.1 (#78545, @neolit123)
- Move online volume expansion to beta (#77755, @gnufied)
- Fixes a memory leak in Kubelet on Windows caused by not not closing containers when fetching container metrics (#78594, @benmoss)
- Windows kube-proxy will wait for HNS network creation on start (#78612, @ksubrmnn)
- Fix error handling for loading initCfg in kubeadm upgrade and apply (#78611, @odinuge)
- Route controller now respects rate limiting to the cloud provider on deletion, previously it was only for create. (#78581, @andrewsykim)
- Windows Kubelet nodes will now correctly search the default location for Docker credentials (`%USERPROFILE%.docker* Windows nodes on GCE now have the Windows firewall enabled by default. (#78507, @pjh)
- Added objectSelector to admission webhook configurations. objectSelector is evaluated the oldObject and newObject that would be sent to the webhook, and is considered to match if either object matches the selector. A null object (oldObject in the case of create, or newObject in the case of delete) or an object that cannot have labels (like a DeploymentRollback or a PodProxyOptions object) is not considered to match. Use the object selector only if the webhook is opt-in, because end users may skip the admission webhook by setting the labels. (#78505, @caesarxuchao)
- Deprecate kubelet cAdvisor json endpoints (#78504, @dashpole)
- Supports configure accessLogs for AWS NLB (#78497, @M00nF1sh)
- Till this release, iscsi plugin was waiting 10 seconds for a path to appear in the device list. However this timeout is not enough or less than default device discovery timeout in most of the systems which cause certain device to be not accounted for the volume. This timeout has been lifted to 30seconds from this release and it should help to avoid mount issues due to device discovery. (#78475, @humblec)
- Remove deprecated -pod/-p flag from kubectl exec. The flag has been marked as deprecated since k8s version v1.12 (#76713, @prksu)
- CustomResourceDefinition with invalid regular expression in the pattern field of OpenAPI v3 validation schemas are not considere structural. (#78453, @sttts)
- Fixed panic in kube-proxy when parsing iptables-save output (#78428, @luksa)
- Remove deprecated flag -cleanup-iptables from kube-proxy (#78344, @aramase)
- The storageVersionHash feature is beta now. "StorageVersionHash" is a field in the discovery document of each resource. It allows clients to detect if the storage version of that resource has changed. Its value must be treated as opaque by clients. Only equality comparison on the value is valid. (#78325, @caesarxuchao)
- Use zone from node for topology aware aws-ebs volume creation to reduce unnecessary cloud provider calls (#78276, @zhan849)
- Finalizer Protection for Service LoadBalancers is now added as Alpha (disabled by default). This feature ensures the Service resource is not fully deleted until the correlating load balancer resources are deleted. (#78262, @MrHohn)
- Introducing new semantic for metric "volume_operation_total_seconds" to be the end to end latency of volume provisioning/deletion. Existing metric

"storage_operation_duration_seconds" will remain untouched however exposed to the following potential issues: (#78061, @yuxiangqian)
    - 1. for volume's provisioned/deleted via external provisioner/deleter, "storage_operation_duration_seconds" will NOT wait for the external operation to be done before reporting latency metric (effectively close to 0). This will be fixed by using "volume_operation_total_seconds" instead
    - 2. if there's a transient error happened during "provisioning/deletion", i.e., a volume is still in-use while a deleteVolume has been called, original "storage_operation_duration_seconds" will NOT wait until a volume has been finally deleted before reporting a not accurate latency metric. The newly implemented metric "volume_operation_total_seconds", however, wait util a provisioning/deletion operation has been fully executed.
    - Potential impacts:
    - If an SLO/alert has been defined based on "volume_operation_total_seconds", it might get violated because of the more accurate metric might be significantly larger than previously reported. The metric is defined to be a histogram and the new semantic could change the distribution.
- metrics added to kubelet endpoint â€˜metrics/probes': (#77975, @logicalhan)
    - process_start_time_seconds
- NodeLocal DNSCache graduating to beta. (#77887, @prameshj)
- Kubelet will attempt to use wincat.exe in the pause container for port forwarding when running on Windows (#75479, @benmoss)
- iptables proxier: route local traffic to LB IPs to service chain (#77523, @andrewsykim)
- When the number of jobs exceeds 500, cronjob should schedule without error. (#77475, @liucimin)
- Enable 3rd party device monitoring by default (#77274, @RenaudWasTaken)
- This change enables a user to specify a DataSource/Kind of type "PersistentVolumeClaim" in their PVC spec. This can then be detected by the external csi-provisioner and plugins if capable. (#76913, @j-griffith)
- proxy/transport: Support Content-Encoding: deflate (#76551, @JieJhih)
- Add -sort-by option to kubectl top command (#75920, @artmello)
- Introduce Topolgy into the runtimeClass API (#75744, @yastij)
- Kubelet plugin registration now has retry and exponential backoff logic for when registration of plugins (like CSI or device plugin) fail. (#73891, @taragu)
- Windows support for preserving the destination IP as the VIP when loadbalancing with DSR. (#74825, @ksubrmnn)
- Add NonPrempting field to the PriorityClass. (#74614, @denkensk)
- The kubelet only collects metrics for the node, container runtime, kubelet, pods, and containers. (#72787, @dashpole)
- Improved README for k8s.io/sample-apiserver (#73447, @MikeSpreitzer)
- kubeadm: flag "-experimental-control-plane" is now deprecated. use "-control-plane" instead (#78452, @fabriziopandini)
    - kubeadm: flag "-experimental-upload-certs" is now deprecated. use "-upload-certs" instead
- Promote resource quota scope selector to GA (#78448, @ravisantoshgudimetla)
- `kubectl scale job`, deprecated since 1.10, has been removed (#78445, @soltysh)
- CustomResourcesDefinition conversion via webhooks is promoted to beta. It requires that spec.preserveUnknownFields is set to false. (#78426, @sttts)
- kubeadm: a new command `kubeadm upgrade node` is introduced for upgrading nodes (both secondary control-plane nodes and worker nodes) (#78408, @fabriziopandini)
    - The command `kubeadm upgrade node config` is now deprecated; use `kubeadm upgrade node` instead.
    - The command `kubeadm upgrade node experimental-control-plane` is now deprecated; use `kubeadm upgrade node` instead.

- Increase log level to 2 for IPVS graceful termination (#78395, @andrewsykim)
- Add support for Azure File plugin to csi-translation-lib (#78356, @andyzhangx)
- refactor AWS NLB securityGroup handling (#74692, @M00nF1sh)
- Handle resize operation for volume plugins migrated to CSI (#77994, @gnufied)
- Inline CSI ephemeral volumes can now be controlled with PodSecurityPolicy when the CSIInlineVolume alpha feature is enabled (#76915, @vladimirvivien)
- Add support for Azure Disk plugin to csi-translation-lib (#78330, @andyzhangx)
- Ensures that the node-role.kubernetes.io/master taint is applied to the master with NoSchedule on GCE. (#78183, @cheftako)
- Add Post-bind extension point to the scheduling framework (#77567, @wgliang)
- Add CRD support for default values in OpenAPI v3 validation schemas. `default` values are set for object fields which are undefined in request payload and in data read from etcd. Defaulting is alpha and disabled by default, if the feature gate CustomResourceDefaulting is not enabled. (#77558, @sttts)
- kubeadm: v1beta2 InitConfiguration no longer embeds ClusterConfiguration it it. (#77739, @rosti)
- kube-apiserver: the `--enable-logs-handler` flag and log-serving functionality is deprecated, and scheduled to be removed in v1.19. (#77611, @rohitsardesai83)
- Fix vSphere SAML token auth when using Zones (#78137, @dougm)
- Admission webhooks can now register for a single version of a resource (for example, `apps/v1 deployments`) and be called when any other version of that resource is modified (for example `extensions/v1beta1 deployments`). This allows new versions of a resource to be handled by admission webhooks without needing to update every webhook to understand the new version. See the API documentation for the `matchPolicy: Equivalent` option in MutatingWebhookConfiguration and ValidatingWebhookConfiguration types. (#78135, @liggitt)
- Add `kubeadm alpha certs certificate-key` command to generate secure random key to use on `kubeadm init --experimental-upload-certs` (#77848, @yagonobre)
- IPVS: Disable graceful termination for UDP traffic to solve issues with high number of UDP connections (DNS / syslog in particular) (#77802, @lbernail)
- In CRD webhook conversion ignore changes to metadata other than for labels and annotations. (#77743, @sttts)
- Allow trailing dots in the externalName of Services of type ExternalName. (#78385, @thz)
- Fix a bug where kube-proxy returns error due to existing ipset rules using a different hash type. (#77371, @andrewsykim)
- kubeadm: implement CRI detection for Windows worker nodes (#78053, @ksubrmnn)

v1.15.0-beta.1

Documentation

Downloads for v1.15.0-beta.1

filename	sha512 hash
kubernetes.tar.gz	`c0dcbe90feaa665613a6a1ca99c1ab68d9174c5bcd3965ff9b8d9bad345dfa`
kubernetes-src.tar.gz	`b79bc690792e0fbc380e47d6708250211a4e742d306fb433a1b6b50d5cea79`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	b79ca71cf048515084cffd9459153e6ad4898f123fda1b6aa158e5b59033
kubernetes-client-darwin-amd64.tar.gz	699a76b03ad3d1a38bd7e1ffb7765526cc33fb40b0e7dc0a782de3e9473e
kubernetes-client-linux-386.tar.gz	5fa8bc2cbd6c9f6a8c9fe3fa96cad85f98e2d21132333ab7068b73d2c7cd
kubernetes-client-linux-amd64.tar.gz	3dfbd496cd8bf9348fd2532f4c0360fe58ddfaab9d751f81cfbf9d9ddb8a
kubernetes-client-linux-arm.tar.gz	4abcac1fa5c1ca5e9d245e87ca6f601f7013b6a7e9a9d8dae7b322e62c83
kubernetes-client-linux-arm64.tar.gz	22e2d6fc8eb1f64528215901c7cc8a016dda824557667199b9c9d5478f16
kubernetes-client-linux-ppc64le.tar.gz	8d6f283020d76382e00b9e96f1c880654196aead67f17285ad1faf7ca7d1
kubernetes-client-linux-s390x.tar.gz	3320edd26be88e9ba60b5fbb326a0e42934255bb8f1c2774eb2d309318e6
kubernetes-client-windows-386.tar.gz	951d1c9b2e68615b6f26b85e27895a6dfea948b7e4c566e27b11fde8f325
kubernetes-client-windows-amd64.tar.gz	2f049941d3902b2915bea5430a29254ac0936e4890c742162993ad13a6e6

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	9049dc0680cb96245473422bb2c5c6ca8b1930d7e0256d993001f5de95f4c998
kubernetes-server-linux-arm.tar.gz	38f08b9e78ea3cbe72b473cda1cd48352ee879ce0cd414c0decf2abce63bab6b
kubernetes-server-linux-arm64.tar.gz	6cd0166162fc13c9d47cb441e8dd3ff21fae6d2417d3eb780b24ebcd615ac084
kubernetes-server-linux-ppc64le.tar.gz	79570f97383f102be77478a4bc19d0d2c2551717c5f37e8aa159a0889590fc2a
kubernetes-server-linux-s390x.tar.gz	7e1371631373407c3a1b231d09610d1029d1981026f02206a11fd58471287400

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	`819bc76079474791d468a2945c9d0858f066a54b54fcc8a84e3f9827707d6f52`
kubernetes-node-linux-arm.tar.gz	`1054e793d5a38ac0616cc3e56c85053beda3f39bc3dad965d73397756e3d78ea`
kubernetes-node-linux-arm64.tar.gz	`8357b8ee1ff5b2705fea1f70fdb3a10cb09ed1e48ee0507032dbadfb68b44b3c`
kubernetes-node-linux-ppc64le.tar.gz	`cf62d7a660dd16ee56717a786c04b457478bf51f262fefa2d1500035ccf5bb7c`
kubernetes-node-linux-s390x.tar.gz	`60f3eb8bfe3694f5def28661c62b67a56fb5d9efad7cfeb5dc7e76f8a15be625`
kubernetes-node-windows-amd64.tar.gz	`66fb625fd68a9b754e63a3e1369a21e6d2116120b5dc5aae837896f21072ce4c`

Changelog since v1.15.0-alpha.3

Action Required

- ACTION REQUIRED: Deprecated Kubelet security controls AllowPrivileged, HostNetworkSources, HostPIDSources, HostIPCSources have been removed. Enforcement of these restrictions should be done through admission control instead (e.g. PodSecurityPolicy). (#77820, @dims)
  - ACTION REQUIRED: The deprecated Kubelet flag `--allow-privileged` has been removed. Remove any use of `--allow-privileged` from your kubelet scripts or manifests.
- Fix public IPs issues when multiple clusters are sharing the same resource group. (#77630, @feiskyer)
  - action required:
    - If the cluster is upgraded from old releases and the same resource group would be shared by multiple clusters, please recreate those LoadBalancer services or add a new tag â€˜kubernetes-cluster-name: ' manually for existing public IPs.
    - For multiple clusters sharing the same resource group, they should be configured with different cluster name by `kube-controller-manager --cluster-name=<cluster-name>`

Other notable changes

- fix azure retry issue when return 2XX with error (#78298, @andyzhangx)
- The dockershim container runtime now accepts the `docker` runtime handler from a RuntimeClass. (#78323, @tallclair)

- GCE: Disable the Windows defender to work around a bug that could cause nodes to crash and reboot (#78272, @yujuhong)
- The CustomResourcePublishOpenAPI feature is now beta and enabled by default. CustomResourceDefinitions with structural schemas now publish schemas in the OpenAPI document served at `/openapi/v2`. CustomResourceDefinitions with non-structural schemas have a `NonStructuralSchema` condition added with details about what needs to be corrected in the validation schema. (#77825, @roycaihw)
- kubeadm's ignored pre-flight errors can now be configured via InitConfiguration and JoinConfiguration. (#75499, @marccarre)
- Fix broken detection of non-root image user ID (#78261, @tallclair)
- kubelet: fix fail to close kubelet->API connections on heartbeat failure when bootstrapping or client certificate rotation is disabled (#78016, @gaorong)
- remove vmsizelist call in azure disk GetVolumeLimits which happens in kubelet finally (#77851, @andyzhangx)
- reverts an aws-ebs volume provisioner optimization as we need to further discuss a viable optimization (#78200, @zhan849)
- API changes and deprecating the use of special annotations for Windows GMSA support (version beta) (#75459, @wk8)
- apiextensions: publish (only) structural OpenAPI schemas (#77554, @sttts)
- Set selinux label at plugin socket directory (#73241, @vikaschoudhary16)
- Fix a bug that causes DaemonSet rolling update to hang when its pod gets stuck at terminating. (#77773, @DaiHao)
- Kubeadm: a new command `kubeadm alpha certs check-expiration` was created in order to help users in managing expiration for local PKI certificates (#77863, @fabriziopandini)
- kubeadm: fix a bug related to volume unmount if the kubelet run directory is a symbolic link (#77507, @cuericlee)
- n/a (#78059, @figo)
- Add configuration options for the scheduling framework and its plugins. (#77501, @JieJhih)
- Publish DeleteOptions parameters for deletecollection endpoints in OpenAPI spec (#77843, @roycaihw)
- CoreDNS is now version 1.5.0 (#78030, @rajansandeep)
    - - A `ready` plugin has been included to report pod readiness
    - - The `proxy` plugin has been deprecated. The `forward` plugin is to be used instead.
    - - CoreDNS fixes the logging now that kubernetes' client lib switched to klog from glog.
- Upgrade Azure network API version to 2018-07-01, so that EnableTcpReset could be enabled on Azure standard loadbalancer (SLB). (#78012, @feiskyer)
- Fixed a scheduler racing issue to ensure low priority pods to be unschedulable on the node(s) where high priority pods have `NominatedNodeName` set to the node(s). (#77990, @Huang-Wei)
- Support starting Kubernetes on GCE using containerd in COS and Ubuntu with `KUBE_CONTAINER_RUNTIME=containerd`. (#77889, @Random-Liu)
- DelayingQueue.ShutDown() is now able to be invoked multiple times without causing a closed channel panic. (#77170, @smarterclayton)
- For admission webhooks registered for DELETE operations on k8s built APIs or CRDs, the apiserver now sends the existing object as admissionRequest.Request.OldObject to the webhook. (#76346, @caesarxuchao)
    - For custom apiservers they uses the generic registry in the apiserver library, they get this behavior automatically.
- Expose CSI volume stats via kubelet volume metrics (#76188, @humblec)

- Active watches of custom resources now terminate properly if the CRD is modified. (#78029, @liggitt)
- Add CRD spec.preserveUnknownFields boolean, defaulting to true in v1beta1 and to false in v1 CRDs. If false, fields not specified in the validation schema will be removed when sent to the API server or when read from etcd. (#77333, @sttts)
- Updates that remove remaining `metadata.finalizers` from an object that is pending deletion (non-nil metadata.deletionTimestamp) and has no graceful deletion pending (nil or 0 metadata.deletionGracePeriodSeconds) now results in immediate deletion of the object. (#77952, @liggitt)
- Deprecates the kubeadm config upload command as it's replacement is now graduated. Please see `kubeadm init phase upload-config` (#77946, @Klaven)
- k8s.io/client-go/dynamic/dynamicinformer.NewFilteredDynamicSharedInformerFactory now honours namespace argument (#77945, @michaelfig)
- `kubectl rollout restart` now works for daemonsets and statefulsets. (#77423, @apelisse)
- Fix incorrect azuredisk lun error (#77912, @andyzhangx)
- Kubelet could be run with no Azure identity now. A sample cloud provider configure is: `{"vmType": "vmss", "useInstanceMetadata": true}` (#77906, @feiskyer)
- client-go and kubectl no longer write cached discovery files with world-accessible file permissions (#77874, @yuchengwu)
- kubeadm: expose the kubeadm reset command as phases (#77847, @yagonobre)
- kubeadm: kubeadm alpha certs renew -csr-only now reads the current certificates as the authoritative source for certificates attributes (same as kubeadm alpha certs renew) (#77780, @fabriziopandini)
- Support "queue-sort" extension point for scheduling framework (#77529, @draveness)
- Allow init container to get its own field value as environment variable values(downwardAPI spport) (#75109, @yuchengwu)
- The metric `kube_proxy_sync_proxy_rules_last_timestamp_seconds` is now available, indicating the last time that kube-proxy successfully applied proxying rules. (#74027, @squeed)
- Fix panic logspam when running kubelet in standalone mode. (#77888, @tallclair)
- consume the AWS region list from the AWS SDK instead of a hard-coded list in the cloud provider (#75990, @mcrute)
- Add `Option` field to the admission webhook `AdmissionReview` API that provides the operation options (e.g. `DeleteOption` or `CreateOption`) for the operation being performed. (#77563, @jpbetz)
- Fix bug where cloud-controller-manager initializes nodes multiple times (#75405, @tghartland)
- Fixed a transient error API requests for custom resources could encounter while changes to the CustomResourceDefinition were being applied. (#77816, @liggitt)
- Fix kubectl exec usage string (#77589, @soltysh)
- CRD validation schemas should not specify `metadata` fields other than `name` and `generateName`. A schema will not be considered structural (and therefore ready for future features) if `metadata` is specified in any other way. (#77653, @sttts)
- Implement Permit extension point of the scheduling framework. (#77559, @ahg-g)
- Fixed a bug in the apiserver storage that could cause just-added finalizers to be ignored on an immediately following delete request, leading to premature deletion. (#77619, @caesarxuchao)
- add operation name for vm/vmss update operations in prometheus metrics (#77491, @andyzhangx)
- fix incorrect prometheus azure metrics (#77722, @andyzhangx)
- Clients may now request that API objects are converted to the `v1.Table` and `v1.PartialObjectMetadata` forms for generic access to objects. (#77448, @smarterclayton)

- ingress: Update in-tree Ingress controllers, examples, and clients to target networking.k8s.io/v1beta1 (#77617, @cmluciano)
- util/initsystem: add support for the OpenRC init system (#73101, @oz123)
- Signal handling is initialized within hyperkube commands that require it (apiserver, kubelet) (#76659, @S-Chan)
- Fix some service tags not supported issues for Azure LoadBalancer service (#77719, @feiskyer)
- Add Un-reserve extension point for the scheduling framework. (#77598, @danielqsj)
- Once merged, `legacy cloud providers` unit tests will run as part of ci, just as they were before they move from `./pkg/cloudproviders/providers` (#77704, @khenidak)
- Check if container memory stats are available before accessing it (#77656, @yastij)
- Add a field to store CSI volume expansion secrets (#77516, @gnufied)
- Add a condition NonStructuralSchema to CustomResourceDefinition listing Structural Schema violations as defined in KEP https://github.com/kubernetes/enhancements/blob/master/keps/sig-api-machinery/20190425-structural-openapi.md. CRD authors should update their validation schemas to be structural in order to participate in future CRD features. (#77207, @sttts)
- NONE (#74314, @oomichi)
- Update to use go 1.12.5 (#77528, @cblecker)
- Fix race conditions for Azure loadbalancer and route updates. (#77490, @feiskyer)
- remove VM API call dep in azure disk WaitForAttach (#77483, @andyzhangx)
- N/A (#77425, @figo)
- Fix TestEventChannelFull random fail (#76603, @changyaowei)
- `aws-cloud-provider` service account in the `kube-system` namespace need to be granted with list node permission with this optimization (#76976, @zhan849)
- Remove hyperkube short aliases from source code, Because hyperkube docker image currently create these aliases. (#76953, @Rand01ph)
- Allow to define kubeconfig file for OpenStack cloud provider. (#77415, @Fedosin)
- API servers using the default Google Compute Engine bootstrapping scripts will have their insecure port (`:8080`) disabled by default. To enable the insecure port, set `ENABLE_APISERVER_INSECURE_PORT=true` in kube-env or as an environment variable. (#77447, @dekkagaijin)
- GCE clusters will include some IP ranges that are not in used on the public Internet to the list of non-masq IPs. (#77458, @grayluck)
  - Bump ip-masq-agent version to v2.3.0 with flag `nomasq-all-reserved-ranges` turned on.
- Implement un-reserve extension point for the scheduling framework. (#77457, @danielqsj)
- If a pod has a running instance, the stats of its previously terminated instances will not show up in the kubelet summary stats any more for CRI runtimes like containerd and cri-o. (#77426, @Random-Liu)
  - This keeps the behavior consistent with Docker integration, and fixes an issue that some container Prometheus metrics don't work when there are summary stats for multiple instances of the same pod.
- Limit use of tags when calling EC2 API to prevent API throttling for very large clusters (#76749, @mcrute)
- When specifying an invalid value for a label, it was not always (#77144, @kenegozi)
  - clear which label the value was specified for. Starting with this release, the
  - label's key is included in such error messages, which makes debugging easier.

v1.15.0-alpha.3

Documentation

Downloads for v1.15.0-alpha.3

filename	sha512 hash
kubernetes.tar.gz	88d9ced283324136e9230a0c92ad9ade10d1f52d095d5a3f9827a1ebe0cf87
kubernetes-src.tar.gz	c6cfe656825da66e863cd08887b3ce4374e3dae0448e33c77f960aec168c1c

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	9df574b99dd03b15c784afa0bf91e826d687c5a2c7279878ddc9489e5542
kubernetes-client-darwin-amd64.tar.gz	bd8ac74d57e2c5dbfb36a8a3f79802a85393d914c0f513f83395f4b951a4
kubernetes-client-linux-386.tar.gz	8ffecc41f973564b18ee6ee0cf3d2c553e9f4649b13e99dc92f427a3861b
kubernetes-client-linux-amd64.tar.gz	8c62df3e8f02d0fe6388f82cf3af32c592783a012744b0595e5ae6609764
kubernetes-client-linux-arm.tar.gz	6e411c605778e2a079971bfe6f066bd834dcaa13a6e1369d1a5064cc16a9
kubernetes-client-linux-arm64.tar.gz	52daf658b97c66bf67b24ad45adf27e70cf8e721e616250bef06c8d4d4b6
kubernetes-client-linux-ppc64le.tar.gz	0f2fe4d16518640a958166bc9e1963d594828e6edfa37c018778ccce7976
kubernetes-client-linux-s390x.tar.gz	58582b030c95160460f7061000c19da225d175249beff26d4a3f5d415670
kubernetes-client-windows-386.tar.gz	d2898a2e2c6d28c9069479b7dfcf5dc640864e20090441c9bb101e3f6a1c
kubernetes-client-windows-amd64.tar.gz	50fa515ba4be8a30739cb811d8750260f2746914b98de9989c58e9b100d0

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	b124b2fa18935bbc15b9a3c0447df931314b41d36d2cd9a65bebd090dafec9bc
kubernetes-server-linux-arm.tar.gz	cde20282adb8d43e350c932c5a52176c2e1accb80499631a46c6d6980c1967c3
kubernetes-server-linux-arm64.tar.gz	657b24b24dddb475a737be8e65669caf3c41102de5feb990b8b0f29066f82313
kubernetes-server-linux-ppc64le.tar.gz	2373012c73109a38a6a2b64f1db716d62a65a4a64ccf246680f226dba96b598f

filename	sha512 hash
kubernetes-server-linux-s390x.tar.gz	`c2ce4362766bb08ffccea13893431c5f59d02f996fbb5fad1fe0014a9670440d`

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	`c26b0b2fff310d791c91e610252a86966df271b745a3ded8067328dab04fd3c1`
kubernetes-node-linux-arm.tar.gz	`79e70e550a401435b0f3d06b60312bc0740924ca56607eae9cd0d12dce1a6ea1`
kubernetes-node-linux-arm64.tar.gz	`5f920cf9e169c863760a27022f3f0e1503cedcb6b84089a7e77a05d2d449a9a6`
kubernetes-node-linux-ppc64le.tar.gz	`6037b555f484337e659b347ce0ca725e0a25e2e3034100a9ebc4c18668eb1020`
kubernetes-node-linux-s390x.tar.gz	`a32a0a22ade7658e5fb924ca8b0ccca40e96f872d136062842c046fd3f17ecc0`
kubernetes-node-windows-amd64.tar.gz	`005120b6500ee9839a6914a08ec270ccd273b5dea863da17d4da5ab1e47a7dee`

Changelog since v1.15.0-alpha.2

Other notable changes

- Adding ListMeta.RemainingItemCount. When responding a LIST request, if the server has more data available, and if the request does not contain label selectors or field selectors, the server sets the ListOptions.RemainingItemCount to the number of remaining objects. (#75993, @caesarxuchao)
- This PR removes unused soak test cauldron (#77335, @loqutus)
- N/A (#76966, @figo)
- kubeadm: kubeadm alpha certs renew and kubeadm upgrade now supports renews of certificates embedded in KubeConfig files managed by kubeadm; this does not apply to certificates signed by external CAs. (#77180, @fabriziopandini)
- As of Kubernetes 1.15, the SupportNodePidsLimit feature introduced as alpha in Kubernetes 1.14 is now beta, and the ability to utilize it is enabled by default. It is no longer necessary to set the feature gate `SupportNodePidsLimit=true`. In all other respects, this functionality behaves as it did in Kubernetes 1.14. (#76221, @RobertKrawitz)
- Bump addon-manager to v9.0.1 (#77282, @MrHohn)
    - - Rebase image on debian-base:v1.0.0
- Fix kubectl describe CronJobs error of `Successful Job History Limit`. (#77347, @danielqsj)

- Remove extra pod creation expections when daemonset fails to create pods in batches. (#74856, @draveness)
- enhance the daemonset sync logic in clock-skew scenario (#77208, @DaiHao)
- GCE-only flag `cloud-provider-gce-lb-src-cidrs` becomes optional for external cloud providers. (#76627, @timoreimann)
- The GCERegionalPersistentDisk feature gate (GA in 1.13) can no longer be disabled. The feature gate will be removed in v1.17. (#77412, @liggitt)
- API requests rejected by admission webhooks which specify an http status code < 400 are now assigned a 400 status code. (#77022, @liggitt)
- kubeadm: Add ability to specify certificate encryption and decryption key for the upload/download certificates phases as part of the new v1beta2 kubeadm config format. (#77012, @rosti)
- Fixes incorrect handling by kubectl of custom resources whose Kind is "Status" (#77368, @liggitt)
- kubeadm: disable the kube-proxy DaemonSet on non-Linux nodes. This step is required to support Windows worker nodes. (#76327, @neolit123)
- Add etag for NSG updates so as to fix nsg race condition (#77210, @feiskyer)
- The `series.state` field in the events.k8s.io/v1beta1 Event API is deprecated and will be removed in v1.18 (#75987, @yastij)
- API paging is now enabled by default in k8s.io/apiserver recommended options, and in k8s.io/sample-apiserver (#77278, @liggitt)
- GCE/Windows: force kill Stackdriver logging processes when the service cannot be stopped (#77378, @yujuhong)
- ingress objects are now persisted in etcd using the networking.k8s.io/v1beta1 version (#77139, @cmluciano)
- [fluentd-gcp addon] Bump fluentd-gcp-scaler to v0.5.2 to pick up security fixes. (#76762, @serathius)
- Add RuntimeClass restrictions & defaulting to PodSecurityPolicy. (#73795, @tallclair)
- Promote meta.k8s.io/v1beta1 Table and PartialObjectMetadata to v1. (#77136, @smarterclayton)
- Fix bug with block volume expansion (#77317, @gnufied)
- Fixes spurious error messages about failing to clean up iptables rules when using iptables 1.8. (#77303, @danwinship)
- Add TLS termination support for NLB (#74910, @M00nF1sh)
- Preserves existing namespace information in manifests when running `kubectl set ... --local` commands (#77267, @liggitt)
- fix issue that pull image failed from a cross-subscription Azure Container Registry when using MSI to authenticate (#77245, @norshtein)
- Clean links handling in cp's tar code (#76788, @soltysh)
- Implement and update interfaces and skeleton for the scheduling framework. (#75848, @bsalamat)
- Fixes segmentation fault issue with Protobuf library when log entries are deeply nested. (#77224, @qingling128)
- kubeadm: support sub-domain wildcards in certificate SANs (#76920, @sempr)
- Fixes an error with stuck informers when an etcd watch receives update or delete events with missing data (#76675, @ryanmcnamara)

v1.15.0-alpha.2

Documentation

Downloads for v1.15.0-alpha.2

filename	sha512 hash
kubernetes.tar.gz	88ca590c9bc2a095492310fee73bd191398375bc7f549e66e8978c48be8a9c
kubernetes-src.tar.gz	f587073d7b58903a52beeaa911c932047294be54b6f395063c65b46a61113a

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	1b944693f3813702e64f41fc11102af59beceb5ded52aac3109ebe39eb2e
kubernetes-client-darwin-amd64.tar.gz	233bba8324f7570e527f7ef22a01552c28dbabc6eef658311668ed554923
kubernetes-client-linux-386.tar.gz	1203729b3180328631d4192c5f4cfb09e3fea958be544fe4ee3e86826422
kubernetes-client-linux-amd64.tar.gz	ad0613c88d4f97b2a8f35fff607bf6168724b28838587218ccece14afb52
kubernetes-client-linux-arm.tar.gz	e9d3905d306504838d417051df43431f724ea689fd3564e575f8235fc80d
kubernetes-client-linux-arm64.tar.gz	a426b27d0851d84b76d225b9366668521441539e7582b2439e973c98c849
kubernetes-client-linux-ppc64le.tar.gz	be717777159b6f0c472754be704d543b80168cc02d76ca936f6559a55752
kubernetes-client-linux-s390x.tar.gz	4a4a08d23be247e1543c85895c211e9fee8e8fa276e5aa31ed012804fa09
kubernetes-client-windows-386.tar.gz	8d16d655d7d4213a45a583f81b31056a02dd2100d06d8072a8ec77e25563
kubernetes-client-windows-amd64.tar.gz	d4ece03464aaa9c2416d7acf9de7f94f3e01fa17f6f7469a9aedaefa90d4

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	932557827bfcc329162fcf29510f40951bdd5da4890de62fd5c44d5290349b09
kubernetes-server-linux-arm.tar.gz	e1c5349feab83ad458b9a5956026c48c7ce53f3becc09c537eda8984cea56bb2
kubernetes-server-linux-arm64.tar.gz	e8cfe09ff625b36b58d97440d82dbc06795d503729b45a8d077de7c73b70f350
kubernetes-server-linux-ppc64le.tar.gz	99770fe0abd0ec2d5f7e38d434a82fa323b2e25124e62aadf483dd68e763b072

filename	sha512 hash
kubernetes-server-linux-s390x.tar.gz	3f0772f3b470d59330dd6b44a43af640a7ec42354d734a1aef491769d20a2dad

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	9c879a12174a8c69124a649a8e6d51a5d4c174741d743f68f9ccec349aa671ca
kubernetes-node-linux-arm.tar.gz	3ac31c7f6b01896da60028037f30f8b6f331b7cd989dcfabd5623dbfbbed8a60
kubernetes-node-linux-arm64.tar.gz	669376d5673534d53d2546bc7768f00a3add74da452061dbc2892f59efba28dc
kubernetes-node-linux-ppc64le.tar.gz	b1c7fb9fcafc216fa2bd9551399f11a592922556dfad4c56fa273a7c54426fbb
kubernetes-node-linux-s390x.tar.gz	b93ae8cebd79d1ce0cb2aed66ded63b3541fcca23a1f879299c422774fb757ad
kubernetes-node-windows-amd64.tar.gz	e99127789e045972d0c52c61902f00297c208851bb65e01d28766b6f9439f81a

Changelog since v1.15.0-alpha.1

Other notable changes

- Kubemark scripts have been fixed for IKS clusters. (#76909, @Huang-Wei)
- fix azure disk list corruption issue (#77187, @andyzhangx)
- kubeadm: kubeadm upgrade now renews all the certificates used by one component before upgrading the component itself, with the exception of certificates signed by external CAs. User can eventually opt-out from certificate renewal during upgrades by setting the new flag -certificate-renewal to false. (#76862, @fabriziopandini)
- kube-proxy: os exit when CleanupAndExit is set to true (#76732, @JieJhih)
- kubectl exec now allows using resource name (e.g., deployment/mydeployment) to select a matching pod. (#73664, @prksu)
    - kubectl exec now allows using -pod-running-timeout flag to wait till at least one pod is running.
- kubeadm: add optional ECDSA support. (#76390, @rojkov)
    - kubeadm still generates RSA keys when deploying a node, but also accepts ECDSA
    - keys if they exist already in the directory specified in -cert-dir option.
- kube-proxy: HealthzBindAddress and MetricsBindAddress support ipv6 address. (#76320, @JieJhih)
- Packets considered INVALID by conntrack are now dropped. In particular, this fixes (#74840, @anfernee)
    - a problem where spurious retransmits in a long-running TCP connection to a service

    - IP could result in the connection being closed with the error "Connection reset by
    - peer"
- Introduce the v1beta2 config format to kubeadm. (#76710, @rosti)
- kubeadm: bump the minimum supported Docker version to 1.13.1 (#77051, @chenzhiwei)
- Rancher credential provider has now been removed (#77099, @dims)
- Support print volumeMode using `kubectl get pv/pvc -o wide` (#76646, @cwdsuzhou)
- Upgrade go-autorest to v11.1.2 (#77070, @feiskyer)
- Fixes a bug where dry-run is not honored for pod/eviction sub-resource. (#76969, @apelisse)
- Reduce event spam for AttachVolume storage operation (#75986, @mucahitkurt)
- Report cp errors consistently (#77010, @soltysh)
- specify azure file share name in azure file plugin (#76988, @andyzhangx)
- Migrate oom watcher not relying on cAdviosr's API any more (#74942, @WanLinghao)
- Validating admission webhooks are now properly called for CREATE operations on the following resources: tokenreviews, subjectaccessreviews, localsubjectaccessreviews, selfsubjectaccessreviews, selfsubjectrulesreviews (#76959, @sbezverk)
- Fix OpenID Connect (OIDC) token refresh when the client secret contains a special character. (#76914, @tsuna)
- kubeadm: Improve resiliency when it comes to updating the `kubeadm-config` config map upon new control plane joins or resets. This allows for safe multiple control plane joins and/or resets. (#76821, @ereslibre)
- Validating admission webhooks are now properly called for CREATE operations on the following resources: pods/binding, pods/eviction, bindings (#76910, @liggitt)
- Default TTL for DNS records in kubernetes zone is changed from 5s to 30s to keep consistent with old dnsmasq based kube-dns. The TTL can be customized with command `kubectl edit -n kube-system configmap/coredns`. (#76238, @Dieken)
- Fixed a kubemark panic when hollow-node is morphed as proxy. (#76848, @Huang-Wei)
- k8s-dns-node-cache image version v1.15.1 (#76640, @george-angel)
- GCE/Windows: add support for stackdriver logging agent (#76850, @yujuhong)
- Admission webhooks are now properly called for `scale` and `deployments/rollback` subresources (#76849, @liggitt)
- Switch to instance-level update APIs for Azure VMSS loadbalancer operations (#76656, @feiskyer)
- kubeadm: kubeadm alpha cert renew now ignores certificates signed by external CAs (#76865, @fabriziopandini)
- Update to use go 1.12.4 (#76576, @cblecker)
- [metrics-server addon] Restore connecting to nodes via IP addresses (#76819, @serathius)
- fix detach azure disk back off issue which has too big lock in failure retry condition (#76573, @andyzhangx)
- Updated klog to 0.3.0 (#76474, @vincepri)
- kube-up.sh no longer supports "centos" and "local" providers (#76711, @dims)
- Ensure the backend pools are set correctly for Azure SLB with multiple backend pools (e.g. outbound rules) (#76691, @feiskyer)
- Windows nodes on GCE use a known-working 1809 image rather than the latest 1809 image. (#76722, @pjh)
- The userspace proxy now respects the IPTables proxy's minSyncInterval parameter. (#71735, @dcbw)
- Kubeadm will now include the missing certificate key if it is unable to find an expected key during `kubeadm join` when used with the `--experimental-control-plane` flow (#76636, @mdaniel)

v1.15.0-alpha.1

Documentation

Downloads for v1.15.0-alpha.1

filename	sha512 hash
kubernetes.tar.gz	e07246d1811bfcaf092a3244f94e4bcbfd050756aea1b56e8af54e9c016c16
kubernetes-src.tar.gz	ebd902a1cfdde0d9a0062f3f21732eed76eb123da04a25f9f5c7cfce8a2926

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	88ce20f3c1f914aebca3439b3f4b642c9c371970945a25e623730826168e
kubernetes-client-darwin-amd64.tar.gz	a5c1a43c7e3dbb27c1a4c7e4111596331887206f768072e3fb7671075c11
kubernetes-client-linux-386.tar.gz	cf7513ab821cd0c979b1421034ce50e9bc0f347c184551cf4a9b6beab065
kubernetes-client-linux-amd64.tar.gz	964296e9289e12bc02ec05fb5ca9e6766654f81e1885989f8185ee8b4757
kubernetes-client-linux-arm.tar.gz	3480209c2112315d81e9ac22bc2a5961a805621b82ad80dc04c7044b7a8d
kubernetes-client-linux-arm64.tar.gz	be7d5bb5fddfbbe95d32b354b6ed26831b1afc406dc78e9188eae3d95799
kubernetes-client-linux-ppc64le.tar.gz	bfaeb3b8b0b2e2dde8900cd2910786cb68804ad7d173b6b52c15400041d7
kubernetes-client-linux-s390x.tar.gz	653c99e3171f74e52903ac9101cf8280a5e9d82969c53e9d481a72e0cb5b
kubernetes-client-windows-386.tar.gz	9b2862996eadf4e97d890f21bd4392beca80e356c7f94abaf5968b4ea3c2
kubernetes-client-windows-amd64.tar.gz	97d87fcbc0cd821b3ca5ebfbda0b38fdc9c5a5ec58e521936163fead9369

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	ffa2db2c39676e39535bcee3f41f4d178b239ca834c1aa6aafb75fb58cc5909a
kubernetes-server-linux-arm.tar.gz	a526cf7009fec5cd43da693127668006d3d6c4ebfb719e8c5b9b78bd5ad34887

filename	sha512 hash
kubernetes-server-linux-arm64.tar.gz	`4f9c8f85eebbf9f0023c9311560b7576cb5f4d2eac491e38aa4050c82b34f6a0`
kubernetes-server-linux-ppc64le.tar.gz	`bf95f15c3edd9a7f6c2911eedd55655a60da288c9df3fed4c5b2b7cc11d5e1da`
kubernetes-server-linux-s390x.tar.gz	`a2588d8b3df5f7599cd84635e5772f9ba2c665287c54a6167784bb284eb09fb0`

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	`b4e9faadd0e03d3d89de496b5248547b159a7fe0c26319d898a448f3da80eb7d`
kubernetes-node-linux-arm.tar.gz	`bf6db10d15a97ae39e2fcdf32c11c6cd8afcd254dc2fbc1fc00c5c74d6179f4e`
kubernetes-node-linux-arm64.tar.gz	`e89b95a23e36164b10510492841d7d140a9bd1799846f4ee1e8fbd74e8f6c512`
kubernetes-node-linux-ppc64le.tar.gz	`47f47c8b7fafc7d6ed0e55308ccb2a3b289e174d763c4a6415b7f1b7d2b81e4e`
kubernetes-node-linux-s390x.tar.gz	`8a0af4be530008bc8f120cd82ec592d08b09a85a2a558c10d712ff44867c4ef3`
kubernetes-node-windows-amd64.tar.gz	`f48886bf8f965572b78baf9e02417a56fab31870124240cac02809615caa0bc9`

Changelog since v1.14.0

Action Required

- client-go: The `rest.AnonymousClientConfig(*rest.Config)` `*rest.Config` helper method no longer copies custom `Transport` and `WrapTransport` fields, because those can be used to inject user credentials. (#75771, @liggitt)
- ACTION REQUIRED: The Node.Status.Volumes.Attached.DevicePath field is now unset for CSI volumes. Update any external controllers that depend on this field. (#75799, @msau42)

Other notable changes

- Remove the function Parallelize, please convert to use the function ParallelizeUntil. (#76595, @danielqsj)

- StorageObjectInUseProtection admission plugin is additionally enabled by default. (#74610, @oomichi)
    - So default enabled admission plugins are now `NamespaceLifecycle,LimitRanger,ServiceAccount,PersistentVolumeLabel,DefaultStorageClass,DefaultTolerationSeconds,MutatingAdmissionWebhook,ValidatingAdmissionWebhook,ResourceQuota,StorageObjectInUseProtection`. Please note that if you previously had not set the `--admission-control` flag, your cluster behavior may change (to be more standard).
- Juju provider source moved to the Charmed Kubernetes org (#76628, @kwmonroe)
- improve `kubectl auth can-i` command by warning users when they try access resource out of scope (#76014, @WanLinghao)
- Introduce API for watch bookmark events. (#74074, @wojtek-t)
    - Introduce Alpha field `AllowWatchBookmarks` in ListOptions for requesting watch bookmarks from apiserver. The implementation in apiserver is hidden behind feature gate `WatchBookmark` (currently in Alpha stage).
- Override protocol between etcd server and kube-apiserver on master with HTTPS instead HTTP when mTLS is enabled in GCE (#74690, @wenjiaswe)
- Fix issue in Portworx volume driver causing controller manager to crash (#76341, @harsh-px)
- kubeadm: Fix a bug where if couple of CRIs are installed a user override of the CRI during join (via kubeadm join -cri-socket â€¦) is ignored and kubeadm bails out with an error (#76505, @rosti)
- UpdateContainerResources is no longer recorded as a `container_status` operation. It now uses the label `update_container` (#75278, @Nessex)
- Bump metrics-server to v0.3.2 (#76437, @brett-elliott)
- The kubelet's /spec endpoint no longer provides cloud provider information (cloud_provider, instance_type, instance_id). (#76291, @dims)
- Change kubelet probe metrics to counter type. (#76074, @danielqsj)
    - The metrics `prober_probe_result` is replaced by `prober_probe_total`.
- Reduce GCE log rotation check from 1 hour to every 5 minutes. Rotation policy is unchanged (new day starts, log file size > 100MB). (#76352, @jpbetz)
- Add ListPager.EachListItem utility function to client-go to enable incremental processing of chunked list responses (#75849, @jpbetz)
- Added `CNI_VERSION` and `CNI_SHA1` environment variables in kube-up.sh to configure CNI versions on GCE. (#76353, @Random-Liu)
- Update cri-tools to v1.14.0 (#75658, @feiskyer)
- 2X performance improvement on both required and preferred PodAffinity. (#76243, @Huang-Wei)
- scheduler: add metrics to record number of pending pods in different queues (#75501, @Huang-Wei)
- Create a new `kubectl rollout restart` command that does a rolling restart of a deployment. (#76062, @apelisse)
- - Added port configuration to Admission webhook configuration service reference. (#74855, @mbohlool)
    - - Added port configuration to AuditSink webhook configuration service reference.
    - - Added port configuration to CRD Conversion webhook configuration service reference.
    - - Added port configuration to kube-aggregator service reference.
- `kubectl get -w` now prints custom resource definitions with custom print columns (#76161, @liggitt)
- Fixes bug in DaemonSetController causing it to stop processing some DaemonSets for 5 minutes after node removal. (#76060, @krzysztof-jastrzebski)
- no (#75820, @YoubingLi)
- Use stdlib to log stack trace when a panic occurs (#75853, @roycaihw)

- Fixes a NPD bug on GCI, so that it disables glog writing to files for log-counter (#76211, @wangzhen127)
- Tolerations with the same key and effect will be merged into one which has the value of the latest toleration for best effort pods. (#75985, @ravisantoshgudimetla)
- Fix empty array expansion error in cluster/gce/util.sh (#76111, @kewu1992)
- kube-proxy no longer automatically cleans up network rules created by running kube-proxy in other modes. If you are switching the mode that kube-proxy is in running in (EG: iptables to IPVS), you will need to run `kube-proxy --cleanup`, or restart the worker node (recommended) before restarting kube-proxy. (#76109, @vllry)
    - If you are not switching kube-proxy between different modes, this change should not require any action.
- Adds a new "storage_operation_status_count" metric for kube-controller-manager and kubelet to count success and error statues. (#75750, @msau42)
- GCE/Windows: disable stackdriver logging agent to prevent node startup failures (#76099, @yujuhong)
- StatefulSet controllers no longer force a resync every 30 seconds when nothing has changed. (#75622, @jonsabo)
- Ensures the conformance test image saves results before exiting when ginkgo returns non-zero value. (#76039, @johnSchnake)
- Add -image-repository flag to "kubeadm config images". (#75866, @jmkeyes)
- Paginate requests from the kube-apiserver watch cache to etcd in chunks. (#75389, @jpbetz)
    - Paginate reflector init and resync List calls that are not served by watch cache.
- `k8s.io/kubernetes` and published components (like `k8s.io/client-go` and `k8s.io/api`) now publish go module files containing dependency version information. See http://git.k8s.io/client-go/INSTALL.md#go-modules for details on consuming `k8s.io/client-go` using go modules. (#74877, @liggitt)
- give users the option to suppress detailed output in integration test (#76063, @Huang-Wei)
- CSI alpha CRDs have been removed (#75747, @msau42)
- Fixes a regression proxying responses from aggregated API servers which could cause watch requests to hang until the first event was received (#75887, @liggitt)
- Support specify the Resource Group of Route Table when update Pod network route (Azure) (#75580, @suker200)
- Support parsing more v1.Taint forms. `key:effect`, `key=:effect-` are now accepted. (#74159, @dlipovetsky)
- Resource list requests for PartialObjectMetadata now correctly return list metadata like the resourceVersion and the continue token. (#75971, @smarterclayton)
- `StubDomains` and `Upstreamnameserver` which contains a service name will be omitted while translating to the equivalent CoreDNS config. (#75969, @rajansandeep)
- Count PVCs that are unbound towards attach limit (#73863, @gnufied)
- Increased verbose level for local openapi aggregation logs to avoid flooding the log during normal operation (#75781, @roycaihw)
- In the â€˜kubectl describe' output, the fields with names containing special characters are displayed as-is without any pretty formatting. (#75483, @gsadhani)
- Support both JSON and YAML for scheduler configuration. (#75857, @danielqsj)
- kubeadm: fix "upgrade plan" not defaulting to a "stable" version if no version argument is passed (#75900, @neolit123)
- clean up func podTimestamp in queue (#75754, @denkensk)
- The AWS credential provider can now obtain ECR credentials even without the AWS cloud provider or being on an EC2 instance. Additionally, AWS credential provider caching has been improved to honor the ECR credential timeout. (#75587, @tiffanyfay)
- Add completed job status in Cronjob event. (#75712, @danielqsj)
- kubeadm: implement deletion of multiple bootstrap tokens at once (#75646, @bart0sh)

- GCE Windows nodes will rely solely on kubernetes and kube-proxy (and not the GCE agent) for network address management. (#75855, @pjh)
- kubeadm: preflight checks on external etcd certificates are now skipped when joining a control-plane node with automatic copy of cluster certificates (-certificate-key) (#75847, @fabriziopandini)
- [stackdriver addon] Bump prometheus-to-sd to v0.5.0 to pick up security fixes. (#75362, @serathius)
    - [fluentd-gcp addon] Bump fluentd-gcp-scaler to v0.5.1 to pick up security fixes.
    - [fluentd-gcp addon] Bump event-exporter to v0.2.4 to pick up security fixes.
    - [fluentd-gcp addon] Bump prometheus-to-sd to v0.5.0 to pick up security fixes.
    - [metatada-proxy addon] Bump prometheus-to-sd v0.5.0 to pick up security fixes.
- Support describe pod with inline csi volumes (#75513, @cwdsuzhou)
- Object count quota is now supported for namespaced custom resources using the count/. syntax. (#72384, @zhouhaibing089)
- In case kubeadm can't access the current Kubernetes version remotely and fails to parse (#72454, @rojkov)
    - the git-based version it falls back to a static predefined value of
    - k8s.io/kubernetes/cmd/kubeadm/app/constants.CurrentKubernetesVersion.
- Fixed a potential deadlock in resource quota controller (#74747, @liggitt) * Enabled recording partial usage info for quota objects specifying multiple resources, when only some of the resources' usage can be determined.
- CRI API will now be available in the kubernetes/cri-api repository (#75531, @dims)
- Support vSphere SAML token auth when using Zones (#75515, @dougm)
- Transition service account controller clients to TokenRequest API (#72179, @WanLinghao)
- kubeadm: reimplemented IPVS Proxy check that produced confusing warning message. (#75036, @bart0sh)
- Allow to read OpenStack user credentials from a secret instead of a local config file. (#75062, @Fedosin)
- watch can now be enabled for events using the flag -watch-cache-sizes on kube-apiserver (#74321, @yastij)
- kubeadm: Support for deprecated old kubeadm v1alpha3 config is totally removed. (#75179, @rosti)
- The Kubelet now properly requests protobuf objects where they are (#75602, @smarterclayton)
    - supported from the apiserver, reducing load in large clusters.
- Add name validation for dynamic client methods in client-go (#75072, @lblackstone)
- Users may now execute `get-kube-binaries.sh` to request a client for an OS/Arch unlike the one of the host on which the script is invoked. (#74889, @akutz)
- Move config local to controllers in kube-controller-manager (#72800, @stewart-yu)
- Fix some potential deadlocks and file descriptor leaking for inotify watches. (#75376, @cpuguy83)
- [IPVS] Introduces flag ipvs-strict-arp to configure stricter ARP sysctls, defaulting to false to preserve existing behaviors. This was enabled by default in 1.13.0, which impacted a few CNI plugins. (#75295, @lbernail)
- [IPVS] Allow for transparent kube-proxy restarts (#75283, @lbernail)
- Replace *_admission_latencies_milliseconds_summary and *_admission_latencies_milliseconds metrics due to reporting wrong unit (was labelled milliseconds, but reported seconds), and multiple naming guideline violations (units should be in base units and "duration" is the best practice labelling to measure the time a request takes). Please convert to use *_admission_duration_seconds and *_admission_duration_seconds_summary, these now report the unit as described, and follow the instrumentation best practices. (#75279, @danielqsj)
- Reset exponential backoff when storage operation changes (#75213, @gnufied)

- Watch will now support converting response objects into Table or PartialObjectMetadata forms. (#71548, @smarterclayton)
- N/A (#74974, @goodluckbot)
- kubeadm: fix the machine readability of "kubeadm token create -print-join-command" (#75487, @displague)
- Update Cluster Autoscaler to 1.14.0; changelog: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.14.0 (#75480, @losipiuk)

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 24, 2019 at 11:18 PM PST by update release notes (#15041) (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, rkt, 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_proxy=<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 v1.15.0, you would run the following:

```
minikube start --kubernetes-version v1.15.0
```

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
- rkt container engine

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
```

To use rkt as the container runtime run:

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

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

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

Driver	OS	HostFolder	VM
VirtualBox	Windows	C://Users	/c/Users
VMware Fusion	macOS	/Users	/Users
Xhyve	macOS	/Users	/Users

Private Container Registries

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

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

Add-ons

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

Using Minikube with an HTTP Proxy

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

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

For example:

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

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

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

Known Issues

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

Design

Minikube uses libmachine for provisioning VMs, and kubeadm to provision a Kubernetes cluster.

For more information about Minikube, see the proposal.

Additional Links

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

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
  - RHEL 7
  - Fedora 25/26 (best-effort)
  - HypriotOS v1.0.1+
  - Container Linux (tested with 1800.6.0)
- 2 GB or more of RAM per machine (any less will leave little room for your apps)
- 2 CPUs or more
- Full network connectivity between all machines in the cluster (public or private network is fine)
- Unique hostname, MAC address, and product_uuid for every node. See here for more details.
- Certain ports are open on your machines. See here for more details.
- Swap disabled. You **MUST** disable swap in order for the kubelet to work properly.

Verify the MAC address and product_uuid are unique for every node

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

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

Check network adapters

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

Check required ports

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

Protocol	Direction	Port Range	Purpose	Used By
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="v0.7.5"
mkdir -p /opt/cni/bin
curl -L "https://github.com/containernetworking/plugins/releases/
download/${CNI_VERSION}/cni-plugins-amd64-${CNI_VERSION}.tgz" |
tar -C /opt/cni/bin -xz
```

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

```
CRICTL_VERSION="v1.12.0"
mkdir -p /opt/bin
curl -L "https://github.com/kubernetes-incubator/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,ku
belet,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.

Analytics

Create an Issue Edit This Page
Page last modified on July 07, 2019 at 5:58 AM PST by Issue with k8s.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/ No package kubeadm available. (#15111) (Page History)

Edit This Page

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-311-candidate/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

```
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 June 19, 2019 at 9:20 PM PST by Issue with k8s.io/docs/setup/production-environment/container-runtimes/ (#14945) (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 `CrashLoopBackOff` state. The error happens when Kubernetes does not find `var/run/secrets/kubernetes.io/serviceaccount` files.

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

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

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

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

Area	Maturity Level
Command line UX	GA
Implementation	GA
Config file API	Beta
CoreDNS	GA
kubeadm alpha subcommands	Alpha
High availability	Beta
DynamicKubeletConfig	Alpha

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
v1.6.x	March 2017	December 2017
v1.7.x	June 2017	March 2018
v1.8.x	September 2017	June 2018
v1.9.x	December 2017	September 2018 Â
v1.10.x	March 2018	December 2018 Â
v1.11.x	June 2018	March 2019 Â
v1.12.x	September 2018	June 2019 Â
v1.13.x	December 2018	September 2019 Â
v1.14.x	March 2019	December 2019 Â
v1.15.x	June 2019	March 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 master Kubernetes cluster or high availability cluster
- Install a Pod network on the cluster so that your Pods can talk to each other

Instructions

Installing kubeadm on your hosts

See "Installing kubeadm".

> **Note:**
>
> If you have already installed kubeadm, run `apt-get update && apt-get upgrade` or `yum update` to get the latest version of kubeadm.

When you upgrade, the kubelet restarts every few seconds as it waits in a crashloop for kubeadm to tell it what to do. This crashloop is expected and normal. After you initialize your master, the kubelet runs normally.

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

Now run:

```
kubeadm init <args>
```

More information

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

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

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

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

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

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

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

```
annotation
[mark-control-plane] Marking the node kubeadm-master as control-
plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node kubeadm-master 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 master has initialized successfully!

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

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

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

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

  kubeadm join <master-ip>:<master-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:

```
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/62e44c867a2846fefb68bd5f178daf4da3095ccb/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> <master-ip>:<master-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 `<master-ip>:<master-port>`, IPv6 address must be enclosed in square brackets, for example: `[fd00::101]:2073`.

The output should look something like:

```
[preflight] Running pre-flight checks

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

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

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

A few seconds later, you should notice this node in the output from `kubectl get nodes` when run on the 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@<master ip>:/etc/kubernetes/admin.conf .
kubectl --kubeconfig ./admin.conf get nodes
```

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

(Optional) Proxying API Server to localhost

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

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

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

Tear down

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

Talking to the 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.

Edit This Page

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: v1.13.0
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: v1.13.0
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: v1.13.0
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.

Create an Issue Edit This Page
Page last modified on July 03, 2019 at 10:52 AM PST by add kubeadm ClusterConfiguration usage (#15175) (Page History)

Edit This Page

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.

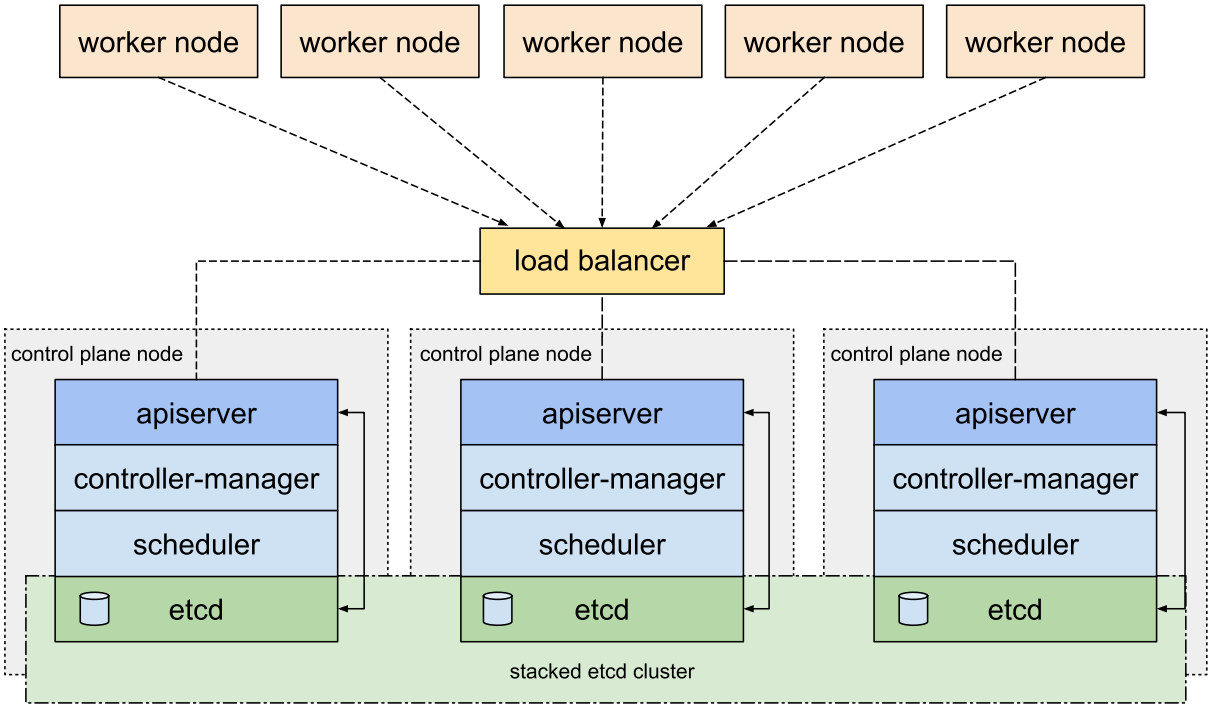
This topology couples the control planes and etcd members on the same nodes. It is simpler to set up than a cluster with external etcd nodes, and simpler to manage for replication.

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

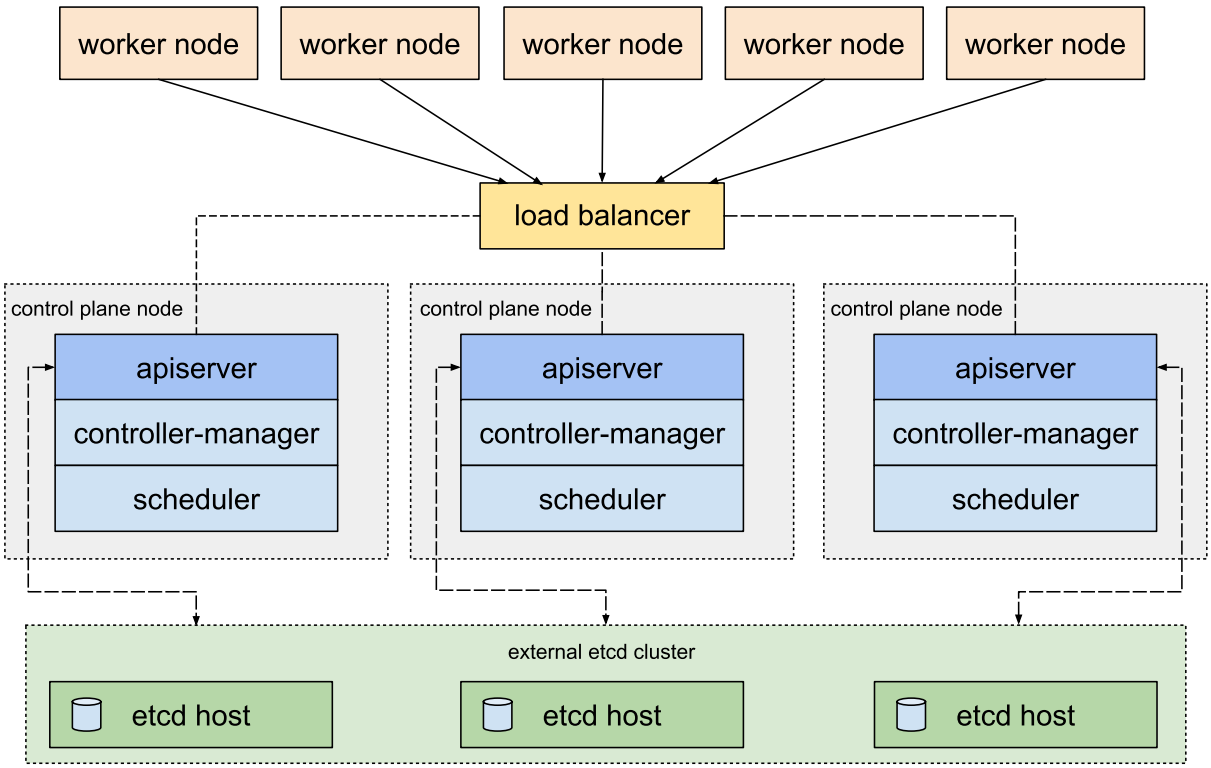
An HA cluster with external etcd is a topology where the distributed data storage cluster provided by etcd is external to the cluster formed by the nodes that run control plane components.

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. On the first control plane node, create a configuration file called `kubeadm-config.yaml`:

    ```
    apiVersion: kubeadm.k8s.io/v1beta2
    kind: ClusterConfiguration
    kubernetesVersion: stable
    controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
    ```

    - `kubernetesVersion` should be set to the Kubernetes version to use. This example uses `stable`.
    - `controlPlaneEndpoint` should match the address or DNS and port of the load balancer.

- It's recommended that the versions of kubeadm, kubelet, kubectl and Kubernetes match.

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 set the `podSubnet: 192.168.0.0/16` field under the `networking` object of `ClusterConfiguration`.

1. Initialize the control plane:

```
sudo kubeadm init --config=kubeadm-config.yaml --upload-certs
```

   - 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.

   After the command completes you should see something like so:

```
...
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
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff535
9e26aec866 --experimental-control-plane --certificate-key
f8902e114ef118304e561c3ecd4d0b543adc226b7a07f675f56564185ffe0
c07

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
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff535
9e26aec866
```

   - 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:

```
  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:

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

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: 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 we are using the `external` field for `etcd` in the kubeadm config. 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 exactly the same as described for 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.

Create an Issue Edit This Page
Page last modified on June 21, 2019 at 11:22 PM PST by Corrected flag name from v.1.14 to v1.15 (#15051) (Page History)

Edit This Page

Set up a High Availability etcd cluster with kubeadm

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

Before you begin

- 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=
ExecStart=/usr/bin/kubelet --address=127.0.0.1 --pod-manifest-path=/etc/kubernetes/manifests
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/${HOST0}/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 June 22, 2019 at 1:46 AM PST by remove kubelet --allow-privileged flag when setup ha etcd with kubeadm (#15045) (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
- Configure kubelets using kubeadm

- The kubelet drop-in file for systemd
- Kubernetes binaries and package contents

Kubelet configuration patterns

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

Propagating cluster-level configuration to each kubelet

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

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

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

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

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

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

Providing instance-specific configuration details

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

- The path to the DNS resolution file, as specified by the `--resolv-conf` kubelet configuration flag, may differ among operating systems, or depending on whether you are using `systemd-resolved`. If this path is wrong, DNS resolution will fail on the Node whose kubelet is configured incorrectly.

- The Node API object `.metadata.name` is set to the machine's hostname by default, unless you are using a cloud provider. You can use the `--hostname-override` flag to override the default behavior if you need to specify a Node name different from the machine's hostname.

- Currently, the kubelet cannot automatically detects the cgroup driver used by the CRI runtime, but the value of `--cgroup-driver` must match the cgroup driver used by the CRI runtime to ensure the health of the kubelet.

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

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

Configure kubelets using kubeadm

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

By calling `kubeadm config print 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

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

```
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/
kubernetes/bootstrap-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"
generates at runtime, populating
the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the
kubelet args as a last resort. 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.

Analytics

Create an Issue Edit This Page
Page last modified on July 14, 2019 at 4:51 PM PST by fix missing word "to" in kubeadm/kubelet-integration.md (#15419) (Page History)

Edit This Page

Configuring your kubernetes cluster to self-host the control plane

Self-hosting the Kubernetes control plane

As of 1.8, you can 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.

Page last modified on July 03, 2019 at 4:02 PM PST by kubeadm: apply some fixes to the self-hosting doc (#15252) (Page History)

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

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.

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 Alibaba Cloud

- ◦ Alibaba Cloud Container Service
  ◦ Custom Deployments

Alibaba Cloud Container Service

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

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

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

Custom Deployments

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

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

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 AWS EC2

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

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.

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.

Running Kubernetes on CenturyLink Cloud

- ◦ Find Help

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.

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 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.

Edit This Page

Running Kubernetes on Multiple Clouds with IBM Cloud Private

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.

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 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.

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

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.

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

Cloudstack

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

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

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

- Prerequisites
- Support Level

Prerequisites

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

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

```
yum install libselinux-python
```

cs is a python module for the CloudStack API.

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

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

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

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

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

Clone the playbook

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

Create a Kubernetes cluster

You simply need to run the playbook.

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

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

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

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

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

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

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

SSH to it using the key that was created and using the *core* user.

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

And you can list the machines in your cluster:

```
fleetctl list-machines
```

```
MACHINE         IP              METADATA
a017c422...     <node #1 IP>    role=node
ad13bf84...     <master IP>     role=master
e9af8293...     <node #2 IP>    role=node
```

Support Level

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

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.

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

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.

Storage

Kubernetes Volumes enable complex applications with data persistence and Pod volume sharing requirements to be deployed on Kubernetes. Kubernetes on Windows supports the following types of volumes:

- FlexVolume out-of-tree plugin with SMB and iSCSI support
- azureDisk
- azureFile
- gcePersistentDisk

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 L2 adjacency between container hosts
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, inner packet remains the same.	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 may be used when the container hosts are not L2 adjacent but have L3 connectivity

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 (net-conf.json), 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
- CSI plugins which require privileged containers
- 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**.

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

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.

Guide for adding Windows Nodes in Kubernetes

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

- Register a Windows node to the cluster
- Configure networking so pods on Linux and Windows can communicate

Before you begin

- Obtain a Windows Server license 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 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 controller

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 yaml 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
   ```

   alt_text

   Verify that the Flannel DaemonSet has the NodeSelector applied.

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

   alt_text

Join Windows Worker

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 next section.

Preparing a Windows Node

> **Note:** All code snippets in Windows sections are to be run in a PowerShell environment with elevated permissions (Admin).

1. Install Docker (requires a system reboot)

   Kubernetes uses Docker as its container engine, so we need to install it. You can follow the official Docs instructions, the Docker instructions, or try the following *recommended* steps:

   ```
   Enable-WindowsOptionalFeature -FeatureName Containers
   Restart-Computer -Force
   Install-Module -Name DockerMsftProvider -Repository
   ```

```
PSGallery -Force
Install-Package -Name Docker -ProviderName DockerMsftProvider
```

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
)
```

If after reboot you see the following error, you need to restart the docker service manually

alt_text

```
Start-Service docker
```

> **Note:** 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". The DOCKERFILE is available at https://github.com/kubernetes-sigs/sig-windows-tools/tree/master/cmd/wincat.

2. Prepare a Windows directory for Kubernetes

   Create a "Kubernetes for Windows" directory to store Kubernetes binaries as well as any deployment scripts and config files.

   ```
   mkdir c:\k
   ```

3. Copy Kubernetes certificate

   Copy the Kubernetes certificate file `$HOME/.kube/config` from the Linux controller to this new `C:\k` directory on your Windows node.

   Tip: You can use tools such as xcopy, WinSCP, or this PowerShell wrapper for WinSCP to transfer the config file between nodes.

4. Download Kubernetes binaries

   To be able to run Kubernetes, you first need to download the `kubelet` and `kube-proxy` binaries. You download these from the Node Binaries links in the CHANGELOG.md file of the latest releases. For example â€˜kubernetes-node-windows-amd64.tar.gz'. You may also optionally download `kubectl` to run on Windows which you can find under Client Binaries.

   Use the Expand-Archive PowerShell command to extract the archive and place the binaries into `C:\k`.

Join the Windows node to the Flannel cluster

The Flannel overlay deployment scripts and documentation are available in this repository. The following steps are a simple walkthrough of the more comprehensive instructions available there.

Download the Flannel start.ps1 script, the contents of which should be extracted to `C:\k`:

```
cd c:\k
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
wget https://raw.githubusercontent.com/Microsoft/SDN/master/Kubernetes/flannel/start.ps1 -o c:\k\start.ps1
```

> **Note:** start.ps1 references install.ps1, which downloads additional files such as the `flanneld` executable and the Dockerfile for infrastructure pod and install those for you. For overlay networking mode, the firewall is opened for local UDP port 4789. There may be multiple powershell windows being opened/closed as well as a few seconds of network outage while the new external vSwitch for the pod network is being created the first time. Run the script using the arguments as specified below:

```
.\start.ps1 -ManagementIP <Windows Node IP> -NetworkMode overlay  -ClusterCIDR <Cluster CIDR> -ServiceCIDR <Service CIDR> -KubeDnsServiceIP <Kube-dns Service IP> -LogDir <Log directory>
```

Parameter	Default Value	Notes
-ManagementIP	N/A (required)	The IP address assigned to the Windows node. You can use `ipconfig` to find this.
-NetworkMode	l2bridge	We're using `overlay` here
-ClusterCIDR	10.244.0.0/16	Refer to your cluster IP plan
-ServiceCIDR	10.96.0.0/12	Refer to your cluster IP plan
-KubeDnsServiceIP	10.96.0.10	
-InterfaceName	Ethernet	The name of the network interface of the Windows host. You can use `ipconfig` to find this.
-LogDir	C:\k	The directory where kubelet and kube-proxy logs are redirected into their respective output files.

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

```
kubectl get nodes
```

> **Note:** You may want to configure your Windows node components like kubelet and kube-proxy to run as services. View the services and background processes section under troubleshooting for additional instructions. Once you are running the node components as services, collecting logs becomes an important part of troubleshooting. View the gathering logs section of the contributing guide for further instructions.

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 will come in a future release. We are also making investments in cluster API to ensure Windows nodes are properly provisioned.

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 July 12, 2019 at 5:51 AM PST by Fix kubeadm link (#15286) (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.

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.

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

Building large clusters

Support

At v1.15, 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.

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.

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

Validate node setup

- ◦ Node Conformance Test
  ◦ Limitations
  ◦ Node Prerequisite

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.

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
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.
