Edit This Page

Getting started

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

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

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

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

- Learning environment
- Production environment

Learning environment

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

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

Production environment

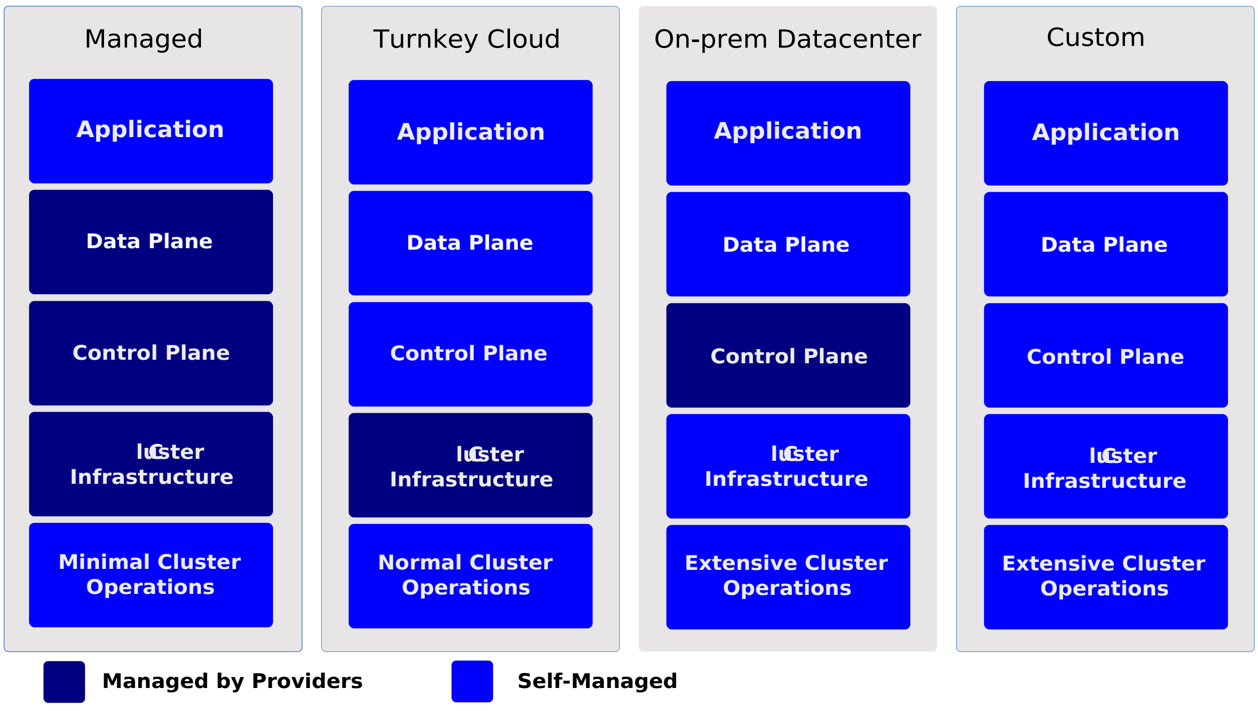
When evaluating a solution for a production environment, consider which aspects of operating a Kubernetes cluster (or *abstractions*) you want to manage yourself or offload to a provider.

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.

Edit This Page

v1.17 Release Notes

v1.17.0

Documentation

Downloads for v1.17.0

filename	sha512 hash
kubernetes.tar.gz	`68d5af15901281954de01164426cfb5ca31c14341387fad34d0cb9aa5f4`
kubernetes-src.tar.gz	`5424576d7f7936df15243fee0036e7936d2d6224e98ac805ce96cdf7b83`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	4c9a06409561b8ecc8901d0b88bc955ab8b8c99256b3f606681153921
kubernetes-client-darwin-amd64.tar.gz	78ce6875c5f5a03bc057e7194fd1966beb621f825ba786d35a9921ab1
kubernetes-client-linux-386.tar.gz	7a4bcd7d06d0f4ba929451f652c92a3c4d428f9b38ed83093f076bb25
kubernetes-client-linux-amd64.tar.gz	7f9fc9ac07e9acbf12b58ae9077a8ce1f7fb4b5ceccd3856b55d2beb5
kubernetes-client-linux-arm.tar.gz	8f74fff80a000cfaefa2409bdce6fd0d546008c7942a7178a4fa88a9b
kubernetes-client-linux-arm64.tar.gz	18d92b320f138f5080f98f1ffee20e405187549ab3aad55b7f60f02e3
kubernetes-client-linux-ppc64le.tar.gz	fd9b15a88b3d5a506a84ebfb56de291b85978b14f61a2c05f4bdb6a7e
kubernetes-client-linux-s390x.tar.gz	ae3b284a78975cbfccaac04ea802085c31fd75cccf4ece3a983f44faf
kubernetes-client-windows-386.tar.gz	4ba83b068e7f4a203bcc5cc8bb2c456a6a9c468e695f86f69d8f2ac81
kubernetes-client-windows-amd64.tar.gz	fc79b0e926a823c7d8b9010dee0c559587b7f97c9290b2126d517c427

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	28b2703c95894ab0565e372517c4a4b2c33d1be3d778fae384a6ab52c06ce
kubernetes-server-linux-arm.tar.gz	b36a9f602131dba23f267145399aad0b19e97ab7b5194b2e3c01c57f678d7
kubernetes-server-linux-arm64.tar.gz	42adae077603f25b194e893f15e7f415011f25e173507a190bafbee0d0e86
kubernetes-server-linux-ppc64le.tar.gz	7e72d4255e661e946203c1c0c684cd0923034eb112c35e3ba08fbf9d1ef5e
kubernetes-server-linux-s390x.tar.gz	00bc634654ec7d1ec2eca7a3e943ac287395503a06c8da22b7efb3a35435c

Node Binaries

filename	sha512 hash
[kubernetes-node-linux-amd64.tar.gz]()	`49ef6a41c65b3f26a4f3ffe63b92c8096c26aa27a89d227d935bc06a497c9`
[kubernetes-node-linux-arm.tar.gz]()	`21a213fd572200998bdd71f5ebbb96576fc7a7e7cfb1469f028cc1a310bc2`
[kubernetes-node-linux-arm64.tar.gz]()	`3642ee5e7476080a44005db8e7282fdbe4e4f220622761b95951c2c15b3e1`
[kubernetes-node-linux-ppc64le.tar.gz]()	`99687088be50a794894911d43827b7e1125fbc86bfba799f77c096ddaa5b2`
[kubernetes-node-linux-s390x.tar.gz]()	`73b9bc356de43fbed7d3294be747b83e0aac47051d09f1df7be52c33be670`
[kubernetes-node-windows-amd64.tar.gz]()	`2fbc80f928231f60a5a7e4f427953ef17244b3a8f6fdeebcbfceb05b0587b8`

Changes

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

What's New (Major Themes)

Cloud Provider Labels reach General Availability

Added as a beta feature way back in v1.2, v1.17 sees the general availability of cloud provider labels.

Volume Snapshot Moves to Beta

The Kubernetes Volume Snapshot feature is now beta in Kubernetes v1.17. It was introduced as alpha in Kubernetes v1.12, with a second alpha with breaking changes in Kubernetes v1.13.

CSI Migration Beta

The Kubernetes in-tree storage plugin to Container Storage Interface (CSI) migration infrastructure is now beta in Kubernetes v1.17. CSI migration was introduced as alpha in Kubernetes v1.14.

Known Issues

- volumeDevices mapping ignored when container is privileged
- The `Should recreate evicted statefulset` conformance test fails because `Pod ss-0 expected to be re-created at least once`. This was caused by the `Predicate PodFitsHostPorts failed` scheduling error. The root cause was a host port conflict for port `21017`. This port was in-use as an ephemeral port by another application running on the node. This will be looked at for the 1.18 release.
- client-go discovery clients constructed using `NewDiscoveryClientForConfig` or `NewDiscoveryClientForConfigOrDie` default to rate limits that cause normal discovery request patterns to take several seconds. This is fixed in https://issue.k8s.io/86168 and will be resolved in v1.17.1. As a workaround, the `Burst` value can be adjusted higher in the rest.Config passed into `NewDiscoveryClientForConfig` or `NewDiscoveryClientForConfigOrDie`.
- The IP allocator in v1.17.0 can return errors such as `the cluster IP <ip> for service <service-name> is not within the service CIDR <cidr>; please recreate` in the logs of the kube-apiserver. The cause is incorrect CIDR calculations if the service CIDR (`--service-cluster-ip-range`) is set to bits lower than `/16`. This is fixed in http://issue.k8s.io/86534 and will be resolved in v1.17.1.

Urgent Upgrade Notes

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

Cluster Lifecycle

- Kubeadm: add a new `kubelet-finalize` phase as part of the `init` workflow and an experimental sub-phase to enable automatic kubelet client certificate rotation on primary control-plane nodes. Prior to 1.17 and for existing nodes created by `kubeadm init` where kubelet client certificate rotation is desired, you must modify `/etc/kubernetes/kubelet.conf` to point to the PEM symlink for rotation: `client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem` and `client-key: /var/lib/kubelet/pki/kubelet-client-current.pem`, replacing the embedded client certificate and key. (#84118, @neolit123)

Network

- EndpointSlices: If upgrading a cluster with EndpointSlices already enabled, any EndpointSlices that should be managed by the EndpointSlice controller should have a `http://endpointslice.kubernetes.io/managed-by` label set to `endpointslice-controller.k8s.io`.

Scheduling

- Kubeadm: when adding extra apiserver authorization-modes, the defaults `Node,RBAC` are no longer prepended in the resulting static Pod manifests and a full override is allowed. (#82616, @ghouscht)

Storage

- A node that uses a CSI raw block volume needs to be drained before kubelet can be upgraded to 1.17. (#74026, @mkimuram)

Windows

- The Windows containers RunAsUsername feature is now beta.
- Windows worker nodes in a Kubernetes cluster now support Windows Server version 1903 in addition to the existing support for Windows Server 2019
- The RuntimeClass scheduler can now simplify steering Linux or Windows pods to appropriate nodes
- All Windows nodes now get the new label `node.kubernetes.io/windows-build` that reflects the Windows major, minor, and build number that are needed to match compatibility between Windows containers and Windows worker nodes.

Deprecations and Removals

- `kubeadm.k8s.io/v1beta1` has been deprecated, you should update your config to use newer non-deprecated API versions. (#83276, @Klaven)
- The deprecated feature gates GCERegionalPersistentDisk, EnableAggregatedDiscoveryTimeout and PersistentLocalVolumes are now unconditionally enabled and can no longer be specified in component invocations. (#82472, @draveness)
- Deprecate the default service IP CIDR. The previous default was `10.0.0.0/24` which will be removed in 6 months/2 releases. Cluster admins must specify their own desired value, by using `--service-cluster-ip-range` on kube-apiserver. (#81668, @darshanime)
- Remove deprecated "include-uninitialized" flag. (#80337, @draveness)
- All resources within the `rbac.authorization.k8s.io/v1alpha1` and `rbac.authorization.k8s.io/v1beta1` API groups are deprecated in favor of `rbac.authorization.k8s.io/v1`, and will no longer be served in v1.20. (#84758, @liggitt)
- The certificate signer no longer accepts ca.key passwords via the `CFSSL_CA_PK_PASSWORD` environment variable. This capability was not prompted by user request, never advertised, and recommended against in the security audit. (#84677, @mikedanese)
- Deprecate the instance type beta label (`beta.kubernetes.io/instance-type`) in favor of its GA equivalent: `node.kubernetes.io/instance-type` (#82049, @andrewsykim)
- The built-in system:csi-external-provisioner and system:csi-external-attacher cluster roles are removed as of 1.17 release (#84282, @tedyu)

- The in-tree GCE PD plugin `kubernetes.io/gce-pd` is now deprecated and will be removed in 1.21. Users that self-deploy Kubernetes on GCP should enable CSIMigration + CSIMigrationGCE features and install the GCE PD CSI Driver (https://github.com/kubernetes-sigs/gcp-compute-persistent-disk-csi-driver) to avoid disruption to existing Pod and PVC objects at that time. Users should start using the GCE PD CSI CSI Driver directly for any new volumes. (#85231, @davidz627)
- The in-tree AWS EBS plugin `kubernetes.io/aws-ebs` is now deprecated and will be removed in 1.21. Users that self-deploy Kubernetes on AWS should enable CSIMigration + CSIMigrationAWS features and install the AWS EBS CSI Driver (https://github.com/kubernetes-sigs/aws-ebs-csi-driver) to avoid disruption to existing Pod and PVC objects at that time. Users should start using the AWS EBS CSI CSI Driver directly for any new volumes. (#85237, @leakingtapan)
- The CSINodeInfo feature gate is deprecated and will be removed in a future release. The storage.k8s.io/v1beta1 CSINode object is deprecated and will be removed in a future release. (#83474, @msau42)
- Removed Alpha feature `MountContainers` (#84365, @codenrhoden)
- Removed plugin watching of the deprecated directory `{kubelet_root_dir}/plugins` and CSI V0 support in accordance with deprecation announcement in https://v1-13.docs.kubernetes.io/docs/setup/release/notes (#84533, @davidz627)
- kubeadm deprecates the use of the hyperkube image (#85094, @rosti)

Metrics Changes

Added metrics

- Add `scheduler_goroutines` metric to track number of kube-scheduler binding and prioritizing goroutines (#83535, @wgliang)
- Adding initial EndpointSlice metrics. (#83257, @robscott)
- Adds a metric `apiserver_request_error_total` to kube-apiserver. This metric tallies the number of `request_errors` encountered by verb, group, version, resource, subresource, scope, component, and code. (#83427, @logicalhan)
- A new `kubelet_preemptions` metric is reported from Kubelets to track the number of preemptions occurring over time, and which resource is triggering those preemptions. (#84120, @smarterclayton)
- Kube-apiserver: Added metrics `authentication_latency_seconds` that can be used to understand the latency of authentication. (#82409, @RainbowMango)
- Add `plugin_execution_duration_seconds` metric for scheduler framework plugins. (#84522, @liu-cong)
- Add `permit_wait_duration_seconds` metric to the scheduler. (#84011, @liu-cong)

Deprecated/changed metrics

- etcd version monitor metrics are now marked as with the ALPHA stability level. (#83283, @RainbowMango)

- Change `pod_preemption_victims` metric from Gauge to Histogram. (#83603, @Tabrizian)
- Following metrics from kubelet are now marked as with the ALPHA stability level: `kubelet_container_log_filesystem_used_bytes` `kubelet_volume_stats_capacity_bytes` `kubelet_volume_stats_available_bytes` `kubelet_volume_stats_used_bytes` `kubelet_volume_stats_inodes` `kubelet_volume_stats_inodes_free` `kubelet_volume_stats_inodes_used` `plugin_manager_total_plugins` `volume_manager_total_volumes` (#84907, @RainbowMango)
- Deprecated metric `rest_client_request_latency_seconds` has been turned off. (#83836, @RainbowMango)
- Following metrics from kubelet are now marked as with the ALPHA stability level: `node_cpu_usage_seconds_total` `node_memory_working_set_bytes` `container_cpu_usage_seconds_total` `container_memory_working_set_bytes` `scrape_error` (#84987, @RainbowMango)
- Deprecated prometheus request meta-metrics have been removed `http_request_duration_microseconds` `http_request_duration_microseconds_sum` `http_request_duration_microseconds_count` `http_request_size_bytes` `http_request_size_bytes_sum` `http_request_size_bytes_count` `http_requests_total, http_response_size_bytes` `http_response_size_bytes_sum` `http_response_size_bytes_count` due to removal from the prometheus client library. Prometheus http request meta-metrics are now generated from `promhttp.InstrumentMetricHandler` instead.
- Following metrics from kube-controller-manager are now marked as with the ALPHA stability level: `storage_count_attachable_volumes_in_use` `attachdetach_controller_total_volumes` `pv_collector_bound_pv_count` `pv_collector_unbound_pv_count` `pv_collector_bound_pvc_count` `pv_collector_unbound_pvc_count` (#84896, @RainbowMango)
- Following metrics have been turned off: `apiserver_request_count` `apiserver_request_latencies` `apiserver_request_latencies_summary` `apiserver_dropped_requests` `etcd_request_latencies_summary` `apiserver_storage_transformation_latencies_microseconds` `apiserver_storage_data_key_generation_latencies_microseconds` `apiserver_storage_transformation_failures_total` (#83837, @RainbowMango)
- Following metrics have been turned off: `scheduler_scheduling_latency_seconds` `scheduler_e2e_scheduling_latency_microseconds` `scheduler_scheduling_algorithm_latency_microseconds` `scheduler_scheduling_algorithm_predicate_evaluation` `scheduler_scheduling_algorithm_priority_evaluation` `scheduler_scheduling_algorithm_preemption_evaluation` `scheduler_scheduling_binding_latency_microseconds ([#83838](https://github.com/kubernetes/kubernetes/pull/83838),` @RainbowMango)
- Deprecated metric `kubeproxy_sync_proxy_rules_latency_microseconds` has been turned off. (#83839, @RainbowMango)

Notable Features

Stable

- Graduate ScheduleDaemonSetPods to GA. (feature gate will be removed in 1.18) (#82795, @draveness)
- Graduate TaintNodesByCondition to GA in 1.17. (feature gate will be removed in 1.18) (#82703, @draveness)
- The WatchBookmark feature is promoted to GA. With WatchBookmark feature, clients are able to request watch events with BOOKMARK type. See https://kubernetes.io/docs/reference/using-api/api-concepts/#watch-bookmarks for more details. (#83195, @wojtek-t)
- Promote NodeLease feature to GA. The feature make Lease object changes an additional healthiness signal from Node. Together with that, we reduce frequency of NodeStatus updates to 5m by default in case of no changes to status itself (#84351, @wojtek-t)
- CSI Topology feature is GA. (#83474, @msau42)
- The VolumeSubpathEnvExpansion feature is graduating to GA. The `VolumeSubpathEnvExpansion` feature gate is unconditionally enabled, and will be removed in v1.19. (#82578, @kevtaylor)
- Node-specific volume limits has graduated to GA. (#83568, @bertinatto)
- The ResourceQuotaScopeSelectors feature has graduated to GA. The `ResourceQuotaScopeSelectors` feature gate is now unconditionally enabled and will be removed in 1.18. (#82690, @draveness)

Beta

- The Kubernetes Volume Snapshot feature has been moved to beta. The VolumeSnapshotDataSource feature gate is on by default in this release. This feature enables you to take a snapshot of a volume (if supported by the CSI driver), and use the snapshot to provision a new volume, pre-populated with data from the snapshot.
- Feature gates CSIMigration to Beta (on by default) and CSIMigrationGCE to Beta (off by default since it requires installation of the GCE PD CSI Driver) (#85231, @davidz627)
- EndpointSlices are now beta but not yet enabled by default. Use the EndpointSlice feature gate to enable this feature. (#85365, @robscott)
- Promote CSIMigrationAWS to Beta (off by default since it requires installation of the AWS EBS CSI Driver) (#85237, @leakingtapan)
- Moving Windows RunAsUserName feature to beta (#84882, @marosset)

CLI Improvements

- The kubectl's api-resource command now has a `--sort-by` flag to sort resources by name or kind. (#81971, @laddng)
- A new `--prefix` flag added into kubectl logs which prepends each log line with information about it's source (pod name and container name) (#76471, @m1kola)

API Changes

- CustomResourceDefinitions now validate documented API semantics of `x-kubernetes-list-type` and `x-kubernetes-map-type` atomic to reject non-atomic sub-types. (#84722, @sttts)
- Kube-apiserver: The `AdmissionConfiguration` type accepted by `--admission-control-config-file` has been promoted to `apiserver.config.k8s.io/v1` with no schema changes. (#85098, @liggitt)
- Fixed EndpointSlice port name validation to match Endpoint port name validation (allowing port names longer than 15 characters) (#84481, @robscott)
- CustomResourceDefinitions introduce `x-kubernetes-map-type` annotation as a CRD API extension. Enables this particular validation for server-side apply. (#84113, @enxebre)

Other notable changes

API Machinery

- kube-apiserver: the `--runtime-config` flag now supports an `api/beta=false` value which disables all built-in REST API versions matching `v[0-9]+beta[0-9]+`. (#84304, @liggitt) The `--feature-gates` flag now supports an `AllBeta=false` value which disables all beta feature gates. (#84304, @liggitt)
- New flag `--show-hidden-metrics-for-version` in kube-apiserver can be used to show all hidden metrics that deprecated in the previous minor release. (#84292, @RainbowMango)
- kube-apiserver: Authentication configuration for mutating and validating admission webhooks referenced from an `--admission-control-config-file` can now be specified with `apiVersion: apiserver.config.k8s.io/v1, kind: WebhookAdmissionConfiguration`. (#85138, @liggitt)
- kube-apiserver: The `ResourceQuota` admission plugin configuration referenced from `--admission-control-config-file` admission config has been promoted to `apiVersion: apiserver.config.k8s.io/v1`, `kind: ResourceQuotaConfiguration` with no schema changes. (#85099, @liggitt)
- kube-apiserver: fixed a bug that could cause a goroutine leak if the apiserver encountered an encoding error serving a watch to a websocket watcher (#84693, @tedyu)
- Fix the bug that EndpointSlice for masters wasn't created after enabling EndpointSlice feature on a pre-existing cluster. (#84421, @tnqn)
- Switched intstr.Type to sized integer to follow API guidelines and improve compatibility with proto libraries (#83956, @liggitt)
- Client-go: improved allocation behavior of the delaying workqueue when handling objects with far-future ready times. (#83945, @barkbay)
- Fixed an issue with informers missing an `Added` event if a recently deleted object was immediately recreated at the same time the informer dropped a watch and relisted. (#83911, @matte21)

- Fixed panic when accessing CustomResources of a CRD with `x-kubernetes-int-or-string`. (#83787, @sttts)
- The resource version option, when passed to a list call, is now consistently interpreted as the minimum allowed resource version. Previously when listing resources that had the watch cache disabled clients could retrieve a snapshot at that exact resource version. If the client requests a resource version newer than the current state, a TimeoutError is returned suggesting the client retry in a few seconds. This behavior is now consistent for both single item retrieval and list calls, and for when the watch cache is enabled or disabled. (#72170, @jpbetz)
- Fixes a goroutine leak in kube-apiserver when a request times out. (#83333, @lavalamp)
- Fixes the bug in informer-gen that it produces incorrect code if a type has nonNamespaced tag set. (#80458, @tatsuhiro-t)
- Resolves bottleneck in internal API server communication that can cause increased goroutines and degrade API Server performance (#80465, @answer1991)
- Resolves regression generating informers for packages whose names contain `.` characters (#82410, @nikhita)
- Resolves issue with `/readyz` and `/livez` not including etcd and kms health checks (#82713, @logicalhan)
- Fixes regression in logging spurious stack traces when proxied connections are closed by the backend (#82588, @liggitt)
- Kube-apiserver now reloads serving certificates from disk every minute to allow rotation without restarting the server process (#84200, @jackkleeman)
- Client-ca bundles for the all generic-apiserver based servers will dynamically reload from disk on content changes (#83579, @deads2k)
- Client-go: Clients can request protobuf and json and correctly negotiate with the server for JSON for CRD objects, allowing all client libraries to request protobuf if it is available. If an error occurs negotiating a watch with the server, the error is immediately return by the client `Watch()` method instead of being sent as an `Error` event on the watch stream. (#84692, @smarterclayton) Renamed FeatureGate RequestManagement to APIPriorityAndFairness. This feature gate is an alpha and has not yet been associated with any actual functionality. (#85260, @MikeSpreitzer)
- Filter published OpenAPI schema by making nullable, required fields non-required in order to avoid kubectl to wrongly reject null values. (#85722, @sttts)
- kube-apiserver: fixed a conflict error encountered attempting to delete a pod with `gracePeriodSeconds=0` and a resourceVersion precondition (#85516, @michaelgugino)
- Use context to check client closed instead of http.CloseNotifier in processing watch request which will reduce 1 goroutine for each request if proto is HTTP/2.x . (#85408, @answer1991)
- Reload apiserver SNI certificates from disk every minute (#84303, @jackkleeman)
- The mutating and validating admission webhook plugins now read configuration from the admissionregistration.k8s.io/v1 API. (#80883, @liggitt)

- kube-proxy: a configuration file specified via `--config` is now loaded with strict deserialization, which fails if the config file contains duplicate or unknown fields. This protects against accidentally running with config files that are malformed, mis-indented, or have typos in field names, and getting unexpected behavior. (#82927, @obitech)
- When registering with a 1.17+ API server, MutatingWebhookConfiguration and ValidatingWebhookConfiguration objects can now request that only `v1` AdmissionReview requests be sent to them. Previously, webhooks were required to support receiving `v1beta1` AdmissionReview requests as well for compatibility with API servers <= 1.15.
  - When registering with a 1.17+ API server, a CustomResourceDefinition conversion webhook can now request that only `v1` ConversionReview requests be sent to them. Previously, conversion webhooks were required to support receiving `v1beta1` ConversionReview requests as well for compatibility with API servers <= 1.15. (#82707, @liggitt)
- OpenAPI v3 format in CustomResourceDefinition schemas are now documented. (#85381, @sttts)
- kube-apiserver: Fixed a regression accepting patch requests > 1MB (#84963, @liggitt)
- The example API server has renamed its `wardle.k8s.io` API group to `wardle.example.com` (#81670, @liggitt)
- CRDs defaulting is promoted to GA. Note: the feature gate CustomResourceDefaulting will be removed in 1.18. (#84713, @sttts)
- Restores compatibility with <=1.15.x custom resources by not publishing OpenAPI for non-structural custom resource definitions (#82653, @liggitt)
- If given an IPv6 bind-address, kube-apiserver will now advertise an IPv6 endpoint for the kubernetes.default service. (#84727, @danwinship)
- Add table convertor to component status. (#85174, @zhouya0)
- Scale custom resource unconditionally if resourceVersion is not provided (#80572, @knight42)
- When the go-client reflector relists, the ResourceVersion list option is set to the reflector's latest synced resource version to ensure the reflector does not "go back in time" and reprocess events older than it has already processed. If the server responds with an HTTP 410 (Gone) status code response, the relist falls back to using `resourceVersion=""`. (#83520, @jpbetz)
- Fix unsafe JSON construction in a number of locations in the codebase (#81158, @zouyee)
- Fixes a flaw (CVE-2019-11253) in json/yaml decoding where large or malformed documents could consume excessive server resources. Request bodies for normal API requests (create/delete/update/patch operations of regular resources) are now limited to 3MB. (#83261, @liggitt)
- CRDs can have fields named `type` with value `array` and nested array with `items` fields without validation to fall over this. (#85223, @sttts)

Apps

- Support Service Topology (#72046, @m1093782566)
- Finalizer Protection for Service LoadBalancers is now in GA (enabled by default). This feature ensures the Service resource is not fully deleted until the correlating load balancer resources are deleted. (#85023, @MrHohn)
- Pod process namespace sharing is now Generally Available. The `PodShareProcessNamespace` feature gate is now deprecated and will be removed in Kubernetes 1.19. (#84356, @verb)
- Fix handling tombstones in pod-disruption-budged controller. (#83951, @zouyee)
- Fixed the bug that deleted services were processed by EndpointSliceController repeatedly even their cleanup were successful. (#82996, @tnqn)
- Add `RequiresExactMatch` for `label.Selector` (#85048, @shaloulcy)
- Adds a new label to indicate what is managing an EndpointSlice. (#83965, @robscott)
- Fix handling tombstones in pod-disruption-budged controller. (#83951, @zouyee)
- Fixed the bug that deleted services were processed by EndpointSliceController repeatedly even their cleanup were successful. (#82996, @tnqn)
- An end-user may choose to request logs without confirming the identity of the backing kubelet. This feature can be disabled by setting the `AllowInsecureBackendProxy` feature-gate to false. (#83419, @deads2k)
- When scaling down a ReplicaSet, delete doubled up replicas first, where a "doubled up replica" is defined as one that is on the same node as an active replica belonging to a related ReplicaSet. ReplicaSets are considered "related" if they have a common controller (typically a Deployment). (#80004, @Miciah)
- Kube-controller-manager: Fixes bug setting headless service labels on endpoints (#85361, @liggitt)
- People can see the right log and note. (#84637, @zhipengzuo)
- Clean duplicate GetPodServiceMemberships function (#83902, @gongguan)

Auth

- K8s docker config json secrets are now compatible with docker config desktop authentication credentials files (#82148, @bbourbie)
- Kubelet and aggregated API servers now use v1 TokenReview and SubjectAccessReview endpoints to check authentication/authorization. (#84768, @liggitt)
- Kube-apiserver can now specify `--authentication-token-webhook-version=v1` or `--authorization-webhook-version=v1` to use `v1` TokenReview and SubjectAccessReview API objects when communicating with authentication and authorization webhooks. (#84768, @liggitt)

- Authentication token cache size is increased (from 4k to 32k) to support clusters with many nodes or many namespaces with active service accounts. (#83643, @lavalamp)
- Apiservers based on k8s.io/apiserver with delegated authn based on cluster authentication will automatically update to new authentication information when the authoritative configmap is updated. (#85004, @deads2k)
- Configmaps/extension-apiserver-authentication in kube-system is continuously updated by kube-apiservers, instead of just at apiserver start (#82705, @deads2k)

CLI

- Fixed kubectl endpointslice output for get requests (#82603, @robscott)
- Gives the right error message when using `kubectl delete` a wrong resource. (#83825, @zhouya0)
- If a bad flag is supplied to a kubectl command, only a tip to run `--help` is printed, instead of the usage menu. Usage menu is printed upon running `kubectl command --help`. (#82423, @sallyom)
- Commands like `kubectl apply` now return errors if schema-invalid annotations are specified, rather than silently dropping the entire annotations section. (#83552, @liggitt)
- Fixes spurious 0 revisions listed when running `kubectl rollout history` for a StatefulSet (#82643, @ZP-AlwaysWin)
- Correct a reference to a not/no longer used kustomize subcommand in the documentation (#82535, @demobox)
- Kubectl set resources will no longer return an error if passed an empty change for a resource. kubectl set subject will no longer return an error if passed an empty change for a resource. (#85490, @sallyom)
- Kubectl: -resource-version now works properly in label/annotate/set selector commands when racing with other clients to update the target object (#85285, @liggitt)
- The `--certificate-authority` flag now correctly overrides existing skip-TLS or CA data settings in the kubeconfig file (#83547, @liggitt)

Cloud Provider

- Azure: update disk lock logic per vm during attach/detach to allow concurrent updates for different nodes. (#85115, @aramase)
- Fix vmss dirty cache issue in disk attach/detach on vmss node (#85158, @andyzhangx)
- Fix race condition when attach/delete azure disk in same time (#84917, @andyzhangx)
- Change GCP ILB firewall names to contain the `k8s-fw-` prefix like the rest of the firewall rules. This is needed for consistency and also for other components to identify the firewall rule as k8s/service-controller managed. (#84622, @prameshj)
- Ensure health probes are created for local traffic policy UDP services on Azure (#84802, @feiskyer)

- Openstack: Do not delete managed LB in case of security group reconciliation errors (#82264, @multi-io)
- Fix aggressive VM calls for Azure VMSS (#83102, @feiskyer)
- Fix: azure disk detach failure if node not exists (#82640, @andyzhangx)
- Add azure disk encryption(SSE+CMK) support (#84605, @andyzhangx)
- Update Azure SDK versions to v35.0.0 (#84543, @andyzhangx)
- Azure: Add allow unsafe read from cache (#83685, @aramase)
- Reduces the number of calls made to the Azure API when requesting the instance view of a virtual machine scale set node. (#82496, @hasheddan)
- Added cloud operation count metrics to azure cloud controller manager. (#82574, @kkmsft)
- On AWS nodes with multiple network interfaces, kubelet should now more reliably report the same primary node IP. (#80747, @danwinship)
- Update Azure load balancer to prevent orphaned public IP addresses (#82890, @chewong)

Cluster Lifecycle

- Kubeadm alpha certs command now skip missing files (#85092, @fabriziopandini)
- Kubeadm: the command "kubeadm token create" now has a "-certificate-key" flag that can be used for the formation of join commands for control-planes with automatic copy of certificates (#84591, @TheLastProject)
- Kubeadm: Fix a bug where kubeadm cannot parse kubelet's version if the latter dumps logs on the standard error. (#85351, @rosti)
- Kubeadm: added retry to all the calls to the etcd API so kubeadm will be more resilient to network glitches (#85201, @fabriziopandini)
- Fixes a bug in kubeadm that caused init and join to hang indefinitely in specific conditions. (#85156, @chuckha)
- Kubeadm now includes CoreDNS version 1.6.5
  - `kubernetes` plugin adds metrics to measure kubernetes control plane latency.
  - the `health` plugin now includes the `lameduck` option by default, which waits for a duration before shutting down. (#85109, @rajansandeep)
- Fixed bug when using kubeadm alpha certs commands with clusters using external etcd (#85091, @fabriziopandini)
- Kubeadm no longer defaults or validates the component configs of the kubelet or kube-proxy (#79223, @rosti)
- Kubeadm: remove the deprecated `--cri-socket` flag for `kubeadm upgrade apply`. The flag has been deprecated since v1.14. (#85044, @neolit123)
- Kubeadm: prevent potential hanging of commands such as "kubeadm reset" if the apiserver endpoint is not reachable. (#84648, @neolit123)
- Kubeadm: fix skipped etcd upgrade on secondary control-plane nodes when the command `kubeadm upgrade node` is used. (#85024, @neolit123)
- Kubeadm: fix an issue with the kube-proxy container env. variables (#84888, @neolit123)

- Utilize diagnostics tool to dump GKE windows test logs (#83517, @YangLu1031)
- Kubeadm: always mount the kube-controller-manager hostPath volume that is given by the `--flex-volume-plugin-dir` flag. (#84468, @neolit123)
- Update Cluster Autoscaler version to 1.16.2 (CA release docs: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.16.2) (#84038, @losipiuk)
- Kubeadm no longer removes /etc/cni/net.d as it does not install it. Users should remove files from it manually or rely on the component that created them (#83950, @yastij)
- Kubeadm: fix wrong default value for the `upgrade node --certificate-renewal` flag. (#83528, @neolit123)
- Bump metrics-server to v0.3.5 (#83015, @olagacek)
- Dashboard: disable the dashboard Deployment on non-Linux nodes. This step is required to support Windows worker nodes. (#82975, @wawa0210)
- Fixes a panic in kube-controller-manager cleaning up bootstrap tokens (#82887, @tedyu)

- Kubeadm: add a new `kubelet-finalize` phase as part of the `init` workflow and an experimental sub-phase to enable automatic kubelet client certificate rotation on primary control-plane nodes.

  Prior to 1.17 and for existing nodes created by `kubeadm init` where kubelet client certificate rotation is desired, you must modify "/etc/kubernetes/kubelet.conf" to point to the PEM symlink for rotation:

`client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem` and `client-key: /var/lib/kubelet/pki/kubelet-client-current.pem`, replacing the embedded client certificate and key. (#84118, @neolit123)

- Kubeadm: add a upgrade health check that deploys a Job (#81319, @neolit123)
- Kubeadm now supports automatic calculations of dual-stack node cidr masks to kube-controller-manager. (#85609, @Arvinderpal)
- Kubeadm: reset raises warnings if it cannot delete folders (#85265, @SataQiu)
- Kubeadm: enable the usage of the secure kube-scheduler and kube-controller-manager ports for health checks. For kube-scheduler was 10251, becomes 10259. For kube-controller-manager was 10252, becomes 10257. (#85043, @neolit123)
- A new kubelet command line option, `--reserved-cpus`, is introduced to explicitly define the CPU list that will be reserved for system. For example, if `--reserved-cpus=0,1,2,3` is specified, then cpu 0,1,2,3 will be reserved for the system. On a system with 24 CPUs, the user may specify `isolcpus=4-23` for the kernel option and use CPU 4-23 for the user containers. (#83592, @jianzzha)
- Kubelet: a configuration file specified via `--config` is now loaded with strict deserialization, which fails if the config file contains duplicate or unknown fields. This protects against accidentally running with config

files that are malformed, mis-indented, or have typos in field names, and getting unexpected behavior. (#83204, @obitech)

- Kubeadm now propagates proxy environment variables to kube-proxy (#84559, @yastij)
- Update the latest validated version of Docker to 19.03 (#84476, @neolit123)
- Update to Ingress-GCE v1.6.1 (#84018, @rramkumar1)
- Kubeadm: enhance certs check-expiration to show the expiration info of related CAs (#83932, @SataQiu)
- Kubeadm: implemented structured output of â€˜kubeadm token list' in JSON, YAML, Go template and JsonPath formats (#78764, @bart0sh)
- Kubeadm: add support for `127.0.0.1` as advertise address. kubeadm will automatically replace this value with matching global unicast IP address on the loopback interface. (#83475, @fabriziopandini)
- Kube-scheduler: a configuration file specified via `--config` is now loaded with strict deserialization, which fails if the config file contains duplicate or unknown fields. This protects against accidentally running with config files that are malformed, mis-indented, or have typos in field names, and getting unexpected behavior. (#83030, @obitech)
- Kubeadm: use the `--service-cluster-ip-range` flag to init or use the ServiceSubnet field in the kubeadm config to pass a comma separated list of Service CIDRs. (#82473, @Arvinderpal)
- Update crictl to v1.16.1. (#82856, @Random-Liu)
- Bump addon-resizer to 1.8.7 to fix issues with using deprecated extensions APIs (#85864, @liggitt)
- Simple script based hyperkube image that bundles all the necessary binaries. This is an equivalent replacement for the image based on the go based hyperkube command + image. (#84662, @dims)
- Hyperkube will now be available in a new Github repository and will not be included in the kubernetes release from 1.17 onwards (#83454, @dims)
- Remove prometheus cluster monitoring addon from kube-up (#83442, @serathius)
- SourcesReady provides the readiness of kubelet configuration sources such as apiserver update readiness. (#81344, @zouyee)
- This PR sets the -cluster-dns flag value to kube-dns service IP whether or not NodeLocal DNSCache is enabled. NodeLocal DNSCache will listen on both the link-local as well as the service IP. (#84383, @prameshj)
- kube-dns add-on:
    - All containers are now being executed under more restrictive privileges.
    - Most of the containers now run as non-root user and has the root filesystem set as read-only.
    - The remaining container running as root only has the minimum Linux capabilities it requires to run.
    - Privilege escalation has been disabled for all containers. (#82347, @pjbgf)
- Kubernetes no longer monitors firewalld. On systems using firewalld for firewall maintenance, kube-proxy will take slightly longer to recover from disruptive firewalld operations that delete kube-proxy's iptables rules.

As a side effect of these changes, kube-proxy's `sync_proxy_rules_last_timestamp_seconds` metric no longer behaves the way it used to; now it will only change when services or endpoints actually change, rather than reliably updating every 60 seconds (or whatever). If you are trying to monitor for whether iptables updates are failing, the `sync_proxy_rules_iptables_restore_failures_total` metric may be more useful. (#81517, @danwinship)

Instrumentation

- Bump version of event-exporter to 0.3.1, to switch it to protobuf. (#83396, @loburm)
- Bumps metrics-server version to v0.3.6 with following bugfix:
    - Don't break metric storage when duplicate pod metrics encountered causing hpa to fail (#83907, @olagacek)
- addons: elasticsearch discovery supports IPv6 (#85543, @SataQiu)
- Update Cluster Autoscaler to 1.17.0; changelog: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.17.0 (#85610, @losipiuk)

Network

- The official kube-proxy image (used by kubeadm, among other things) is now compatible with systems running iptables 1.8 in "nft" mode, and will autodetect which mode it should use. (#82966, @danwinship)
- Kubenet: added HostPort IPv6 support. HostPortManager: operates only with one IP family, failing if receives port mapping entries with different IP families. HostPortSyncer: operates only with one IP family, skipping portmap entries with different IP families (#80854, @aojea)
- Kube-proxy now supports DualStack feature with EndpointSlices and IPVS. (#85246, @robscott)
- Remove redundant API validation when using Service Topology with externalTrafficPolicy=Local (#85346, @andrewsykim)
- Update github.com/vishvananda/netlink to v1.0.0 (#83576, @andrewsykim)

- `-- kube-controller-manager` `--node-cidr-mask-size-ipv4 int32` Default: 24. Mask size for IPv4 node-cidr in dual-stack cluster. `--node-cidr-mask-size-ipv6 int32` Default: 64. Mask size for IPv6 node-cidr in dual-stack cluster.

  These 2 flags can be used only for dual-stack clusters. For non dual-stack clusters, continue to use `--node-cidr-mask-size` flag to configure the mask size.

  The default node cidr mask size for IPv6 was 24 which is now changed to 64. (#79993, @aramase)

- deprecate cleanup-ipvs flag (#83832, @gongguan)

- Kube-proxy: emits a warning when a malformed component config file is used with v1alpha1. (#84143, @phenixblue)

- Set config.BindAddress to IPv4 address `127.0.0.1` if not specified (#83822, @zouyee)
- Updated kube-proxy ipvs README with correct grep argument to list loaded ipvs modules (#83677, @pete911)
- The userspace mode of kube-proxy no longer confusingly logs messages about deleting endpoints that it is actually adding. (#83644, @danwinship)
- Kube-proxy iptables probabilities are now more granular and will result in better distribution beyond 319 endpoints. (#83599, @robscott)
- Significant kube-proxy performance improvements for non UDP ports. (#83208, @robscott)
- Improved performance of kube-proxy with EndpointSlice enabled with more efficient sorting. (#83035, @robscott)
- EndpointSlices are now beta for better Network Endpoint performance at scale. (#84390, @robscott)
- Updated EndpointSlices to use PublishNotReadyAddresses from Services. (#84573, @robscott)
- When upgrading to 1.17 with a cluster with EndpointSlices enabled, the `endpointslice.kubernetes.io/managed-by` label needs to be set on each EndpointSlice. (#85359, @robscott)
- Adds FQDN addressType support for EndpointSlice. (#84091, @robscott)
- Fix incorrect network policy description suggesting that pods are isolated when a network policy has no rules of a given type (#84194, @jackkleeman)
- Fix bug where EndpointSlice controller would attempt to modify shared objects. (#85368, @robscott)
- Splitting IP address type into IPv4 and IPv6 for EndpointSlices (#84971, @robscott)
- Added appProtocol field to EndpointSlice Port (#83815, @howardjohn)
- The docker container runtime now enforces a 220 second timeout on container network operations. (#71653, @liucimin)
- Fix panic in kubelet when running IPv4/IPv6 dual-stack mode with a CNI plugin (#82508, @aanm)
- EndpointSlice hostname is now set in the same conditions Endpoints hostname is. (#84207, @robscott)

- Improving the performance of Endpoint and EndpointSlice controllers by caching Service Selectors (#84280, @gongguan)
- Significant kube-proxy performance improvements when using Endpoint Slices at scale. (#83206, @robscott)

Node

- Mirror pods now include an ownerReference for the node that created them. (#84485, @tallclair)
- Fixed a bug in the single-numa-policy of the TopologyManager. Previously, best-effort pods would result in a terminated state with a TopologyAffinity error. Now they will run as expected. (#83777, @lmdaly)
- Fixed a bug in the single-numa-node policy of the TopologyManager. Previously, pods that only requested CPU resources and did not request any third-party devices would fail to launch with a TopologyAffinity error. Now they will launch successfully. (#83697, @klueska)
- Fix error where metrics related to dynamic kubelet config isn't registered (#83184, @odinuge)
- If container fails because ContainerCannotRun, do not utilize the FallbackToLogsOnError TerminationMessagePolicy, as it masks more useful logs. (#81280, @yqwang-ms)
- Use online nodes instead of possible nodes when discovering available NUMA nodes (#83196, @zouyee)
- Use IPv4 in wincat port forward. (#83036, @liyanhui1228)
- Single static pod files and pod files from http endpoints cannot be larger than 10 MB. HTTP probe payloads are now truncated to 10KB. (#82669, @rphillips)
- Limit the body length of exec readiness/liveness probes. remote CRIs and Docker shim read a max of 16MB output of which the exec probe itself inspects 10kb. (#82514, @dims)
- Kubelet: Added kubelet serving certificate metric `server_rotation_seconds` which is a histogram reporting the age of a just rotated serving certificate in seconds. (#84534, @sambdavidson)
- Reduce default NodeStatusReportFrequency to 5 minutes. With this change, periodic node status updates will be send every 5m if node status doesn't change (otherwise they are still send with 10s).

Bump NodeProblemDetector version to v0.8.0 to reduce forced NodeStatus updates frequency to 5 minutes. (#84007, @wojtek-t)

- The topology manager aligns resources for pods of all QoS classes with respect to NUMA locality, not just Guaranteed QoS pods. (#83492, @ConnorDoyle)
- Fix a bug that a node Lease object may have been created without OwnerReference. (#84998, @wojtek-t)
- External facing APIs in plugin registration and device plugin packages are now available under k8s.io/kubelet/pkg/apis/ (#83551, @dims)

Release

- Added the `crictl` Windows binaries as well as the Linux 32bit binary to the release archives (#83944, @saschagrunert)
- Bumps the minimum version of Go required for building Kubernetes to 1.12.4. (#83596, @jktomer)
- The deprecated mondo `kubernetes-test` tarball is no longer built. Users running Kubernetes e2e tests should use the `kubernetes-test-portable` and `kubernetes-test-{OS}-{ARCH}` tarballs instead. (#83093, @ixdy)

Scheduling

- Only validate duplication of the RequestedToCapacityRatio custom priority and allow other custom predicates/priorities (#84646, @liu-cong)
- Scheduler policy configs can no longer be declared multiple times (#83963, @damemi)
- TaintNodesByCondition was graduated to GA, CheckNodeMemoryPressure, CheckNodePIDPressure, CheckNodeDiskPressure, CheckNodeCondition were accidentally removed since 1.12, the replacement is to use CheckNodeUnschedulablePred (#84152, @draveness)
- [migration phase 1] PodFitsHostPorts as filter plugin (#83659, @wgliang)
- [migration phase 1] PodFitsResources as framework plugin (#83650, @wgliang)
- [migration phase 1] PodMatchNodeSelector/NodAffinity as filter plugin (#83660, @wgliang)
- Add more tracing steps in generic_scheduler (#83539, @wgliang)
- [migration phase 1] PodFitsHost as filter plugin (#83662, @wgliang)
- Fixed a scheduler panic when using PodAffinity. (#82841, @Huang-Wei)
- Take the context as the first argument of Schedule. (#82119, @wgliang)
- Fixed an issue that the correct PluginConfig.Args is not passed to the corresponding PluginFactory in kube-scheduler when multiple PluginConfig items are defined. (#82483, @everpeace)
- Profiling is enabled by default in the scheduler (#84835, @denkensk)
- Scheduler now reports metrics on cache size including nodes, pods, and assumed pods (#83508, @damemi)
- User can now use component config to configure NodeLabel plugin for the scheduler framework. (#84297, @liu-cong)
- Optimize inter-pod affinity preferredDuringSchedulingIgnoredDuringExecution type, up to 4x in some cases. (#84264, @ahg-g)
- Filter plugin for cloud provider storage predicate (#84148, @gongguan)
- Refactor scheduler's framework permit API. (#83756, @hex108)
- Add incoming pods metrics to scheduler queue. (#83577, @liu-cong)
- Allow dynamically set glog logging level of kube-scheduler (#83910, @mrkm4ntr)

- Add latency and request count metrics for scheduler framework. (#83569, @liu-cong)
- Expose SharedInformerFactory in the framework handle (#83663, @draveness)
- Add per-pod scheduling metrics across 1 or more schedule attempts. (#83674, @liu-cong)
- Add `podInitialBackoffDurationSeconds` and `podMaxBackoffDurationSeconds` to the scheduler config API (#81263, @draveness)
- Expose kubernetes client in the scheduling framework handle. (#82432, @draveness)
- Remove MaxPriority in the scheduler API, please use MaxNodeScore or MaxExtenderPriority instead. (#83386, @draveness)
- Consolidate ScoreWithNormalizePlugin into the ScorePlugin interface (#83042, @draveness)
- New APIs to allow adding/removing pods from pre-calculated prefilter state in the scheduling framework (#82912, @ahg-g)
- Added Clone method to the scheduling framework's PluginContext and ContextData. (#82951, @ahg-g)
- Modified the scheduling framework's Filter API. (#82842, @ahg-g)
- Critical pods can now be created in namespaces other than kube-system. To limit critical pods to the kube-system namespace, cluster admins should create an admission configuration file limiting critical pods by default, and a matching quota object in the `kube-system` namespace permitting critical pods in that namespace. See https://kubernetes.io/docs/concepts/policy/resource-quotas/&#35;limit-priority-class-consumption-by-default for details. (#76310, @ravisantoshgudimetla)
- Scheduler ComponentConfig fields are now pointers (#83619, @damemi)
- Scheduler Policy API has a new recommended apiVersion `apiVersion: kubescheduler.config.k8s.io/v1` which is consistent with the scheduler API group `kubescheduler.config.k8s.io`. It holds the same API as the old apiVersion `apiVersion: v1`. (#83578, @Huang-Wei)
- Rename PluginContext to CycleState in the scheduling framework (#83430, @draveness)
- Some scheduler extender API fields are moved from `pkg/scheduler/api` to `pkg/scheduler/apis/extender/v1`. (#83262, @Huang-Wei)
- Kube-scheduler: emits a warning when a malformed component config file is used with v1alpha1. (#84129, @obitech)
- Kube-scheduler now falls back to emitting events using core/v1 Events when events.k8s.io/v1beta1 is disabled. (#83692, @yastij)
- Expand scheduler priority functions and scheduling framework plugins' node score range to [0, 100]. Note: this change is internal and does not affect extender and RequestedToCapacityRatio custom priority, which are still expected to provide a [0, 10] range. (#83522, @draveness)

Storage

- Bump CSI version to 1.2.0 (#84832, @gnufied)
- CSI Migration: Fixes issue where all volumes with the same inline volume inner spec name were staged in the same path. Migrated inline

volumes are now staged at a unique path per unique volume. (#84754, @davidz627)
- CSI Migration: GCE PD access mode now reflects read only status of inline volumes - this allows multi-attach for read only many PDs (#84809, @davidz627)
- CSI detach timeout increased from 10 seconds to 2 minutes (#84321, @cduchesne)
- Ceph RBD volume plugin now does not use any keyring (`/etc/ceph/ceph.client.lvs01cinder.keyring`, `/etc/ceph/ceph.keyring`, `/etc/ceph/keyring`, `/etc/ceph/keyring.bin`) for authentication. Ceph user credentials must be provided in PersistentVolume objects and referred Secrets. (#75588, @smileusd)
- Validate Gluster IP (#83104, @zouyee)

- PersistentVolumeLabel admission plugin, responsible for labeling `PersistentVolumes` with topology labels, now does not overwrite existing labels on PVs that were dynamically provisioned. It trusts the dynamic provisioning that it provided the correct labels to the `PersistentVolume`, saving one potentially expensive cloud API call. `PersistentVolumes` created manually by users are labelled by the admission plugin in the same way as before. (#82830, @jsafrane)

- Existing PVs are converted to use volume topology if migration is enabled. (#83394, @bertinatto)

- local: support local filesystem volume with block resource reconstruction (#84218, @cofyc)

- Fixed binding of block PersistentVolumes / PersistentVolumeClaims when BlockVolume feature is off. (#84049, @jsafrane)

- Report non-confusing error for negative storage size in PVC spec. (#82759, @sttts)

- Fixed "requested device X but found Y" attach error on AWS. (#85675, @jsafrane)

- Reduced frequency of DescribeVolumes calls of AWS API when attaching/detaching a volume. (#84181, @jsafrane)

- Fixed attachment of AWS volumes that have just been detached. (#83567, @jsafrane)

- Fix possible fd leak and closing of dirs when using openstack (#82873, @odinuge)

- local: support local volume block mode reconstruction (#84173, @cofyc)

- Fixed cleanup of raw block devices after kubelet restart. (#83451, @jsafrane)

- Add data cache flushing during unmount device for GCE-PD driver in Windows Server. (#83591, @jingxu97)

Windows

- Adds Windows Server build information as a label on the node. (#84472, @gab-satchi)
- Fixes kube-proxy bug accessing self nodeip:port on windows (#83027, @liggitt)
- When using Containerd on Windows, the `TerminationMessagePath` file will now be mounted in the Windows Pod. (#83057, @bclau)
- Fix kubelet metrics gathering on non-English Windows hosts (#84156, @wawa0210)

Dependencies

- Update etcd client side to v3.4.3 (#83987, @wenjiaswe)
- Kubernetes now requires go1.13.4+ to build (#82809, @liggitt)
- Update to use go1.12.12 (#84064, @cblecker)
- Update to go 1.12.10 (#83139, @cblecker)
- Update default etcd server version to 3.4.3 (#84329, @jingyih)
- Upgrade default etcd server version to 3.3.17 (#83804, @jpbetz)
- Upgrade to etcd client 3.3.17 to fix bug where etcd client does not parse IPv6 addresses correctly when members are joining, and to fix bug where failover on multi-member etcd cluster fails certificate check on DNS mismatch (#83801, @jpbetz)

Detailed go Dependency Changes

Added

- github.com/OpenPeeDeeP/depguard: v1.0.1
- github.com/StackExchange/wmi: 5d04971
- github.com/agnivade/levenshtein: v1.0.1
- github.com/alecthomas/template: a0175ee
- github.com/alecthomas/units: 2efee85
- github.com/andreyvit/diff: c7f18ee
- github.com/anmitsu/go-shlex: 648efa6
- github.com/bazelbuild/rules_go: 6dae44d
- github.com/bgentry/speakeasy: v0.1.0
- github.com/bradfitz/go-smtpd: deb6d62
- github.com/cockroachdb/datadriven: 80d97fb
- github.com/creack/pty: v1.1.7
- github.com/gliderlabs/ssh: v0.1.1
- github.com/go-critic/go-critic: 1df3008
- github.com/go-kit/kit: v0.8.0
- github.com/go-lintpack/lintpack: v0.5.2
- github.com/go-logfmt/logfmt: v0.3.0
- github.com/go-ole/go-ole: v1.2.1
- github.com/go-stack/stack: v1.8.0
- github.com/go-toolsmith/astcast: v1.0.0

- github.com/go-toolsmith/astcopy: v1.0.0
- github.com/go-toolsmith/astequal: v1.0.0
- github.com/go-toolsmith/astfmt: v1.0.0
- github.com/go-toolsmith/astinfo: 9809ff7
- github.com/go-toolsmith/astp: v1.0.0
- github.com/go-toolsmith/pkgload: v1.0.0
- github.com/go-toolsmith/strparse: v1.0.0
- github.com/go-toolsmith/typep: v1.0.0
- github.com/gobwas/glob: v0.2.3
- github.com/golangci/check: cfe4005
- github.com/golangci/dupl: 3e9179a
- github.com/golangci/errcheck: ef45e06
- github.com/golangci/go-misc: 927a3d8
- github.com/golangci/go-tools: e32c541
- github.com/golangci/goconst: 041c5f2
- github.com/golangci/gocyclo: 2becd97
- github.com/golangci/gofmt: 0b8337e
- github.com/golangci/golangci-lint: v1.18.0
- github.com/golangci/gosec: 66fb7fc
- github.com/golangci/ineffassign: 42439a7
- github.com/golangci/lint-1: ee948d0
- github.com/golangci/maligned: b1d8939
- github.com/golangci/misspell: 950f5d1
- github.com/golangci/prealloc: 215b22d
- github.com/golangci/revgrep: d9c87f5
- github.com/golangci/unconvert: 28b1c44
- github.com/google/go-github: v17.0.0+incompatible
- github.com/google/go-querystring: v1.0.0
- github.com/gostaticanalysis/analysisutil: v0.0.3
- github.com/jellevandenhooff/dkim: f50fe3d
- github.com/julienschmidt/httprouter: v1.2.0
- github.com/klauspost/compress: v1.4.1
- github.com/kr/logfmt: b84e30a
- github.com/logrusorgru/aurora: a7b3b31
- github.com/mattn/go-runewidth: v0.0.2
- github.com/mattn/goveralls: v0.0.2
- github.com/mitchellh/go-ps: 4fdf99a
- github.com/mozilla/tls-observatory: 8791a20
- github.com/mwitkow/go-conntrack: cc309e4
- github.com/nbutton23/zxcvbn-go: eafdab6
- github.com/olekukonko/tablewriter: a0225b3
- github.com/quasilyte/go-consistent: c6f3937
- github.com/rogpeppe/fastuuid: 6724a57
- github.com/ryanuber/go-glob: 256dc44
- github.com/sergi/go-diff: v1.0.0
- github.com/shirou/gopsutil: c95755e
- github.com/shirou/w32: bb4de01
- github.com/shurcooL/go-goon: 37c2f52
- github.com/shurcooL/go: 9e1955d
- github.com/sourcegraph/go-diff: v0.5.1
- github.com/tarm/serial: 98f6abe
- github.com/tidwall/pretty: v1.0.0

- github.com/timakin/bodyclose: 87058b9
- github.com/ultraware/funlen: v0.0.2
- github.com/urfave/cli: v1.20.0
- github.com/valyala/bytebufferpool: v1.0.0
- github.com/valyala/fasthttp: v1.2.0
- github.com/valyala/quicktemplate: v1.1.1
- github.com/valyala/tcplisten: ceec8f9
- github.com/vektah/gqlparser: v1.1.2
- go.etcd.io/etcd: 3cf2f69
- go.mongodb.org/mongo-driver: v1.1.2
- go4.org: 417644f
- golang.org/x/build: 2835ba2
- golang.org/x/perf: 6e6d33e
- golang.org/x/xerrors: a985d34
- gopkg.in/alecthomas/kingpin.v2: v2.2.6
- gopkg.in/cheggaaa/pb.v1: v1.0.25
- gopkg.in/resty.v1: v1.12.0
- grpc.go4.org: 11d0a25
- k8s.io/system-validators: v1.0.4
- mvdan.cc/interfacer: c200402
- mvdan.cc/lint: adc824a
- mvdan.cc/unparam: fbb5962
- sourcegraph.com/sqs/pbtypes: d3ebe8f

Changed

- github.com/Azure/azure-sdk-for-go: v32.5.0+incompatible â†’ v35.0.0+incompatible
- github.com/Microsoft/go-winio: v0.4.11 â†’ v0.4.14
- github.com/bazelbuild/bazel-gazelle: c728ce9 â†’ 70208cb
- github.com/bazelbuild/buildtools: 80c7f0d â†’ 69366ca
- github.com/beorn7/perks: 3a771d9 â†’ v1.0.0
- github.com/container-storage-interface/spec: v1.1.0 â†’ v1.2.0
- github.com/coredns/corefile-migration: v1.0.2 â†’ v1.0.4
- github.com/coreos/etcd: v3.3.17+incompatible â†’ v3.3.10+incompatible
- github.com/coreos/go-systemd: 39ca1b0 â†’ 95778df
- github.com/docker/go-units: v0.3.3 â†’ v0.4.0
- github.com/docker/libnetwork: a9cd636 â†’ f0e46a7
- github.com/fatih/color: v1.6.0 â†’ v1.7.0
- github.com/ghodss/yaml: c7ce166 â†’ v1.0.0
- github.com/go-openapi/analysis: v0.19.2 â†’ v0.19.5
- github.com/go-openapi/jsonpointer: v0.19.2 â†’ v0.19.3
- github.com/go-openapi/jsonreference: v0.19.2 â†’ v0.19.3
- github.com/go-openapi/loads: v0.19.2 â†’ v0.19.4
- github.com/go-openapi/runtime: v0.19.0 â†’ v0.19.4
- github.com/go-openapi/spec: v0.19.2 â†’ v0.19.3
- github.com/go-openapi/strfmt: v0.19.0 â†’ v0.19.3
- github.com/go-openapi/swag: v0.19.2 â†’ v0.19.5
- github.com/go-openapi/validate: v0.19.2 â†’ v0.19.5
- github.com/godbus/dbus: v4.1.0+incompatible â†’ 2ff6f7f

- github.com/golang/protobuf: v1.3.1 â†’ v1.3.2
- github.com/google/btree: 4030bb1 â†’ v1.0.0
- github.com/google/cadvisor: v0.34.0 â†’ v0.35.0
- github.com/gregjones/httpcache: 787624d â†’ 9cad4c3
- github.com/grpc-ecosystem/go-grpc-middleware: cfaf568 â†’ f849b54
- github.com/grpc-ecosystem/grpc-gateway: v1.3.0 â†’ v1.9.5
- github.com/heketi/heketi: v9.0.0+incompatible â†’ c2e2a4a
- github.com/json-iterator/go: v1.1.7 â†’ v1.1.8
- github.com/mailru/easyjson: 94de47d â†’ v0.7.0
- github.com/mattn/go-isatty: v0.0.3 â†’ v0.0.9
- github.com/mindprince/gonvml: fee913c â†’ 9ebdce4
- github.com/mrunalp/fileutils: 4ee1cc9 â†’ 7d4729f
- github.com/munnerz/goautoneg: a547fc6 â†’ a7dc8b6
- github.com/onsi/ginkgo: v1.8.0 â†’ v1.10.1
- github.com/onsi/gomega: v1.5.0 â†’ v1.7.0
- github.com/opencontainers/runc: 6cc5158 â†’ v1.0.0-rc9
- github.com/opencontainers/selinux: v1.2.2 â†’ 5215b18
- github.com/pkg/errors: v0.8.0 â†’ v0.8.1
- github.com/prometheus/client_golang: v0.9.2 â†’ v1.0.0
- github.com/prometheus/client_model: 5c3871d â†’ fd36f42
- github.com/prometheus/common: 4724e92 â†’ v0.4.1
- github.com/prometheus/procfs: 1dc9a6c → v0.0.2
- github.com/soheilhy/cmux: v0.1.3 â†’ v0.1.4
- github.com/spf13/pflag: v1.0.3 â†’ v1.0.5
- github.com/stretchr/testify: v1.3.0 â†’ v1.4.0
- github.com/syndtr/gocapability: e7cb7fa â†’ d983527
- github.com/vishvananda/netlink: b2de5d1 â†’ v1.0.0
- github.com/vmware/govmomi: v0.20.1 â†’ v0.20.3
- github.com/xiang90/probing: 07dd2e8 â†’ 43a291a
- go.uber.org/atomic: 8dc6146 â†’ v1.3.2
- go.uber.org/multierr: ddea229 â†’ v1.1.0
- go.uber.org/zap: 67bc79d â†’ v1.10.0
- golang.org/x/crypto: e84da03 â†’ 60c769a
- golang.org/x/lint: 8f45f77 â†’ 959b441
- golang.org/x/net: cdfb69a â†’ 13f9640
- golang.org/x/oauth2: 9f33145 â†’ 0f29369
- golang.org/x/sync: 42b3178 â†’ cd5d95a
- golang.org/x/sys: 3b52091 â†’ fde4db3
- golang.org/x/text: e6919f6 â†’ v0.3.2
- golang.org/x/time: f51c127 â†’ 9d24e82
- golang.org/x/tools: 6e04913 â†’ 65e3620
- google.golang.org/grpc: v1.23.0 â†’ v1.23.1
- gopkg.in/inf.v0: v0.9.0 â†’ v0.9.1
- k8s.io/klog: v0.4.0 â†’ v1.0.0
- k8s.io/kube-openapi: 743ec37 â†’ 30be4d1
- k8s.io/repo-infra: 00fe14e â†’ v0.0.1-alpha.1
- k8s.io/utils: 581e001 â†’ e782cd3
- sigs.k8s.io/structured-merge-diff: 6149e45 â†’ b1b620d

Removed

- github.com/cloudflare/cfssl: 56268a6
- github.com/coreos/bbolt: v1.3.3
- github.com/coreos/rkt: v1.30.0
- github.com/globalsign/mgo: eeefdec
- github.com/google/certificate-transparency-go: v1.0.21
- github.com/heketi/rest: aa6a652
- github.com/heketi/utils: 435bc5b
- github.com/pborman/uuid: v1.2.0

Feedback

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 component upgrade order

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 team. The patch release team is part of release managers. For more information, see Kubernetes Patch releases.

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:

Edit This Page

Installing Kubernetes with Minikube

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

Minikube Features

Minikube supports the following Kubernetes features:

- DNS
- NodePorts
- ConfigMaps and Secrets

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

Installation

See Installing Minikube.

Quickstart

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

1. Start Minikube and create a cluster:

   ```
   minikube start
   ```

   The output is similar to this:

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

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

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

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

   ```
   kubectl create deployment hello-minikube --image=k8s.gcr.io/
   echoserver:1.10
   ```

   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 --
   port=8080
   ```

   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 v1.17.0, you would run the following:

```
minikube start --kubernetes-version v1.17.0
```

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 virtual machine. You need to be running Linux and to have DockerDocker is a software technology providing operating-system-level virtualization also known as containers. installed.)

> **Caution:** If you use the `none` driver, some Kubernetes components run as privileged containers that have side effects outside of the

Minikube environment. Those side effects mean that the `none` driver is not recommended for personal workstations.

Starting a cluster on alternative container runtimes

You can start Minikube on the following container runtimes.

- containerd
- CRI-O

To use containerd as the container runtime, run:

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

Or you can use the extended version:

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

Edit This Page

Installing Kubernetes with Kind

Kind is a tool for running local Kubernetes clusters using Docker container "nodes".

- Installation

Installation

See Installing Kind.

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 February 06, 2020 at 4:23 AM PST by Add KIND as the options for spinning up a test kubernetes environment (#17860) (Page History)

Edit This Page

Container runtimes

FEATURE STATE: `Kubernetes v1.6` stable
This feature is *stable*, meaning:

Edit This Page

Installing kubeadm

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

- Before you begin
- Verify the MAC address and product_uuid are unique for every node
- Check network adapters
- Ensure iptables tooling does not use the nftables backend

Before you begin

- One or more machines running one of:
    - Ubuntu 16.04+
    - Debian 9+
    - CentOS 7
    - Red Hat Enterprise Linux (RHEL) 7
    - Fedora 25+
    - HypriotOS v1.0.1+
    - Container Linux (tested with 1800.6.0)
- 2 GB or more of RAM per machine (any less will leave little room for your apps)
- 2 CPUs or more
- Full network connectivity between all machines in the cluster (public or private network is fine)
- Unique hostname, MAC address, and product_uuid for every node. See here for more details.
- Certain ports are open on your machines. See here for more details.
- Swap disabled. You **MUST** disable swap in order for the kubelet to work properly.

Verify the MAC address and product_uuid are unique for every node

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

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

Check network adapters

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

Ensure iptables tooling does not use the nftables backend

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

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

- Debian or Ubuntu
- Fedora

```
# ensure legacy binaries are installed
sudo apt-get install -y iptables arptables ebtables

# switch to legacy versions
sudo update-alternatives --set iptables /usr/sbin/iptables-legacy
sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-
legacy
sudo update-alternatives --set arptables /usr/sbin/arptables-
legacy
sudo update-alternatives --set ebtables /usr/sbin/ebtables-legacy
```

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 --decode` command can be used to decode the certificate and `openssl x509 -text -noout` can be used for viewing the certificate information.

- Unset the `KUBECONFIG` environment variable using:

  ```
  unset KUBECONFIG
  ```

Or set it to the default `KUBECONFIG` location:

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

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

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

Default NIC When using flannel as the pod network in Vagrant

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

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

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

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

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

Non-public IP used for containers

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

```
Error from server: Get https://10.19.0.41:10250/containerLogs/
default/mysql-ddc65b868-glc5m/mysql: dial tcp 10.19.0.41:10250:
getsockopt: no route to host
```

- This may be due to Kubernetes using an IP that can not communicate with other IPs on the seemingly same subnet, possibly by policy of the machine provider.
- DigitalOcean 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 DigitalOcean 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 DigitalOcean, 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-
   ```

`/usr` is mounted read-only on nodes

On Linux distributions such as Fedora CoreOS, the directory `/usr` is mounted as a read-only filesystem. For flex-volume support, Kubernetes components like the kubelet and kube-controller-manager use the default path of `/usr/libexec/kubernetes/kubelet-plugins/volume/exec/`, yet the flex-volume directory *must be writeable* for the feature to work.

To workaround this issue you can configure the flex-volume directory using the kubeadm configuration file.

On the primary control-plane Node (created using `kubeadm init`) pass the following file using `--config`:

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: InitConfiguration
nodeRegistration:
  kubeletExtraArgs:
    volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/
volume/exec/"
---
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
controllerManager:
  extraArgs:
```

```
    flex-volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/volume/exec/"
```

On joining Nodes:

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: JoinConfiguration
nodeRegistration:
  kubeletExtraArgs:
    volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/volume/exec/"
```

Alternatively, you can modify `/etc/fstab` to make the `/usr` mount writeable, but please be advised that this is modifying a design principle of the Linux distribution.

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 a single control-plane cluster with kubeadm

The `kubeadm` tool helps you bootstrap a minimum viable Kubernetes cluster that conforms to best practices. In fact, you can use `kubeadm` to set up a cluster that will pass the Kubernetes Conformance tests. `kubeadm` also supports other cluster lifecycle functions, such as bootstrap tokens and cluster upgrades.

The `kubeadm` tool is good if you need:

- A simple way for you to try out Kubernetes, possibly for the first time.
- A way for existing users to automate setting up a cluster and test their application.
- A building block in other ecosystem and/or installer tools with a larger scope.

You can install and use `kubeadm` on various machines: your laptop, a set of cloud servers, a Raspberry Pi, and more. Whether you're deploying into the cloud or on-premises, you can integrate `kubeadm` into provisioning systems such as Ansible or Terraform.

Before you begin

To follow this guide, you need:

- One or more machines running a deb/rpm-compatible Linux OS; for example: Ubuntu or CentOS.
- 2 GiB or more of RAM per machine-any less leaves little room for your apps.
- At least 2 CPUs on the machine that you use as a control-plane node.
- Full network connectivity among all machines in the cluster. You can use either a public or a private network.

You also need to use a version of `kubeadm` that can deploy the version of Kubernetes that you want to use in your new cluster.

Kubernetes' version and version skew support policy applies to `kubeadm` as well as to Kubernetes overall. Check that policy to learn about what versions of Kubernetes and `kubeadm` are supported. This page is written for Kubernetes v1.17.

The `kubeadm` tool's overall feature state is General Availability (GA). Some sub-features 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.

> **Note:** Any commands under `kubeadm alpha` are, by definition, supported on an alpha level.

Objectives

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

Instructions

Installing kubeadm on your hosts

See "Installing kubeadm".

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

Initializing your control-plane node

The control-plane node is the machine where the control plane components run, including etcdConsistent and highly-available key value store used as Kubernetes' backing store for all cluster data. (the cluster database) and the API ServerControl plane component that serves the Kubernetes API. (which the kubectlA command line tool for communicating with a Kubernetes API server. command line tool communicates with).

1. (Recommended) If you have plans to upgrade this single control-plane `kubeadm` cluster to high availability you should specify the `--control-plane-endpoint` to set the shared endpoint for all control-plane nodes. Such an endpoint can be either a DNS name or an IP address of a load-balancer.
2. Choose a Pod network add-on, and verify whether it requires any arguments to be passed to `kubeadm init`. Depending on which third-party provider you choose, you might need to set the `--pod-network-cidr` to a provider-specific value. See Installing a Pod network add-on.
3. (Optional) Since version 1.14, `kubeadm` tries to detect the container runtime on Linux by using a list of well known domain socket paths. To use different container runtime or if there are more than one installed on the provisioned node, specify the `--cri-socket` argument to `kubeadm init`. See Installing runtime.
4. (Optional) Unless otherwise specified, `kubeadm` uses the network interface associated with the default gateway to set the advertise address for this particular control-plane node's API server. To use a different network interface, specify the `--apiserver-advertise-`

   `address=<ip-address>` argument to `kubeadm init`. To deploy an IPv6 Kubernetes cluster using IPv6 addressing, you must specify an IPv6 address, for example `--apiserver-advertise-address=fd00::101`
5. (Optional) Run `kubeadm config images pull` prior to `kubeadm init` to verify connectivity to the gcr.io container image registry.

To initialize the control-plane node run:

```
kubeadm init <args>
```

Considerations about apiserver-advertise-address and ControlPlaneEndpoint

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

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

Here is an example mapping:

```
192.168.0.102 cluster-endpoint
```

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

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

More information

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

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

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

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

If you join a node with a different architecture to your cluster, make sure that your deployed DaemonSets have container image support for this architecture.

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

```
[init] Using Kubernetes version: vX.Y.Z
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes
cluster
[preflight] This might take a minute or two, depending on the
speed of your internet connection
[preflight] You can also perform this action in beforehand using
'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to
file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/
kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names
[kubeadm-cp localhost] and IPs [10.138.0.4 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [kubeadm-
cp localhost] and IPs [10.138.0.4 127.0.0.1 ::1]
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubeadm-
cp kubernetes kubernetes.default kubernetes.default.svc
kubernetes.default.svc.cluster.local] and IPs [10.96.0.1
10.138.0.4]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-
controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/
kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the
control plane as static Pods from directory "/etc/kubernetes/
```

```
manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after
31.501735 seconds
[uploadconfig] storing the configuration used in ConfigMap
"kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-X.Y" in namespace
kube-system with the configuration for the kubelets in the
cluster
[patchnode] Uploading the CRI Socket information "/var/run/
dockershim.sock" to the Node API object "kubeadm-cp" as an
annotation
[mark-control-plane] Marking the node kubeadm-cp as control-
plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node kubeadm-cp as control-
plane by adding the taints [node-role.kubernetes.io/
master:NoSchedule]
[bootstrap-token] Using token: <token>
[bootstrap-token] Configuring bootstrap tokens, cluster-info
ConfigMap, RBAC Roles
[bootstraptoken] configured RBAC rules to allow Node Bootstrap
tokens to post CSRs in order for nodes to get long term
certificate credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover
controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate
rotation for all node client certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the
"kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

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

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

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

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

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

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

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

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

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

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

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

Installing a Pod network add-on

Caution:

This section contains important information about networking setup and deployment order. Read all of this advice carefully before proceeding.

You must deploy a Container Network InterfaceContainer network interface (CNI) plugins are a type of Network plugin that adheres to the appc/CNI specification. (CNI) based Pod network add-on so that your Pods can communicate with each other.
Cluster DNS (CoreDNS) will not start up before a network is installed.

- Take care that your Pod network must not overlap with any of the host networks: you are likely to see problems if there is any overlap.
(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 block to use instead, then use that during `kubeadm init` with `--pod-network-cidr` and as a replacement in your network plugin's YAML).

- By default, `kubeadm` sets up your cluster to use and enforce use of RBAC (role based access control).
Make sure that your Pod network plugin supports RBAC, and so do any manifests that you use to deploy it.

- If you want to use IPv6-either dual-stack, or single-stack IPv6 only networking-for your cluster, make sure that your Pod

network plugin supports IPv6.
IPv6 support was added to CNI in v0.6.0.

Several external projects provide Kubernetes Pod networks using CNI, some of which also support Network Policy.

See the list of available networking and network policy add-ons.

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

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

You can install only one Pod network per cluster. Below you can find installation instructions for some popular Pod network plugins:

- Calico
- Cilium
- Contiv-VPP
- Flannel
- Kube-router
- Weave Net

Calico is a networking and network policy provider. Calico supports a flexible set of networking options so you can choose the most efficient option for your situation, including non-overlay and overlay networks, with or without BGP. Calico uses the same engine to enforce network policy for hosts, pods, and (if using Istio & Envoy) applications at the service mesh layer. Calico works on several architectures, including `amd64`, `arm64`, and `ppc64le`.

By default, Calico uses `192.168.0.0/16` as the Pod network CIDR, though this can be configured in the calico.yaml file. For Calico to work correctly, you need to pass this same CIDR to the `kubeadm init` command using the `--pod-network-cidr=192.168.0.0/16` flag or via kubeadm's configuration.

```
kubectl apply -f https://docs.projectcalico.org/v3.11/manifests/
calico.yaml
```

Edit This Page

Customizing control plane configuration with kubeadm

FEATURE STATE: `Kubernetes 1.12` stable
This feature is *stable*, meaning:

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.

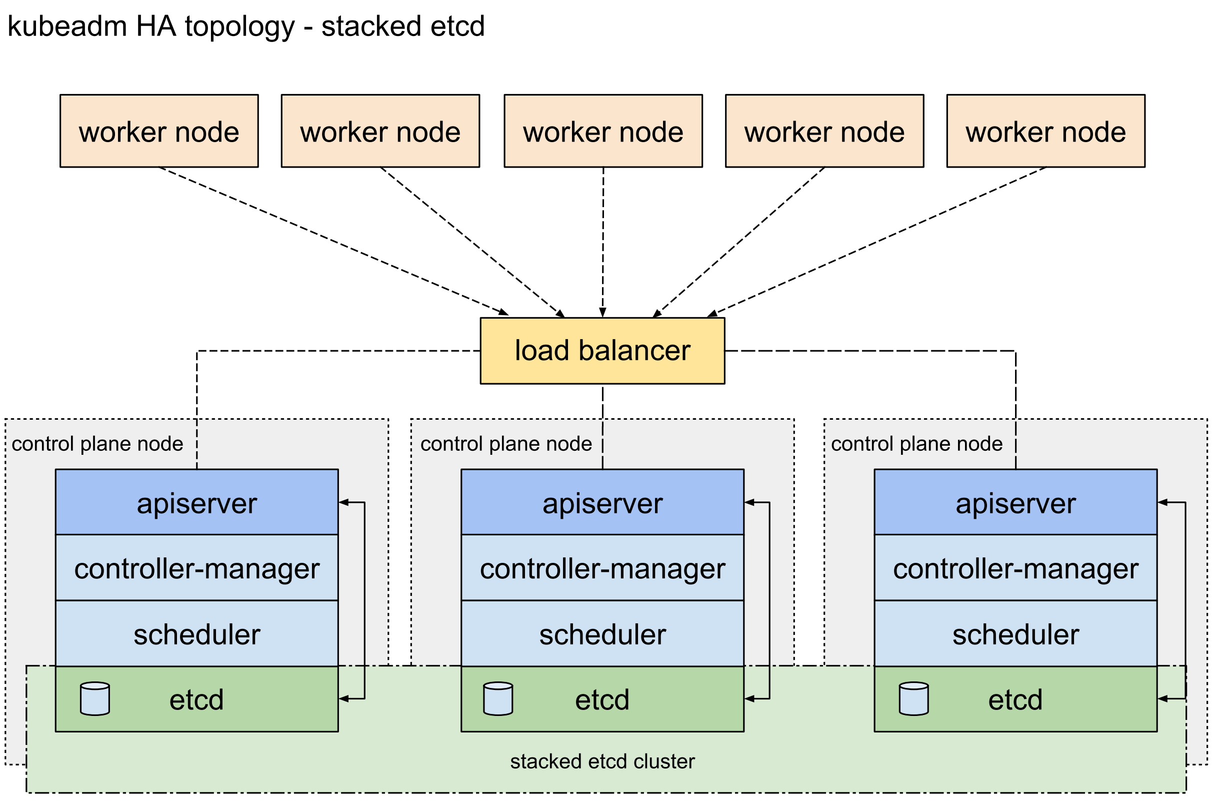
Each control plane node creates a local etcd member and this etcd member communicates only with the `kube-apiserver` of this node. The same applies to the local `kube-controller-manager` and `kube-scheduler` instances.

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`.



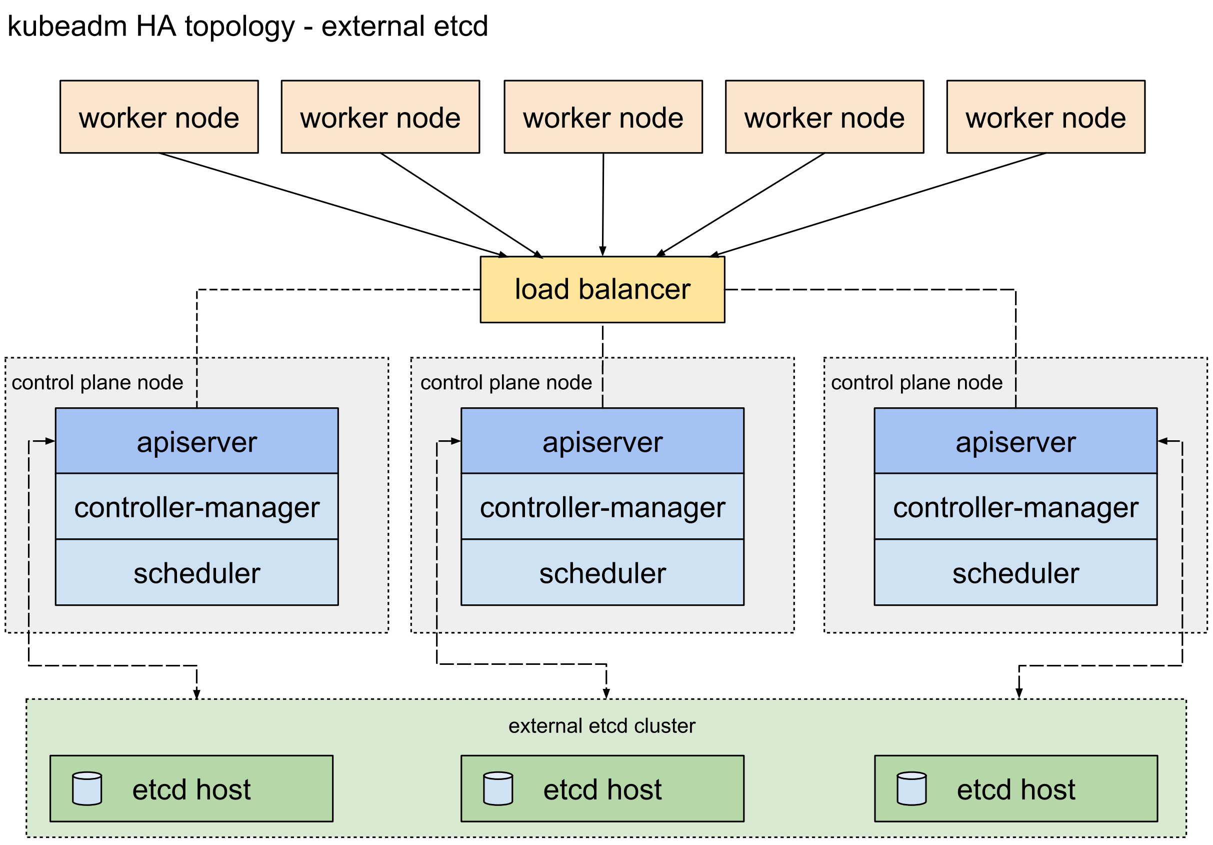
External etcd topology

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.



What's next

- Set up a highly available cluster with kubeadm

Feedback

Was this page helpful?

Yes No

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

Creating Highly Available clusters with kubeadm

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

- With stacked control plane nodes. This approach requires less infrastructure. The etcd members and control plane nodes are co-located.

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

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

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

See also The upgrade documentation.

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

Before you begin

For both methods you need this infrastructure:

- Three machines that meet kubeadm's minimum requirements for the masters
- Three machines that meet kubeadm's minimum requirements for the workers
- Full network connectivity between all machines in the cluster (public or private network)
- sudo privileges on all machines
- SSH access from one device to all nodes in the system
- `kubeadm` and `kubelet` installed on all machines. `kubectl` is optional.

For the external etcd cluster only, you also need:

- Three additional machines for etcd members

First steps for both methods

Create load balancer for kube-apiserver

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

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

    - In a cloud environment you should place your control plane nodes behind a TCP forwarding load balancer. This load balancer distributes traffic to all healthy control plane nodes in its target list. The health check for an apiserver is a TCP check on the port the kube-apiserver listens on (default value `:6443`).

    - It is not recommended to use an IP address directly in a cloud environment.

    - The load balancer must be able to communicate with all control plane nodes on the apiserver port. It must also allow incoming traffic on its listening port.

    - HAProxy can be used as a load balancer.

    - Make sure the address of the load balancer always matches the address of kubeadm's `ControlPlaneEndpoint`.

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

    ```
    nc -v LOAD_BALANCER_IP PORT
    ```

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

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

Stacked control plane and etcd nodes

Steps for the first control plane node

1. Initialize the control plane:

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

- You can use the `--kubernetes-version` flag to set the Kubernetes version to use. It is recommended that the versions of kubeadm, kubelet, kubectl and Kubernetes match.
- The `--control-plane-endpoint` flag should be set to the address or DNS and port of the load balancer.
- The `--upload-certs` flag is used to upload the certificates that should be shared across all the control-plane instances to the cluster. If instead, you prefer to copy certs across control-plane nodes manually or using automation tools, please remove this flag and refer to Manual certificate distribution section below.

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

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

- The output looks similar to:

  ```
  ...
  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
  sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720e
  ff5359e26aec866 --control-plane --certificate-key
  f8902e114ef118304e561c3ecd4d0b543adc226b7a07f675f5656418
  5ffe0c07

  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
  sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720e
  ff5359e26aec866
  ```

   - 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-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!

2. 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')"
   ```

3. 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
   sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff535
   ```

```
9e26aec866 --control-plane --certificate-key
f8902e114ef118304e561c3ecd4d0b543adc226b7a07f675f56564185ffe0
c07
```

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

External etcd nodes

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

Set up the etcd cluster

1. Follow these instructions to set up the etcd cluster.

2. Setup SSH as described here.

3. Copy the following files from any etcd node in the cluster to the first control plane node:

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

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

Set up the first control plane node

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

   ```
   apiVersion: kubeadm.k8s.io/v1beta2
   kind: ClusterConfiguration
   kubernetesVersion: stable
   controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
   etcd:
       external:
           endpoints:
           - https://ETCD_0_IP:2379
           - https://ETCD_1_IP:2379
           - https://ETCD_2_IP:2379
           caFile: /etc/kubernetes/pki/etcd/ca.crt
           certFile: /etc/kubernetes/pki/apiserver-etcd-
   ```

```
client.crt
        keyFile: /etc/kubernetes/pki/apiserver-etcd-
client.key
```

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

- Replace the following variables in the config template with the appropriate values for your cluster:
    - `LOAD_BALANCER_DNS`
    - `LOAD_BALANCER_PORT`
    - `ETCD_0_IP`
    - `ETCD_1_IP`
    - `ETCD_2_IP`

The following steps are similar to the stacked etcd setup:

1. Run `sudo kubeadm init --config kubeadm-config.yaml --upload-certs` on this node.

2. Write the output join commands that are returned to a text file for later use.

3. Apply the CNI plugin of your choice. The given example is for Weave Net:

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

Steps for the rest of the control plane nodes

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

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

Common tasks after bootstrapping control plane

Install workers

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

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

```
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26
aec866
```

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
    # Quote this line if you are using external etcd
    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.

5. 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/${USER}/etcd-ca.crt /etc/kubernetes/pki/etcd/ca.crt
# Quote this line if you are using external etcd
mv /home/${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 November 18, 2019 at 2:53 AM PST by Remove instructions to copy Etcd CA key in HA (#17611) (Page History)

Edit This Page

Set up a High Availability etcd cluster with kubeadm

> **Note:** While kubeadm is being used as the management tool for external etcd nodes in this guide, please note that kubeadm does not plan to support certificate rotation or upgrades for such nodes. The long term plan is to empower the tool etcdadm to manage these aspects.

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

Before you begin

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

Setting up the cluster

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

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

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

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

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

   systemctl daemon-reload
   systemctl restart kubelet
   ```

2. Create configuration files for kubeadm.

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

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

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

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

   for i in "${!ETCDHOSTS[@]}"; do
   HOST=${ETCDHOSTS[$i]}
   NAME=${NAMES[$i]}
   cat << EOF > /tmp/${HOST}/kubeadmcfg.yaml
   apiVersion: "kubeadm.k8s.io/v1beta2"
   kind: ClusterConfiguration
   etcd:
       local:
           serverCertSANs:
           - "${HOST}"
           peerCertSANs:
           - "${HOST}"
           extraArgs:
               initial-cluster: ${NAMES[0]}=https://$
   {ETCDHOSTS[0]}:2380,${NAMES[1]}=https://${ETCDHOSTS[1]}:
   2380,${NAMES[2]}=https://${ETCDHOSTS[2]}:2380
               initial-cluster-state: new
               name: ${NAME}
               listen-peer-urls: https://${HOST}:2380
               listen-client-urls: https://${HOST}:2379
               advertise-client-urls: https://${HOST}:2379
               initial-advertise-peer-urls: https://${HOST}:2380
   EOF
   done
   ```

3. Generate the certificate authority

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

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

```
kubeadm init phase certs etcd-ca
```

This creates two files

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

4. Create certificates for each member

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

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

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

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

5. Copy certificates and kubeadm configs

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

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

6. Ensure all expected files exist

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

```
/tmp/${HOST0}
â””â”€â”€ kubeadmcfg.yaml
---
/etc/kubernetes/pki
â”œâ”€â”€ apiserver-etcd-client.crt
â”œâ”€â”€ apiserver-etcd-client.key
â””â”€â”€ etcd
    â”œâ”€â”€ ca.crt
    â”œâ”€â”€ ca.key
    â”œâ”€â”€ healthcheck-client.crt
    â”œâ”€â”€ healthcheck-client.key
    â”œâ”€â”€ peer.crt
    â”œâ”€â”€ peer.key
    â”œâ”€â”€ server.crt
    â””â”€â”€ server.key
```

   On `$HOST1`:

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

   On `$HOST2`

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

```
â””â”€â”€ etcd
    â”œâ”€â”€ ca.crt
    â”œâ”€â”€ healthcheck-client.crt
    â”œâ”€â”€ healthcheck-client.key
    â”œâ”€â”€ peer.crt
    â”œâ”€â”€ peer.key
    â”œâ”€â”€ server.crt
    â””â”€â”€ server.key
```

7. Create the static pod manifests

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

   ```
   root@HOST0 $ kubeadm init phase etcd local --config=/tmp/${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 k8s.gcr.io/etcd:${ETCD_TAG} etcdctl \
   --cert /etc/kubernetes/pki/etcd/peer.crt \
   --key /etc/kubernetes/pki/etcd/peer.key \
   --cacert /etc/kubernetes/pki/etcd/ca.crt \
   --endpoints https://${HOST0}:2379 endpoint health --cluster
   ...
   https://[HOST0 IP]:2379 is healthy: successfully committed proposal: took = 16.283339ms
   https://[HOST1 IP]:2379 is healthy: successfully committed proposal: took = 19.44402ms
   https://[HOST2 IP]:2379 is healthy: successfully committed proposal: took = 35.926451ms
   ```

   - Set `${ETCD_TAG}` to the version tag of your etcd image. For example `3.4.3-0`. To see the etcd image and tag that kubeadm uses execute `kubeadm config images list --kubernetes-version ${K8S_VERSION}`, where `${K8S_VERSION}` is for example `v1.17.0`
   - 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.

Create an Issue Edit This Page
Page last modified on January 03, 2020 at 11:43 AM PST by Update cluster health check command for newer etcd (#18392) (Page History)

Edit This Page

Configuring each kubelet in your cluster using kubeadm

FEATURE STATE: `Kubernetes 1.11` stable
This feature is *stable*, meaning:

Edit This Page

Configuring your kubernetes cluster to self-host the control plane

Self-hosting the Kubernetes control plane

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

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

Caveats

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

1. Self-hosting in 1.8 and later has some important limitations. In particular, a self-hosted cluster *cannot recover from a reboot of the control-plane node* without manual intervention.
2. By default, self-hosted control plane Pods rely on credentials loaded from `hostPath` volumes. Except for initial creation, these credentials are not managed by kubeadm.
3. The self-hosted portion of the control plane does not include etcd, which still runs as a static Pod.

Process

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

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

1. Waits for this bootstrap static control plane to be running and healthy. This is identical to the `kubeadm init` process without self-hosting.
2. Uses the static control plane Pod manifests to construct a set of DaemonSet manifests that will run the self-hosted control plane. It also modifies these manifests where necessary, for example adding new volumes for secrets.
3. Creates DaemonSets in the `kube-system` namespace and waits for the resulting Pods to be running.
4. Once self-hosted Pods are operational, their associated static Pods are deleted and kubeadm moves on to install the next component. This triggers kubelet to stop those static Pods.
5. When the original static control plane stops, the new self-hosted control plane is able to bind to listening ports and become active.

Feedback

Was this page helpful?

Yes No

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

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

Edit This Page

Installing Kubernetes with kops

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

kops is an automated 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

- Before you begin
- Creating a cluster
- Cleanup
- What's next

Before you begin

- You must have kubectl installed.
- You must install `kops` on a 64-bit (AMD64 and Intel 64) device architecture.
- You must have an AWS account, generate IAM keys and configure them.

Creating a cluster

(1/5) Install kops

Installation

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

- macOS
- Linux

Download the latest release with the command:

```
curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name | cut -d '"' -f 4)/kops-darwin-amd64
```

To download a specific version, replace the

```
$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name | cut -d '"' -f 4)
```

portion of the command with the specific version.

For example, to download kops version v1.15.0 type:

```
curl -LO  https://github.com/kubernetes/kops/releases/download/1.15.0/kops-darwin-amd64
```

Make the kops binary executable.

```
chmod +x kops-darwin-amd64
```

Move the kops binary in to your PATH.

```
sudo mv kops-darwin-amd64 /usr/local/bin/kops
```

You can also install kops using Homebrew.

```
brew update && brew install kops
```

Edit This Page

Installing Kubernetes with KRIB

Overview

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

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

> **Note:** KRIB is not a *stand-alone* installer: Digital Rebar templates drive a standard *kubeadm* configuration that manages the Kubernetes installation with the Digital Rebar cluster pattern to elect leaders *without external supervision*.

KRIB features:

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

- support for Ubuntu 18.04, CentOS/RHEL 7, CoreOS, RancherOS and others

Creating a cluster

Review Digital Rebar documentation for details about installing the platform.

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

(1/5) Discover servers

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

(2/5) Install KRIB Content and Certificate Plugin

Upload the KRIB Content bundle (or build from source) and the Cert Plugin for your DRP platform. Both are freely available via the RackN UX or using the upload from catalog feature of the DRPCLI (shown below).

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

(3/5) Start your cluster deployment

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

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

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

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

(4/5) Monitor your cluster deployment

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

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

(5/5) Access your cluster

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

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

::

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

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

Cluster operations

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

Scale your cluster

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

Cleanup your cluster (for developers)

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

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

Feedback

- Slack Channel: #community
- GitHub Issues

Feedback

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
    - DockerDocker is a software technology providing operating-system-level virtualization also known as containers.
    - rktA security-minded, standards-based container engine.
    - CRI-OA lightweight container runtime specifically for Kubernetes
- 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.

Running Kubernetes on Alibaba Cloud

Alibaba Cloud Container Service

The Alibaba Cloud Container Service lets you run and manage Docker applications on a cluster of either Alibaba Cloud ECS instances or in a Serverless fashion. 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.

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 November 21, 2019 at 10:11 AM PST by fix-up 404 urls (#17668) (Page History)

Edit This Page

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

Edit This Page

Running Kubernetes on Azure

Azure Kubernetes Service (AKS)

The Azure Kubernetes Service offers simple deployments for Kubernetes clusters.

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

Microsoft Azure Kubernetes Service

Custom Deployments: AKS-Engine

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

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

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

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

CoreOS Tectonic for Azure

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

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

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

Feedback

Was this page helpful?

Yes No

Running Kubernetes on CenturyLink Cloud

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

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

Find Help

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

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

Clusters of VMs or Physical Servers, your choice.

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

Requirements

The requirements to run this script are:

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

Script Installation

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

1) Clone this repository and cd into it.

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

2) Install all requirements, including

- Ansible
- CenturyLink Cloud SDK

- Ansible Modules

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

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

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

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

Script Installation Example: Ubuntu 14 Walkthrough

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

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

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

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

Cluster Creation

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

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

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

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

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

Cluster Creation: Script Options

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

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

     -h (--help)                   display this help and exit
     -c= (--clc_cluster_name=)     set the name of the cluster,
as used in CLC group names
     -t= (--minion_type=)          standard -> VM (default),
bareMetal -> physical]
     -d= (--datacenter=)           VA1 (default)
     -m= (--minion_count=)          number of kubernetes minion
nodes
     -mem= (--vm_memory=)           number of GB ram for each
minion
     -cpu= (--vm_cpu=)              number of virtual cps for
each minion node
     -phyid= (--server_conf_id=)    physical server configuration
id, one of

physical_server_20_core_conf_id

physical_server_12_core_conf_id

physical_server_4_core_conf_id (default)
     -etcd_separate_cluster=yes     create a separate cluster of
three etcd nodes,
                                    otherwise run etcd on the
master node
```

Cluster Expansion

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

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

Cluster Expansion: Script Options

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

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

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

Cluster Deletion

There are two ways to delete an existing cluster:

1) Use our python script:

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

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

Examples

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

```
bash kube-up.sh --clc_cluster_name=k8s_1 --minion_type=bareMetal
--minion_count=3 --datacenter=VA1
```

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

```
bash kube-up.sh --clc_cluster_name=k8s_2 --minion_type=standard
--minion_count=6 --datacenter=VA1 --etcd_separate_cluster=yes
```

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

```
bash kube-up.sh --clc_cluster_name=k8s_3 --minion_type=standard
--minion_count=10 --datacenter=VA1 -mem=6 -cpu=4
```

Cluster Features and Architecture

We configure the Kubernetes cluster with the following features:

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

We use the following to create the Kubernetes cluster:

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

Optional add-ons

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

Cluster management

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

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

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

Accessing the cluster programmatically

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

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

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

-
  ```
  * ```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:
  ```

  shell

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

```

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

```

shell kubectl proxy -p 8001 ```

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

What Kubernetes features do not work on CenturyLink Cloud

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

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

Ansible Files

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

Further reading

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

Feedback

Was this page helpful?

Yes No

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

Running Kubernetes on Google Compute Engine

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

Before you begin

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

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

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

Prerequisites

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

Starting a cluster

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

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

or

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

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

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

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

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

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

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

If you run into trouble, please see the section on troubleshooting, post to the Kubernetes Forum, or come ask questions on Slack.

The next few steps will show you:

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

Installing the Kubernetes command line tools on your workstation

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

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

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

```
gcloud components install kubectl
```

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

Getting started with your cluster

Inspect your cluster

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

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

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

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

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

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

command.

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

```
NAMESPACE     NAME
READY     STATUS    RESTARTS   AGE
kube-system   coredns-5f4fbb68df-mc8z8                        1/
1       Running   0          15m
kube-system   fluentd-cloud-logging-kubernetes-minion-63uo   1/
1       Running   0          14m
kube-system   fluentd-cloud-logging-kubernetes-minion-c1n9   1/
```

```
1       Running   0          14m
kube-system   fluentd-cloud-logging-kubernetes-minion-c4og   1/
1       Running   0          14m
kube-system   fluentd-cloud-logging-kubernetes-minion-ngua   1/
1       Running   0          14m
kube-system   kube-ui-v1-curt1                               1/
1       Running   0          15m
kube-system   monitoring-heapster-v5-ex4u3                   1/
1       Running   1          15m
kube-system   monitoring-influx-grafana-v1-piled             2/
2       Running   0          15m
```

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

Run some examples

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

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

Tearing down the cluster

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

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

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

Customizing

The script above relies on Google Storage to stage the Kubernetes release. It then will start (by default) a single master VM along with 3 worker VMs. You can tweak some of these parameters by editing `kubernetes/cluster/gce/config-default.sh` You can view a transcript of a successful cluster creation here.

Troubleshooting

Project settings

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

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

Cluster initialization hang

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

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

SSH

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

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

Networking

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

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

Support Level

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

Further reading

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

Feedback

Was this page helpful?

Yes No

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

Running Kubernetes on Multiple Clouds with 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.

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 Tencent Kubernetes Engine

Tencent Kubernetes Engine

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

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

Custom Deployment

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

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

What's Next

To learn more, see the TKE documentation.

Feedback

Was this page helpful?

Yes No

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

Create an Issue Edit This Page
Page last modified on July 29, 2019 at 4:59 PM PST by Modify document format. (#15567) (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.

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

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

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
```

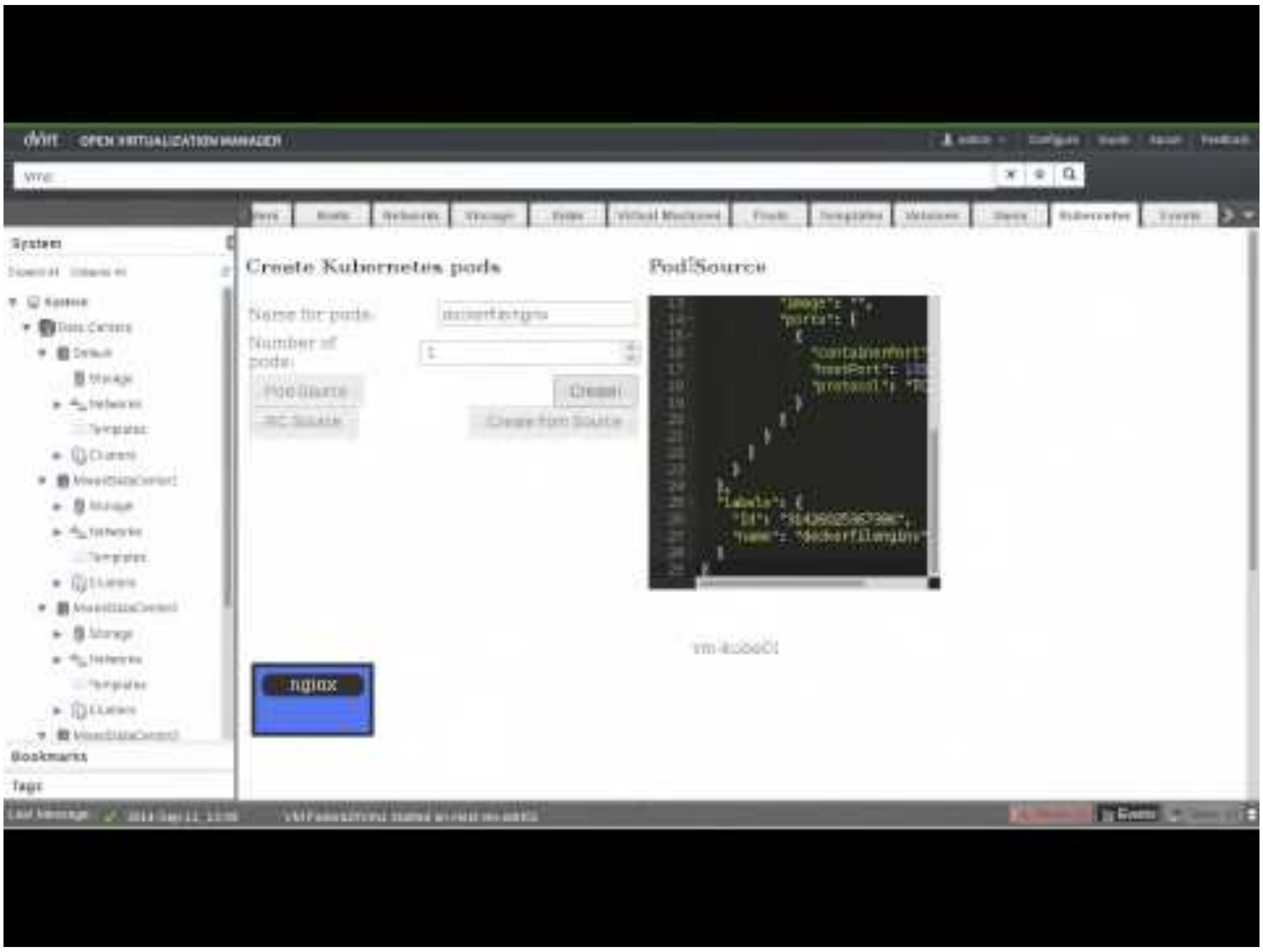
In the above example all the VMs tagged with the `kubernetes` label will be reported as nodes to Kubernetes.

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.



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.

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. You may not deploy Windows and Linux containers in the same Pod. All containers in a Pod are scheduled onto a single Node where each Node represents a specific platform and architecture. The following Pod capabilities, properties and events are supported with Windows containers:

  - Single or multiple containers per Pod with process isolation and volume sharing

    - Pod status fields
    - Readiness and Liveness probes
    - postStart & preStop container lifecycle events
    - ConfigMap, Secrets: as environment variables or volumes
    - EmptyDir
    - Named pipe host mounts
    - Resource limits

- Controllers

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

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

- Services

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

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

Pods, Controllers and Services are critical elements to managing Windows workloads on Kubernetes. However, on their own they are not enough to enable the proper lifecycle management of Windows workloads in a dynamic cloud native environment. We added support for the following features:

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

Container Runtime

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

Persistent Storage

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

In-tree Volume Plugins

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

- awsElasticBlockStore
- azureDisk
- azureFile
- gcePersistentDisk
- vsphereVolume

FlexVolume Plugins

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

- SMB
- iSCSI

CSI Plugins

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

Edit This Page

Guide for adding Windows Nodes in Kubernetes

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

Objectives

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

Before you begin

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

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

Getting Started: Adding a Windows Node to Your Cluster

Plan IP Addressing

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

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

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

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

Components that run on Windows

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

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

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

Networking Configuration

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

Configuring Flannel in VXLAN mode on the Linux control-plane

1. Prepare Kubernetes master for Flannel

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

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

2. Download & configure Flannel

   Download the most recent Flannel manifest:

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

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

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

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

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

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

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

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

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

4. Apply the Flannel manifest and validate

   Let's apply the Flannel configuration:

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

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

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

   The output looks like as follows:

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

   Verify that the Flannel DaemonSet has the NodeSelector applied.

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

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

   ```
   NAME              DESIRED   CURRENT   READY   UP-TO-DATE
   AVAILABLE   NODE
   SELECTOR                                                AGE
   kube-flannel-ds   2         2         2       2
   ```

```
2           beta.kubernetes.io/arch=amd64,beta.kubernetes.io/
os=linux   21d
kube-proxy        2         2         2       2
2           beta.kubernetes.io/
os=linux                                 26d
```

Join Windows Worker Node

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

Preparing a Windows Node

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

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

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityPr
otocolType]::Tls12
Start-BitsTransfer https://github.com/kubernetes-sigs/sig-
windows-tools/archive/master.zip
tar -xvf .\master.zip --strip-components 3 sig-windows-tools-
master/kubeadm/v1.15.0/*
Remove-Item .\master.zip
```

2. Customize the Kubernetes configuration file

```
{
"Cri" : {  // Contains values for container runtime and base
container setup
    "Name" : "dockerd", // Container runtime name
    "Images" : {
        "Pause" : "mcr.microsoft.com/k8s/core/pause:
1.2.0",  // Infrastructure container image
        "Nanoserver" : "mcr.microsoft.com/windows/nanoserver:
1809",  // Base Nanoserver container image
        "ServerCore" : "mcr.microsoft.com/windows/
servercore:ltsc2019"  // Base ServerCore container image
    }
},
"Cni" : {  // Contains values for networking executables
    "Name" : "flannel",  // Name of network fabric
    "Source" : [{ // Contains array of objects containing
values for network daemon(s)
        "Name" : "flanneld",  // Name of network daemon
        "Url" : "https://github.com/coreos/flannel/releases/
download/v0.11.0/flanneld.exe"  // Direct URL pointing to
network daemon executable
```

```
        }
    ],
    "Plugin" : {  // Contains values for CNI network plugin
        "Name": "vxlan" // Backend network mechanism to use:
["vxlan" | "bridge"]
    },
    "InterfaceName" : "Ethernet"  // Designated network
interface name on Windows node to use as container network
},
"Kubernetes" : {  // Contains values for Kubernetes node
binaries
    "Source" : {  // Contains values for Kubernetes node
binaries
        "Release" : "1.15.0",  // Version of Kubernetes node
binaries
        "Url" : "https://dl.k8s.io/v1.15.0/kubernetes-node-
windows-amd64.tar.gz"  // Direct URL pointing to Kubernetes
node binaries tarball
    },
    "ControlPlane" : {  // Contains values associated with
Kubernetes control-plane ("Master") node
        "IpAddress" : "kubemasterIP",  // IP address of
control-plane ("Master") node
        "Username" : "localadmin",  // Username on control-
plane ("Master") node with remote SSH access
        "KubeadmToken" : "token",  // Kubeadm bootstrap token
        "KubeadmCAHash" : "discovery-token-ca-cert-hash"  //
Kubeadm CA key hash
    },
    "KubeProxy" : {  // Contains values for Kubernetes
network proxy configuration
        "Gates" : "WinOverlay=true"  // Comma-separated key-
value pairs passed to kube-proxy feature gate flag
    },
    "Network" : {  // Contains values for IP ranges in CIDR
notation for Kubernetes networking
        "ServiceCidr" : "10.96.0.0/12",  // Service IP
subnet used by Services in CIDR notation
        "ClusterCidr" : "10.244.0.0/16"  // Cluster IP
subnet used by Pods in CIDR notation
    }
},
"Install" : {  // Contains values and configurations for
Windows node installation
    "Destination" : "C:\\ProgramData\\Kubernetes"  //
Absolute DOS path where Kubernetes will be installed on the
Windows node
}
}
```

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

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

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

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

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

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

alt_text

On the Windows node you target, this step will:

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

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

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

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

Join the Windows Node to the Kubernetes cluster

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

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

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

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

alt_text

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

This step will perform the following actions:

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

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

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

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

```
kubectl get nodes
```

Remove the Windows Node from the Kubernetes cluster

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

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

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

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

alt_text

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

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

Public Cloud Providers

Azure

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

GCP

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

Deployment with kubeadm and cluster API

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

Next Steps

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

Feedback

Was this page helpful?

Yes No

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

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: apps/v1
kind: Deployment
metadata:
  labels:
    app: win-webserver
  name: win-webserver
spec:
  replicas: 2
  selector:
    matchLabels:
      app: win-webserver
  template:
    metadata:
      labels:
        app: win-webserver
      name: win-webserver
    spec:
     containers:
      - name: windowswebserver
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        command:
        - powershell.exe
        - -command
        - "<#code used from https://gist.github.com/
wagnerandrade/5424431#> ; $$listener = New-Object
System.Net.HttpListener ; $$listener.Prefixes.Add('http://*:
80/') ; $$listener.Start() ; $$callerCounts = @{} ; Write-
Host('Listening at http://*:80/') ; while ($
$listener.IsListening) { ;$$context = $$listener.GetContext() ;$
$requestUrl = $$context.Request.Url ;$$clientIP = $
$context.Request.RemoteEndPoint.Address ;$$response = $
$context.Response ;Write-Host '' ;Write-Host('> {0}' -f $
$requestUrl) ;  ;$$count = 1 ;$$k=$$callerCounts.Get_Item($
$clientIP) ;if ($$k -ne $$null) { $$count += $$k } ;$
$callerCounts.Set_Item($$clientIP, $$count) ;$$ip=(Get-
NetAdapter | Get-NetIpAddress); $
$header='<html><body><H1>Windows Container Web Server</H1>' ;$
$callerCountsString='' ;$$callerCounts.Keys | % { $
$callerCountsString+='<p>IP {0} callerCount {1} ' -f $
$ip[1].IPAddress,$$callerCounts.Item($$_) } ;$$footer='</body></
html>' ;$$content='{0}{1}{2}' -f $$header,$$callerCountsString,$
$footer ;Write-Output $$content ;$$buffer =
[System.Text.Encoding]::UTF8.GetBytes($$content) ;$
$response.ContentLength64 = $$buffer.Length ;$
$response.OutputStream.Write($$buffer, 0, $$buffer.Length) ;$
$response.Close() ;$$responseStatus = $
$response.StatusCode ;Write-Host('< {0}' -f $
$responseStatus)  } ; "
     nodeSelector:
      beta.kubernetes.io/os: windows
```

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

1. Check that all nodes are healthy:

```
kubectl get nodes
```

2. Deploy the service and watch for pod updates:

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

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

3. Check that the deployment succeeded. To verify:

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

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

Using configurable Container usernames

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

Managing Workload Identity with Group Managed Service Accounts

Starting with Kubernetes v1.14, Windows container workloads can be configured to use Group Managed Service Accounts (GMSA). Group Managed Service Accounts are a specific type of Active Directory account that provides automatic password management, simplified service principal name (SPN) management, and the ability to delegate the management to other administrators across multiple servers. Containers configured with a

GMSA can access external Active Directory Domain resources while carrying the identity configured with the GMSA. Learn more about configuring and using GMSA for Windows containers here.

Taints and Tolerations

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

Ensuring OS-specific workloads land on the appropriate container host

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

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

If a Pod specification does not specify a nodeSelector like `"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=windows: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:
    kubernetes.io/os: windows
    node.kubernetes.io/windows-build: '10.0.17763'
tolerations:
    - key: "os"
      operator: "Equal"
      value: "windows"
      effect: "NoSchedule"
```

Handling multiple Windows versions in the same cluster

The Windows Server version used by each pod must match that of the node. If you want to use multiple Windows Server versions in the same cluster, then you should set additional node labels and nodeSelectors.

Kubernetes 1.17 automatically adds a new label `node.kubernetes.io/windows-build` to simplify this. If you're running an older version, then it's recommended to add this label manually to Windows nodes.

This label reflects the Windows major, minor, and build number that need to match for compatibility. Here are values used today for each Windows Server version.

Product Name	Build Number(s)
Windows Server 2019	10.0.17763
Windows Server version 1809	10.0.17763
Windows Server version 1903	10.0.18362

Simplifying with RuntimeClass

[RuntimeClass] can be used to simplify the process of using taints and tolerations. A cluster administrator can create a `RuntimeClass` object which is used to encapsulate these taints and tolerations.

1. Save this file to `runtimeClasses.yml`. It includes the appropriate `nodeSelector` for the Windows OS, architecture, and version.

   ```
   apiVersion: node.k8s.io/v1beta1
   kind: RuntimeClass
   metadata:
   name: windows-2019
   handler: 'docker'
   scheduling:
   nodeSelector:
   kubernetes.io/os: 'windows'
   kubernetes.io/arch: 'amd64'
   node.kubernetes.io/windows-build: '10.0.17763'
   tolerations:
   - effect: NoSchedule
   key: os
   operator: Equal
   value: "windows"
   ```

2. Run `kubectl create -f runtimeClasses.yml` using as a cluster administrator

3. Add `runtimeClassName: windows-2019` as appropriate to Pod specs

For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iis-2019
  labels:
    app: iis-2019
spec:
  replicas: 1
  template:
    metadata:
      name: iis-2019
      labels:
        app: iis-2019
    spec:
      runtimeClassName: windows-2019
      containers:
      - name: iis
        image: mcr.microsoft.com/windows/servercore/
iis:windowsservercore-ltsc2019
        resources:
          limits:
            cpu: 1
            memory: 800Mi
          requests:
            cpu: .1
            memory: 300Mi
        ports:
          - containerPort: 80
 selector:
    matchLabels:
      app: iis-2019
---
apiVersion: v1
kind: Service
metadata:
  name: iis
spec:
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 80
  selector:
    app: iis-2019
```

Feedback

Was this page helpful?

Yes No

Running in multiple zones

This page describes how to run a cluster in multiple zones.

Introduction

Kubernetes 1.2 adds support for running a single cluster in multiple failure zones (GCE calls them simply "zones", AWS calls them "availability zones", here we'll refer to them as "zones"). This is a lightweight version of a broader Cluster Federation feature (previously referred to by the affectionate nickname "Ubernetes"). Full Cluster Federation allows combining separate Kubernetes clusters running in different regions or cloud providers (or on-premises data centers). However, many users simply want to run a more available Kubernetes cluster in multiple zones of their single cloud provider, and this is what the multizone support in 1.2 allows (this previously went by the nickname "Ubernetes Lite").

Multizone support is deliberately limited: a single Kubernetes cluster can run in multiple zones, but only within the same region (and cloud provider). Only GCE and AWS are currently supported automatically (though it is easy to add similar support for other clouds or even bare metal, by simply arranging for the appropriate labels to be added to nodes and volumes).

Functionality

When nodes are started, the kubelet automatically adds labels to them with zone information.

Kubernetes will automatically spread the pods in a replication controller or service across nodes in a single-zone cluster (to reduce the impact of failures.) With multiple-zone clusters, this spreading behavior is extended across zones (to reduce the impact of zone failures.) (This is achieved via `SelectorSpreadPriority`). This is a best-effort placement, and so if the zones in your cluster are heterogeneous (e.g. different numbers of nodes, different types of nodes, or different pod resource requirements), this might prevent perfectly even spreading of your pods across zones. If desired, you can use homogeneous zones (same number and types of nodes) to reduce the probability of unequal spreading.

When persistent volumes are created, the `PersistentVolumeLabel` admission controller automatically adds zone labels to them. The scheduler (via the `VolumeZonePredicate` predicate) will then ensure that pods that claim a given volume are only placed into the same zone as that volume, as volumes cannot be attached across zones.

Limitations

There are some important limitations of the multizone support:

- We assume that the different zones are located close to each other in the network, so we don't perform any zone-aware routing. In particular, traffic that goes via services might cross zones (even if some pods backing that service exist in the same zone as the client), and this may incur additional latency and cost.
- Volume zone-affinity will only work with a `PersistentVolume`, and will not work if you directly specify an EBS volume in the pod spec (for example).
- Clusters cannot span clouds or regions (this functionality will require full federation support).
- Although your nodes are in multiple zones, kube-up currently builds a single master node by default. While services are highly available and can tolerate the loss of a zone, the control plane is located in a single zone. Users that want a highly available control plane should follow the high availability instructions.

Volume limitations

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

- StatefulSet volume zone spreading when using dynamic provisioning is currently not compatible with pod affinity or anti-affinity policies.
- If the name of the StatefulSet contains dashes ("-"), volume zone spreading may not provide a uniform distribution of storage across zones.
- When specifying multiple PVCs in a Deployment or Pod spec, the StorageClass needs to be configured for a specific single zone, or the PVs need to be statically provisioned in a specific zone. Another workaround is to use a StatefulSet, which will ensure that all the volumes for a replica are provisioned in the same zone.

Walkthrough

We're now going to walk through setting up and using a multi-zone cluster on both GCE & AWS. To do so, you bring up a full cluster (specifying `MULTIZONE=true`), and then you add nodes in additional zones by running `kube-up` again (specifying `KUBE_USE_EXISTING_MASTER=true`).

Bringing up your cluster

Create the cluster as normal, but pass MULTIZONE to tell the cluster to manage multiple zones; creating nodes in us-central1-a.

GCE:

```
curl -sS https://get.k8s.io | MULTIZONE=true
KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-central1-a NUM_NODES=3
bash
```

AWS:

```
curl -sS https://get.k8s.io | MULTIZONE=true
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2a NUM_NODES=3 bash
```

This step brings up a cluster as normal, still running in a single zone (but `MULTIZONE=true` has enabled multi-zone capabilities).

Nodes are labeled

View the nodes; you can see that they are labeled with zone information. They are all in `us-central1-a` (GCE) or `us-west-2a` (AWS) so far. The labels are `failure-domain.beta.kubernetes.io/region` for the region, and `failure-domain.beta.kubernetes.io/zone` for the zone:

```
kubectl get nodes --show-labels
```

The output is similar to this:

```
NAME                     STATUS                     ROLES
AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-1,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-master
kubernetes-minion-87j9   Ready                      <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-87j9
kubernetes-minion-9vlv   Ready                      <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-9vlv
kubernetes-minion-a12q   Ready                      <none>
6m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-a12q
```

Add more nodes in a second zone

Let's add another set of nodes to the existing cluster, reusing the existing master, running in a different zone (us-central1-b or us-west-2b). We run kube-up again, but by specifying `KUBE_USE_EXISTING_MASTER=true` kube-up will not create a new master, but will reuse one that was previously created instead.

GCE:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true
KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-central1-b NUM_NODES=3
kubernetes/cluster/kube-up.sh
```

On AWS we also need to specify the network CIDR for the additional subnet, along with the master internal IP address:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2b NUM_NODES=3 KUBE
_SUBNET_CIDR=172.20.1.0/24 MASTER_INTERNAL_IP=172.20.0.9
kubernetes/cluster/kube-up.sh
```

View the nodes again; 3 more nodes should have launched and be tagged in us-central1-b:

```
kubectl get nodes --show-labels
```

The output is similar to this:

```
NAME                     STATUS                     ROLES
AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>
16m   v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-1,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-master
kubernetes-minion-281d   Ready                      <none>
2m    v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-281d
kubernetes-minion-87j9   Ready                      <none>
16m   v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-87j9
kubernetes-minion-9vlv   Ready                      <none>
16m   v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-9vlv
kubernetes-minion-a12q   Ready                      <none>
17m   v1.13.0          beta.kubernetes.io/instance-type=n1-
```

```
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-a12q
kubernetes-minion-pp2f   Ready                      <none>
2m    v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-pp2f
kubernetes-minion-wf8i   Ready                      <none>
2m    v1.13.0           beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-wf8i
```

Volume affinity

Create a volume using the dynamic volume creation (only PersistentVolumes are supported for zone affinity):

```
kubectl apply -f - <<EOF
{
  "apiVersion": "v1",
  "kind": "PersistentVolumeClaim",
  "metadata": {
    "name": "claim1",
    "annotations": {
        "volume.alpha.kubernetes.io/storage-class": "foo"
    }
  },
  "spec": {
    "accessModes": [
      "ReadWriteOnce"
    ],
    "resources": {
      "requests": {
        "storage": "5Gi"
      }
    }
  }
}
EOF
```

> **Note:** For version 1.3+ Kubernetes will distribute dynamic PV claims across the configured zones. For version 1.2, dynamic persistent volumes were always created in the zone of the cluster master (here us-central1-a / us-west-2a); that issue (#23330) was addressed in 1.3+.

Now let's validate that Kubernetes automatically labeled the zone & region the PV was created in.

```
kubectl get pv --show-labels
```

The output is similar to this:

```
NAME           CAPACITY   ACCESSMODES   RECLAIM POLICY
STATUS    CLAIM            STORAGECLASS    REASON    AGE
LABELS
pv-gce-mj4gm   5Gi        RWO           Retain
Bound     default/claim1   manual                    46s
failure-domain.beta.kubernetes.io/region=us-central1,failure-
domain.beta.kubernetes.io/zone=us-central1-a
```

So now we will create a pod that uses the persistent volume claim. Because GCE PDs / AWS EBS volumes cannot be attached across zones, this means that this pod can only be created in the same zone as the volume:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: claim1
EOF
```

Note that the pod was automatically created in the same zone as the volume, as cross-zone attachments are not generally permitted by cloud providers:

```
kubectl describe pod mypod | grep Node
```

```
Node:        kubernetes-minion-9vlv/10.240.0.5
```

And check node labels:

```
kubectl get node kubernetes-minion-9vlv --show-labels
```

```
NAME                     STATUS    AGE    VERSION          LABELS
kubernetes-minion-9vlv   Ready     22m    v1.6.0+fff5156
beta.kubernetes.io/instance-type=n1-standard-2,failure-
domain.beta.kubernetes.io/region=us-central1,failure-
domain.beta.kubernetes.io/zone=us-central1-a,kubernetes.io/
hostname=kubernetes-minion-9vlv
```

Pods are spread across zones

Pods in a replication controller or service are automatically spread across zones. First, let's launch more nodes in a third zone:

GCE:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true
KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-central1-f NUM_NODES=3
kubernetes/cluster/kube-up.sh
```

AWS:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2c NUM_NODES=3 KUBE
_SUBNET_CIDR=172.20.2.0/24 MASTER_INTERNAL_IP=172.20.0.9
kubernetes/cluster/kube-up.sh
```

Verify that you now have nodes in 3 zones:

```
kubectl get nodes --show-labels
```

Create the guestbook-go example, which includes an RC of size 3, running a simple web app:

```
find kubernetes/examples/guestbook-go/ -name '*.json' | xargs -I
{} kubectl apply -f {}
```

The pods should be spread across all 3 zones:

```
kubectl describe pod -l app=guestbook | grep Node
```

```
Node:       kubernetes-minion-9vlv/10.240.0.5
Node:       kubernetes-minion-281d/10.240.0.8
Node:       kubernetes-minion-olsh/10.240.0.11
```

```
kubectl get node kubernetes-minion-9vlv kubernetes-minion-281d
kubernetes-minion-olsh --show-labels
```

```
NAME                     STATUS    ROLES    AGE
VERSION          LABELS
kubernetes-minion-9vlv   Ready     <none>   34m
v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
a,kubernetes.io/hostname=kubernetes-minion-9vlv
kubernetes-minion-281d   Ready     <none>   20m
v1.13.0          beta.kubernetes.io/instance-type=n1-
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
b,kubernetes.io/hostname=kubernetes-minion-281d
kubernetes-minion-olsh   Ready     <none>   3m
v1.13.0          beta.kubernetes.io/instance-type=n1-
```

```
standard-2,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
f,kubernetes.io/hostname=kubernetes-minion-olsh
```

Load-balancers span all zones in a cluster; the guestbook-go example includes an example load-balanced service:

```
kubectl describe service guestbook | grep LoadBalancer.Ingress
```

The output is similar to this:

```
LoadBalancer Ingress:   130.211.126.21
```

Set the above IP:

```
export IP=130.211.126.21
```

Explore with curl via IP:

```
curl -s http://${IP}:3000/env | grep HOSTNAME
```

The output is similar to this:

```
  "HOSTNAME": "guestbook-44sep",
```

Again, explore multiple times:

```
(for i in `seq 20`; do curl -s http://${IP}:3000/env | grep
HOSTNAME; done)  | sort | uniq
```

The output is similar to this:

```
  "HOSTNAME": "guestbook-44sep",
  "HOSTNAME": "guestbook-hum5n",
  "HOSTNAME": "guestbook-ppm40",
```

The load balancer correctly targets all the pods, even though they are in multiple zones.

Shutting down the cluster

When you're done, clean up:

GCE:

```
KUBERNETES_PROVIDER=gce KUBE_USE_EXISTING_MASTER=true KUBE_GCE_ZO
NE=us-central1-f kubernetes/cluster/kube-down.sh
KUBERNETES_PROVIDER=gce KUBE_USE_EXISTING_MASTER=true KUBE_GCE_ZO
NE=us-central1-b kubernetes/cluster/kube-down.sh
KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-central1-a kubernetes/
cluster/kube-down.sh
```

AWS:

```
KUBERNETES_PROVIDER=aws KUBE_USE_EXISTING_MASTER=true KUBE_AWS_ZO
NE=us-west-2c kubernetes/cluster/kube-down.sh
KUBERNETES_PROVIDER=aws KUBE_USE_EXISTING_MASTER=true KUBE_AWS_ZO
NE=us-west-2b kubernetes/cluster/kube-down.sh
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2a kubernetes/
cluster/kube-down.sh
```

Feedback

Was this page helpful?

Yes No

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

Building large clusters

Support

At v1.17, Kubernetes supports clusters with up to 5000 nodes. More specifically, we support configurations that meet *all* of the following criteria:

- No more than 5000 nodes
- No more than 150000 total pods
- No more than 300000 total containers
- No more than 100 pods per node

Setup

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

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

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

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

Quota Issues

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

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

Etcd storage

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

When creating a cluster, existing salt scripts:

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

Size of master and master components

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

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

And the sizes we use on AWS are

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

Note:

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

On AWS, master node sizes are currently set at cluster startup time and do not change, even if you later scale your cluster up or down by manually removing or adding nodes or using a cluster autoscaler.

Addon Resources

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

For example:

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

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

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

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

node but CPU/memory usage increases slightly along with cluster load/ size as well):
  - FluentD with ElasticSearch Plugin
  - FluentD with GCP Plugin

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

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

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

Allowing minor node failure at startup

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

Feedback

Was this page helpful?

Yes No

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

Validate node setup

Node Conformance Test

Node conformance test is a containerized test framework that provides a system verification and functionality test for a node. The test validates whether the node meets the minimum requirements for Kubernetes; a node that passes the test is qualified to join a Kubernetes cluster.

Limitations

In Kubernetes version 1.5, node conformance test has the following limitations:

- Node conformance test only supports Docker as the container runtime.

Node Prerequisite

To run node conformance test, a node must satisfy the same prerequisites as a standard Kubernetes node. At a minimum, the node should have the following daemons installed:

- Container Runtime (Docker)
- Kubelet

Running Node Conformance Test

To run the node conformance test, perform the following steps:

1. Point your Kubelet to localhost `--api-servers="http://localhost:8080"`, because the test framework starts a local master to test Kubelet. There are some other Kubelet flags you may care:

    - `--pod-cidr`: If you are using `kubenet`, you should specify an arbitrary CIDR to Kubelet, for example `--pod-cidr=10.180.0.0/24`.
    - `--cloud-provider`: If you are using `--cloud-provider=gce`, you should remove the flag to run the test.

2. Run the node conformance test with command:

    ```
    # $CONFIG_DIR is the pod manifest path of your Kubelet.
    # $LOG_DIR is the test output path.
    sudo docker run -it --rm --privileged --net=host \
    -v /:/rootfs -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/
    result \
    k8s.gcr.io/node-test:0.2
    ```

Running Node Conformance Test for Other Architectures

Kubernetes also provides node conformance test docker images for other architectures:

Arch	Image
amd64	node-test-amd64
arm	node-test-arm
arm64	node-test-arm64

Running Selected Test

To run specific tests, overwrite the environment variable `FOCUS` with the regular expression of tests you want to run.

```
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs:ro -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/
result \
  -e FOCUS=MirrorPod \ # Only run MirrorPod test
  k8s.gcr.io/node-test:0.2
```

To skip specific tests, overwrite the environment variable `SKIP` with the regular expression of tests you want to skip.

```
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs:ro -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/
result \
  -e SKIP=MirrorPod \ # Run all conformance tests but skip
MirrorPod test
  k8s.gcr.io/node-test:0.2
```

Node conformance test is a containerized version of node e2e test. By default, it runs all conformance tests.

Theoretically, you can run any node e2e test if you configure the container and mount required volumes properly. But **it is strongly recommended to only run conformance test**, because it requires much more complex configuration to run non-conformance test.

Caveats

- The test leaves some docker images on the node, including the node conformance test image and images of containers used in the functionality test.
- The test leaves dead containers on the node. These containers are created during the functionality test.

Feedback

Was this page helpful?

Yes No

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

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]()

On top of the above CAs, it is also necessary to get a public/private key pair for service account management, `sa.key` and `sa.pub`.

All certificates

If you don't wish to copy the CA private keys to your cluster, 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`, `12` `.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_I` `]`
kube-apiserver-kubelet-client	kubernetes-ca	system:masters	client	
front-proxy-client	kubernetes-front-proxy-ca		client	

[1]: any other IP or DNS name you contact your cluster on (as used by [kubeadm]() the load balancer stable IP and/or DNS name, `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

Note: Hosts/SAN listed above are the recommended ones for getting a working cluster; if required by a specific setup, it is possible to add additional SANs on all the server certificates.

Note: For kubeadm users only: