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.

For a list of Certified Kubernetes providers, see "Partners".

Feedback

Was this page helpful?

Yes No

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

Analytics

Edit This Page
Page last modified on March 17, 2020 at 3:45 PM PST by KEP 1327: Apply feedback for third party and dual sourced content (#19486) (Page History)

Edit This Page

v1.18 Release Notes

v1.18.0

Documentation

Downloads for v1.18.0

filename	sha512 hash
kubernetes.tar.gz	cd5b86a3947a4f2cea6d857743ab2009be127d782b6f2eb4d37d88918a5
kubernetes-src.tar.gz	fb42cf133355ef18f67c8c4bb555aa1f284906c06e21fa41646e086d34e

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	26df342ef65745df12fa52931358e7f744111b6fe1e0bddb8c3c6598f
kubernetes-client-darwin-amd64.tar.gz	803a0fed122ef6b85f7a120b5485723eaade765b7bc8306d0c0da03bd
kubernetes-client-linux-386.tar.gz	110844511b70f9f3ebb92c15105e6680a05a562cd83f79ce2d2e25c2d
kubernetes-client-linux-amd64.tar.gz	594ca3eadc7974ec4d9e4168453e36ca434812167ef8359086cd64d04
kubernetes-client-linux-arm.tar.gz	d3627b763606557a6c9a5766c34198ec00b3a3cd72a55bc2cb4773106
kubernetes-client-linux-arm64.tar.gz	ba9056eff1452cbdaef699efbf88f74f5309b3f7808d372ebf6918442
kubernetes-client-linux-ppc64le.tar.gz	f80fb3769358cb20820ff1a1ce9994de5ed194aabe6c73fb8b8048bff
kubernetes-client-linux-s390x.tar.gz	a9b658108b6803d60fa3cd4e76d9e58bf75201017164fe54054b7ccad
kubernetes-client-windows-386.tar.gz	18adffab5d1be146906fd8531f4eae7153576aac235150ce2da05aee5
kubernetes-client-windows-amd64.tar.gz	162396256429cef07154f817de2a6b67635c770311f414e38b1e2db25

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	a92f8d201973d5dfa44a398e95fcf6a7b4feeb1ef879ab3fee1c54370e21f5
kubernetes-server-linux-arm.tar.gz	62fbff3256bc0a83f70244b09149a8d7870d19c2c4b6dee8ca2714fc7388da
kubernetes-server-linux-arm64.tar.gz	842910a7013f61a60d670079716b207705750d55a9e4f1f93696d19d39e191
kubernetes-server-linux-ppc64le.tar.gz	95c5b952ac1c4127a5c3b519b664972ee1fb5e8e902551ce71c04e26ad44b1
kubernetes-server-linux-s390x.tar.gz	a46522d2119a0fd58074564c1fa95dd8a929a79006b82ba3c4245611da8d2c

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	f714f80feecb0756410f27efb4cf4a1b5232be0444fbecec9f25cb85a7ccc
kubernetes-node-linux-arm.tar.gz	806000b5f6d723e24e2f12d19d1b9b3d16c74b855f51c7063284adf1fcc57
kubernetes-node-linux-arm64.tar.gz	c207e9ab60587d135897b5366af79efe9d2833f33401e469b2a4e0d74ecd2
kubernetes-node-linux-ppc64le.tar.gz	a542ed5ed02722af44ef12d1602f363fcd4e93cf704da2ea5d994638248
kubernetes-node-linux-s390x.tar.gz	651e0db73ee67869b2ae93cb0574168e4bd7918290fc5662a6b12b708fa62
kubernetes-node-windows-amd64.tar.gz	d726ed904f9f7fe7e8831df621dc9094b87e767410a129aa675ee08417b66

Changelog since v1.17.0

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

What's New (Major Themes)

Kubernetes Topology Manager Moves to Beta - Align Up!

A beta feature of Kubernetes in release 1.18, the Topology Manager feature enables NUMA alignment of CPU and devices (such as SR-IOV VFs) that will allow your workload to run in an environment optimized for low-latency. Prior to the introduction of the Topology Manager, the CPU and Device Manager would make resource allocation decisions independent of each other. This could result in undesirable allocations on multi-socket systems, causing degraded performance on latency critical applications.

Serverside Apply - Beta 2

Server-side Apply was promoted to Beta in 1.16, but is now introducing a second Beta in 1.18. This new version will track and manage changes to fields of all new Kubernetes objects, allowing you to know what changed your resources and when.

Extending Ingress with and replacing a deprecated annotation with IngressClass

In Kubernetes 1.18, there are two significant additions to Ingress: A new `pathType` field and a new `IngressClass` resource. The `pathType` field allows specifying how paths should be matched. In addition to the default `ImplementationSpecific` type, there are new `Exact` and `Prefix` path types.

The `IngressClass` resource is used to describe a type of Ingress within a Kubernetes cluster. Ingresses can specify the class they are associated with by using a new `ingressClassName` field on Ingresses. This new resource and field replace the deprecated `kubernetes.io/ingress.class` annotation.

SIG CLI introduces kubectl debug

SIG CLI was debating the need for a debug utility for quite some time already. With the development of ephemeral containers, it became more obvious how we can support developers with tooling built on top of `kubectl exec`. The addition of the `kubectl debug` command (it is alpha but your feedback is more than welcome), allows developers to easily debug their Pods inside the cluster. We think this addition is invaluable. This command allows one to create a temporary container which runs next to the Pod one is trying to examine, but also attaches to the console for interactive troubleshooting.

Introducing Windows CSI support alpha for Kubernetes

With the release of Kubernetes 1.18, an alpha version of CSI Proxy for Windows is getting released. CSI proxy enables non-privileged (pre-approved) containers to perform privileged storage operations on Windows. CSI drivers can now be supported in Windows by leveraging CSI proxy. SIG Storage made a lot of progress in the 1.18 release. In particular, the following storage features are moving to GA in Kubernetes 1.18: - Raw Block Support: Allow volumes to be surfaced as block devices inside containers instead of just mounted filesystems. - Volume Cloning: Duplicate a PersistentVolumeClaim and underlying storage volume using the Kubernetes API via CSI. - CSIDriver Kubernetes API Object: Simplifies CSI driver discovery and allows CSI Drivers to customize Kubernetes behavior.

SIG Storage is also introducing the following new storage features as alpha in Kubernetes 1.18: - Windows CSI Support: Enabling containerized CSI node plugins in Windows via new CSIProxy - Recursive Volume Ownership OnRootMismatch Option: Add a new "OnRootMismatch" policy that can help shorten the mount time for volumes that require ownership change and have many directories and files.

Other notable announcements

SIG Network is moving IPv6 to Beta in Kubernetes 1.18, after incrementing significantly the test coverage with new CI jobs.

NodeLocal DNSCache is an add-on that runs a dnsCache pod as a daemonset to improve clusterDNS performance and reliability. The feature has been in Alpha since 1.13 release. The SIG Network is announcing the GA graduation of Node Local DNSCache #1351

Known Issues

No Known Issues Reported

Urgent Upgrade Notes

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

kube-apiserver:

- in an `--encryption-provider-config` config file, an explicit `cacheSize : 0` parameter previously silently defaulted to caching 1000 keys. In Kubernetes 1.18, this now returns a config validation error. To disable caching, you can specify a negative cacheSize value in Kubernetes 1.18+.
- consumers of the â€˜certificatesigningrequests/approval' API must now have permission to â€˜approve' CSRs for the specific signer requested by the CSR. More information on the new signerName field and the required authorization can be found at https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests#authorization (#88246, @munnerz) [SIG API Machinery, Apps, Auth, CLI, Node and Testing]
- The following features are unconditionally enabled and the corresponding `--feature-gates` flags have been removed: `PodPriority`, `TaintNodesByCondition`, `ResourceQuotaScopeSelectors` and `ScheduleDaemonSetPods` (#86210, @draveness) [SIG Apps and Scheduling]

kubelet:

- `--enable-cadvisor-endpoints` is now disabled by default. If you need access to the cAdvisor v1 Json API please enable it explicitly in the kubelet command line. Please note that this flag was deprecated in 1.15 and will be removed in 1.19. (#87440, @dims) [SIG Instrumentation, Node and Testing]
- Promote CSIMigrationOpenStack to Beta (off by default since it requires installation of the OpenStack Cinder CSI Driver. The in-tree AWS OpenStack Cinder driver "kubernetes.io/cinder" was deprecated in 1.16 and will be removed in 1.20. Users should enable CSIMigration + CSIMigrationOpenStack features and install the OpenStack Cinder CSI Driver (https://github.com/kubernetes-sigs/cloud-provider-openstack) to avoid disruption to existing Pod and PVC objects at that time. Users should start using the OpenStack Cinder CSI Driver directly for any new volumes. (#85637, @dims) [SIG Cloud Provider]

kubectl:

- `kubectl` and k8s.io/client-go no longer default to a server address of `http://localhost:8080`. If you own one of these legacy clusters, you are *strongly* encouraged to secure your server. If you cannot secure your server, you can set the `$KUBERNETES_MASTER` environment variable to `http://localhost:8080` to continue defaulting the server address. `kubectl` users can also set the server address using the `--server` flag, or in a kubeconfig file specified via `--kubeconfig` or `$KUBECONFIG`. (#86173, @soltysh) [SIG API Machinery, CLI and Testing]
- `kubectl run` has removed the previously deprecated generators, along with flags unrelated to creating pods. `kubectl run` now only creates pods. See specific `kubectl create` subcommands to create objects other than pods. (#87077, @soltysh) [SIG Architecture, CLI and Testing]
- The deprecated command `kubectl rolling-update` has been removed (#88057, @julianvmodesto) [SIG Architecture, CLI and Testing]

client-go:

- Signatures on methods in generated clientsets, dynamic, metadata, and scale clients have been modified to accept `context.Context` as a first argument. Signatures of Create, Update, and Patch methods have been updated to accept CreateOptions, UpdateOptions and PatchOptions respectively. Signatures of Delete and DeleteCollection methods now accept DeleteOptions by value instead of by reference. Generated clientsets with the previous interface have been added in new "deprecated" packages to allow incremental migration to the new APIs. The deprecated packages will be removed in the 1.21 release. A tool is available at http://sigs.k8s.io/clientgofix to rewrite method invocations to the new signatures.

- The following deprecated metrics are removed, please convert to the corresponding metrics:

    - The following replacement metrics are available from v1.14.0:
        - `rest_client_request_latency_seconds` -> `rest_client_request_duration_seconds`
        - `scheduler_scheduling_latency_seconds` -> `scheduler_scheduling_duration_seconds`
        - `docker_operations` -> `docker_operations_total`
        - `docker_operations_latency_microseconds` -> `docker_operations_duration_seconds`
        - `docker_operations_errors` -> `docker_operations_errors_total`
        - `docker_operations_timeout` -> `docker_operations_timeout_total`
        - `network_plugin_operations_latency_microseconds` -> `network_plugin_operations_duration_seconds`
        - `kubelet_pod_worker_latency_microseconds` -> `kubelet_pod_worker_duration_seconds`

- kubelet_pod_start_latency_microseconds -> kubelet_pod_start_duration_seconds
- kubelet_cgroup_manager_latency_microseconds -> kubelet_cgroup_manager_duration_seconds
- kubelet_pod_worker_start_latency_microseconds -> kubelet_pod_worker_start_duration_seconds
- kubelet_pleg_relist_latency_microseconds -> kubelet_pleg_relist_duration_seconds
- kubelet_pleg_relist_interval_microseconds -> kubelet_pleg_relist_interval_seconds
- kubelet_eviction_stats_age_microseconds -> kubelet_eviction_stats_age_seconds
- kubelet_runtime_operations -> kubelet_runtime_operations_total
- kubelet_runtime_operations_latency_microseconds -> kubelet_runtime_operations_duration_seconds
- kubelet_runtime_operations_errors -> kubelet_runtime_operations_errors_total
- kubelet_device_plugin_registration_count -> kubelet_device_plugin_registration_total
- kubelet_device_plugin_alloc_latency_microseconds -> kubelet_device_plugin_alloc_duration_seconds
- scheduler_e2e_scheduling_latency_microseconds -> scheduler_e2e_scheduling_duration_seconds
- scheduler_scheduling_algorithm_latency_microseconds -> scheduler_scheduling_algorithm_duration_seconds
- scheduler_scheduling_algorithm_predicate_evaluation -> scheduler_scheduling_algorithm_predicate_evaluation_seconds
- scheduler_scheduling_algorithm_priority_evaluation -> scheduler_scheduling_algorithm_priority_evaluation_seconds
- scheduler_scheduling_algorithm_preemption_evaluation -> scheduler_scheduling_algorithm_preemption_evaluation_seconds
- scheduler_binding_latency_microseconds -> scheduler_binding_duration_seconds
- kubeproxy_sync_proxy_rules_latency_microseconds -> kubeproxy_sync_proxy_rules_duration_seconds
- apiserver_request_latencies -> apiserver_request_duration_seconds
- apiserver_dropped_requests -> apiserver_dropped_requests_total
- etcd_request_latencies_summary -> etcd_request_duration_seconds
- apiserver_storage_transformation_latencies_microseconds -> apiserver_storage_transformation_duration_seconds
- apiserver_storage_data_key_generation_latencies_microseconds -> apiserver_storage_data_key_generation_duration_seconds
- apiserver_request_count -> apiserver_request_total

        - `apiserver_request_latencies_summary`
    - The following replacement metrics are available from v1.15.0:
        - `apiserver_storage_transformation_failures_total` -> `apiserver_storage_transformation_operations_total` (#76496, @danielqsj) [SIG API Machinery, Cluster Lifecycle, Instrumentation, Network, Node and Scheduling]

Changes by Kind

Deprecation

kube-apiserver:

- the following deprecated APIs can no longer be served:
    - All resources under `apps/v1beta1` and `apps/v1beta2` - use `apps/v1` instead
    - `daemonsets`, `deployments`, `replicasets` resources under `extensions/v1beta1` - use `apps/v1` instead
    - `networkpolicies` resources under `extensions/v1beta1` - use `networking.k8s.io/v1` instead
    - `podsecuritypolicies` resources under `extensions/v1beta1` - use `policy/v1beta1` instead (#85903, @liggitt) [SIG API Machinery, Apps, Cluster Lifecycle, Instrumentation and Testing]

kube-controller-manager:

- Azure service annotation service.beta.kubernetes.io/azure-load-balancer-disable-tcp-reset has been deprecated. Its support would be removed in a future release. (#88462, @feiskyer) [SIG Cloud Provider]

kubelet:

- The StreamingProxyRedirects feature and `--redirect-container-streaming` flag are deprecated, and will be removed in a future release. The default behavior (proxy streaming requests through the kubelet) will be the only supported option. If you are setting `--redirect-container-streaming=true`, then you must migrate off this configuration. The flag will no longer be able to be enabled starting in v1.20. If you are not setting the flag, no action is necessary. (#88290, @tallclair) [SIG API Machinery and Node]
- resource metrics endpoint `/metrics/resource/v1alpha1` as well as all metrics under this endpoint have been deprecated. Please convert to the following metrics emitted by endpoint `/metrics/resource`:
    - scrape_error -> scrape_error
    - node_cpu_usage_seconds_total -> node_cpu_usage_seconds
    - node_memory_working_set_bytes -> node_memory_working_set_bytes
    - container_cpu_usage_seconds_total -> container_cpu_usage_seconds

    - container_memory_working_set_bytes -> container_memory_working_set_bytes
    - scrape_error -> scrape_error (#86282, @RainbowMango) [SIG Node]
- In a future release, kubelet will no longer create the CSI NodePublishVolume target directory, in accordance with the CSI specification. CSI drivers may need to be updated accordingly to properly create and process the target path. (#75535) [SIG Storage]

kube-proxy:

- `--healthz-port` and `--metrics-port` flags are deprecated, please use `--healthz-bind-address` and `--metrics-bind-address` instead (#88512, @SataQiu) [SIG Network]
- a new `EndpointSliceProxying` feature gate has been added to control the use of EndpointSlices in kube-proxy. The EndpointSlice feature gate that used to control this behavior no longer affects kube-proxy. This feature has been disabled by default. (#86137, @robscott)

kubeadm:

- command line option "kubelet-version" for `kubeadm upgrade node` has been deprecated and will be removed in a future release. (#87942, @SataQiu) [SIG Cluster Lifecycle]
- deprecate the usage of the experimental flag â€˜-use-api' under the â€˜kubeadm alpha certs renew' command. (#88827, @neolit123) [SIG Cluster Lifecycle]
- kube-dns is deprecated and will not be supported in a future version (#86574, @SataQiu) [SIG Cluster Lifecycle]
- the `ClusterStatus` struct present in the kubeadm-config ConfigMap is deprecated and will be removed in a future version. It is going to be maintained by kubeadm until it gets removed. The same information can be found on `etcd` and `kube-apiserver` pod annotations, `kubeadm.kubernetes.io/etcd.advertise-client-urls` and `kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint` respectively. (#87656, @ereslibre) [SIG Cluster Lifecycle]

kubectl:

- the boolean and unset values for the -dry-run flag are deprecated and a value -dry-run=server|client|none will be required in a future version. (#87580, @julianvmodesto) [SIG CLI]
- `kubectl apply --server-dry-run` is deprecated and replaced with -dry-run=server (#87580, @julianvmodesto) [SIG CLI]

add-ons:

- Remove cluster-monitoring addon (#85512, @serathius) [SIG Cluster Lifecycle, Instrumentation, Scalability and Testing]

kube-scheduler:

- The `scheduling_duration_seconds` summary metric is deprecated (#86586, @xiaoanyunfei) [SIG Scheduling]
- The `scheduling_algorithm_predicate_evaluation_seconds` and `scheduling_algorithm_priority_evaluation_seconds` metrics are deprecated, replaced by `framework_extension_point_duration_seconds[extension_point="Filter"]` and `framework_extension_point_duration_seconds[extension_point="Score"]`. (#86584, @xiaoanyunfei) [SIG Scheduling]
- `AlwaysCheckAllPredicates` is deprecated in scheduler Policy API. (#86369, @Huang-Wei) [SIG Scheduling]

Other deprecations:

- The k8s.io/node-api component is no longer updated. Instead, use the RuntimeClass types located within k8s.io/api, and the generated clients located within k8s.io/client-go (#87503, @liggitt) [SIG Node and Release]
- Removed the â€˜client' label from apiserver_request_total. (#87669, @logicalhan) [SIG API Machinery and Instrumentation]

API Change

New API types/versions:

- A new IngressClass resource has been added to enable better Ingress configuration. (#88509, @robscott) [SIG API Machinery, Apps, CLI, Network, Node and Testing]
- The CSIDriver API has graduated to storage.k8s.io/v1, and is now available for use. (#84814, @huffmanca) [SIG Storage]

New API fields:

- autoscaling/v2beta2 HorizontalPodAutoscaler added a `spec.behavior` field that allows scale behavior to be configured. Behaviors are specified separately for scaling up and down. In each direction a stabilization window can be specified as well as a list of policies and how to select amongst them. Policies can limit the absolute number of pods added or removed, or the percentage of pods added or removed. (#74525, @gliush) [SIG API Machinery, Apps, Autoscaling and CLI]
- Ingress:
  - `spec.ingressClassName` replaces the deprecated `kubernetes.io/ingress.class` annotation, and allows associating an Ingress object with a particular controller.
  - path definitions added a `pathType` field to allow indicating how the specified path should be matched against incoming requests. Valid values are `Exact`, `Prefix`, and `ImplementationSpecific` (#88587, @cmluciano) [SIG Apps, Cluster Lifecycle and Network]

- The alpha feature `AnyVolumeDataSource` enables PersistentVolumeClaim objects to use the spec.dataSource field to reference a custom type as a data source (#88636, @bswartz) [SIG Apps and Storage]
- The alpha feature `ConfigurableFSGroupPolicy` enables v1 Pods to specify a spec.securityContext.fsGroupChangePolicy policy to control how file permissions are applied to volumes mounted into the pod. (#88488, @gnufied) [SIG Storage]
- The alpha feature `ServiceAppProtocol` enables setting an `appProtocol` field in ServicePort and EndpointPort definitions. (#88503, @robscott) [SIG Apps and Network]
- The alpha feature `ImmutableEphemeralVolumes` enables an `immutable` field in both Secret and ConfigMap objects to mark their contents as immutable. (#86377, @wojtek-t) [SIG Apps, CLI and Testing]

Other API changes:

- The beta feature `ServerSideApply` enables tracking and managing changed fields for all new objects, which means there will be `managedFields` in `metadata` with the list of managers and their owned fields.
- The alpha feature `ServiceAccountIssuerDiscovery` enables publishing OIDC discovery information and service account token verification keys at `/.well-known/openid-configuration` and `/openid/v1/jwks` endpoints by API servers configured to issue service account tokens. (#80724, @cceckman) [SIG API Machinery, Auth, Cluster Lifecycle and Testing]
- CustomResourceDefinition schemas that use `x-kubernetes-list-map-keys` to specify properties that uniquely identify list items must make those properties required or have a default value, to ensure those properties are present for all list items. See https://kubernetes.io/docs/reference/using-api/api-concepts/#merge-strategy for details. (#88076, @eloyekunle) [SIG API Machinery and Testing]
- CustomResourceDefinition schemas that use `x-kubernetes-list-type: map` or `x-kubernetes-list-type: set` now enable validation that the list items in the corresponding custom resources are unique. (#84920, @sttts) [SIG API Machinery]

Configuration file changes:

kube-apiserver:

- The `--egress-selector-config-file` configuration file now accepts an apiserver.k8s.io/v1beta1 EgressSelectorConfiguration configuration object, and has been updated to allow specifying HTTP or GRPC connections to the network proxy (#87179, @Jefftree) [SIG API Machinery, Cloud Provider and Cluster Lifecycle]

kube-scheduler:

- A kubescheduler.config.k8s.io/v1alpha2 configuration file version is now accepted, with support for multiple scheduling profiles (#87628, @alculquicondor) [SIG Scheduling]
  - HardPodAffinityWeight moved from a top level ComponentConfig parameter to a PluginConfig parameter of InterPodAffinity Plugin in `kubescheduler.config.k8s.io/v1alpha2` (#88002, @alculquicondor) [SIG Scheduling and Testing]
  - Kube-scheduler can run more than one scheduling profile. Given a pod, the profile is selected by using its `.spec.schedulerName`. (#88285, @alculquicondor) [SIG Apps, Scheduling and Testing]
  - Scheduler Extenders can now be configured in the v1alpha2 component config (#88768, @damemi) [SIG Release, Scheduling and Testing]
  - The PostFilter of scheduler framework is renamed to PreScore in kubescheduler.config.k8s.io/v1alpha2. (#87751, @skilxn-go) [SIG Scheduling and Testing]

kube-proxy:

- Added kube-proxy flags `--ipvs-tcp-timeout`, `--ipvs-tcpfin-timeout`, `--ipvs-udp-timeout` to configure IPVS connection timeouts. (#85517, @andrewsykim) [SIG Cluster Lifecycle and Network]
- Added optional `--detect-local-mode` flag to kube-proxy. Valid values are "ClusterCIDR" (default matching previous behavior) and "NodeCIDR" (#87748, @satyasm) [SIG Cluster Lifecycle, Network and Scheduling]
- Kube-controller-manager and kube-scheduler expose profiling by default to match the kube-apiserver. Use `--enable-profiling=false` to disable. (#88663, @deads2k) [SIG API Machinery, Cloud Provider and Scheduling]
- Kubelet pod resources API now provides the information about active pods only. (#79409, @takmatsu) [SIG Node]
- New flag `--endpointslice-updates-batch-period` in kube-controller-manager can be used to reduce the number of endpointslice updates generated by pod changes. (#88745, @mborsz) [SIG API Machinery, Apps and Network]
- New flag `--show-hidden-metrics-for-version` in kube-proxy, kubelet, kube-controller-manager, and kube-scheduler can be used to show all hidden metrics that are deprecated in the previous minor release. (#85279, @RainbowMango) [SIG Cluster Lifecycle and Network]

Features graduated to beta:

- StartupProbe (#83437, @matthyx) [SIG Node, Scalability and Testing]

Features graduated to GA:

- VolumePVCDataSource (#88686, @j-griffith) [SIG Storage]

- TaintBasedEvictions (#87487, @skilxn-go) [SIG API Machinery, Apps, Node, Scheduling and Testing]
- BlockVolume and CSIBlockVolume (#88673, @jsafrane) [SIG Storage]
- Windows RunAsUserName (#87790, @marosset) [SIG Apps and Windows]
- The following feature gates are removed, because the associated features were unconditionally enabled in previous releases: CustomResourceValidation, CustomResourceSubresources, CustomResourceWebhookConversion, CustomResourcePublishOpenAPI, CustomResourceDefaulting (#87475, @liggitt) [SIG API Machinery]

Feature

- API request throttling (due to a high rate of requests) is now reported in client-go logs at log level 2. The messages are of the form:`Throttling request took 1.50705208s, request: GET:<URL>` The presence of these messages may indicate to the administrator the need to tune the cluster accordingly. (#87740, @jennybuckley) [SIG API Machinery]
- Add support for mount options to the FC volume plugin (#87499, @ejweber) [SIG Storage]
- Added a config-mode flag in azure auth module to enable getting AAD token without spn: prefix in audience claim. When it's not specified, the default behavior doesn't change. (#87630, @weinong) [SIG API Machinery, Auth, CLI and Cloud Provider]
- Allow for configuration of CoreDNS replica count (#85837, @pickledrick) [SIG Cluster Lifecycle]
- Allow user to specify resource using -filename flag when invoking kubectl exec (#88460, @soltysh) [SIG CLI and Testing]
- Apiserver added a new flag -goaway-chance which is the fraction of requests that will be closed gracefully(GOAWAY) to prevent HTTP/2 clients from getting stuck on a single apiserver. (#88567, @answer1991) [SIG API Machinery]
- Azure Cloud Provider now supports using Azure network resources (Virtual Network, Load Balancer, Public IP, Route Table, Network Security Group, etc.) in different AAD Tenant and Subscription than those for the Kubernetes cluster. To use the feature, please reference https://github.com/kubernetes-sigs/cloud-provider-azure/blob/master/docs/cloud-provider-config.md#host-network-resources-in-different-aad-tenant-and-subscription. (#88384, @bowen5) [SIG Cloud Provider]
- Azure VMSS/VMSSVM clients now suppress requests on throttling (#86740, @feiskyer) [SIG Cloud Provider]
- Azure cloud provider cache TTL is configurable, list of the azure cloud provider is as following:
  - "availabilitySetNodesCacheTTLInSeconds"
  - "vmssCacheTTLInSeconds"
  - "vmssVirtualMachinesCacheTTLInSeconds"
  - "vmCacheTTLInSeconds"
  - "loadBalancerCacheTTLInSeconds"
  - "nsgCacheTTLInSeconds"

    - "routeTableCacheTTLInSeconds" (#86266, @zqingqing1) [SIG Cloud Provider]
- Azure global rate limit is switched to per-client. A set of new rate limit configure options are introduced, including routeRateLimit, SubnetsRateLimit, InterfaceRateLimit, RouteTableRateLimit, LoadBalancerRateLimit, PublicIPAddressRateLimit, SecurityGroupRateLimit, VirtualMachineRateLimit, StorageAccountRateLimit, DiskRateLimit, SnapshotRateLimit, VirtualMachineScaleSetRateLimit and VirtualMachineSizeRateLimit. The original rate limit options would be default values for those new client's rate limiter. (#86515, @feiskyer) [SIG Cloud Provider]
- Azure network and VM clients now suppress requests on throttling (#87122, @feiskyer) [SIG Cloud Provider]
- Azure storage clients now suppress requests on throttling (#87306, @feiskyer) [SIG Cloud Provider]
- Azure: add support for single stack IPv6 (#88448, @aramase) [SIG Cloud Provider]
- DefaultConstraints can be specified for PodTopologySpread Plugin in the scheduler's ComponentConfig (#88671, @alculquicondor) [SIG Scheduling]
- DisableAvailabilitySetNodes is added to avoid VM list for VMSS clusters. It should only be used when vmType is "vmss" and all the nodes (including control plane nodes) are VMSS virtual machines. (#87685, @feiskyer) [SIG Cloud Provider]
- Elasticsearch supports automatically setting the advertise address (#85944, @SataQiu) [SIG Cluster Lifecycle and Instrumentation]
- EndpointSlices will now be enabled by default. A new `EndpointSlicePr oxying` feature gate determines if kube-proxy will use EndpointSlices, this is disabled by default. (#86137, @robscott) [SIG Network]
- Kube-proxy: Added dual-stack IPv4/IPv6 support to the iptables proxier. (#82462, @vllry) [SIG Network]
- Kubeadm now supports automatic calculations of dual-stack node cidr masks to kube-controller-manager. (#85609, @Arvinderpal) [SIG Cluster Lifecycle]
- Kubeadm: add a upgrade health check that deploys a Job (#81319, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: add the experimental feature gate PublicKeysECDSA that can be used to create a cluster with ECDSA certificates from "kubeadm init". Renewal of existing ECDSA certificates is also supported using "kubeadm alpha certs renew", but not switching between the RSA and ECDSA algorithms on the fly or during upgrades. (#86953, @rojkov) [SIG API Machinery, Auth and Cluster Lifecycle]
- Kubeadm: implemented structured output of â€˜kubeadm config images list' command in JSON, YAML, Go template and JsonPath formats (#86810, @bart0sh) [SIG Cluster Lifecycle]
- Kubeadm: on kubeconfig certificate renewal, keep the embedded CA in sync with the one on disk (#88052, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: reject a node joining the cluster if a node with the same name already exists (#81056, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: support Windows specific kubelet flags in kubeadm-flags.env (#88287, @gab-satchi) [SIG Cluster Lifecycle and Windows]

- Kubeadm: support automatic retry after failing to pull image (#86899, @SataQiu) [SIG Cluster Lifecycle]
- Kubeadm: upgrade supports fallback to the nearest known etcd version if an unknown k8s version is passed (#88373, @SataQiu) [SIG Cluster Lifecycle]
- Kubectl/drain: add disable-eviction option.Force drain to use delete, even if eviction is supported. This will bypass checking PodDisruptionBudgets, and should be used with caution. (#85571, @michaelgugino) [SIG CLI]
- Kubectl/drain: add skip-wait-for-delete-timeout option. If a pod's `DeletionTimestamp` is older than N seconds, skip waiting for the pod. Seconds must be greater than 0 to skip. (#85577, @michaelgugino) [SIG CLI]
- Option `preConfiguredBackendPoolLoadBalancerTypes` is added to azure cloud provider for the pre-configured load balancers, possible values: `""`, `"internal"`, `"external"`,`"all"` (#86338, @gossion) [SIG Cloud Provider]
- PodTopologySpread plugin now excludes terminatingPods when making scheduling decisions. (#87845, @Huang-Wei) [SIG Scheduling]
- Provider/azure: Network security groups can now be in a separate resource group. (#87035, @CecileRobertMichon) [SIG Cloud Provider]
- SafeSysctlWhitelist: add net.ipv4.ping_group_range (#85463, @AkihiroSuda) [SIG Auth]
- Scheduler framework permit plugins now run at the end of the scheduling cycle, after reserve plugins. Waiting on permit will remain in the beginning of the binding cycle. (#88199, @mateuszlitwin) [SIG Scheduling]
- Scheduler: Add DefaultBinder plugin (#87430, @alculquicondor) [SIG Scheduling and Testing]
- Skip default spreading scoring plugin for pods that define TopologySpreadConstraints (#87566, @skilxn-go) [SIG Scheduling]
- The kubectl -dry-run flag now accepts the values â€˜client', â€˜server', and â€˜none', to support client-side and server-side dry-run strategies. The boolean and unset values for the -dry-run flag are deprecated and a value will be required in a future version. (#87580, @julianvmodesto) [SIG CLI]
- Support server-side dry-run in kubectl with -dry-run=server for commands including apply, patch, create, run, annotate, label, set, autoscale, drain, rollout undo, and expose. (#87714, @julianvmodesto) [SIG API Machinery, CLI and Testing]
- Add -dry-run=server|client to kubectl delete, taint, replace (#88292, @julianvmodesto) [SIG CLI and Testing]
- The feature PodTopologySpread (feature gate `EvenPodsSpread`) has been enabled by default in 1.18. (#88105, @Huang-Wei) [SIG Scheduling and Testing]
- The kubelet and the default docker runtime now support running ephemeral containers in the Linux process namespace of a target container. Other container runtimes must implement support for this feature before it will be available for that runtime. (#84731, @verb) [SIG Node]
- The underlying format of the `CPUManager` state file has changed. Upgrades should be seamless, but any third-party tools that rely on

reading the previous format need to be updated. (#84462, @klueska) [SIG Node and Testing]
- Update CNI version to v0.8.5 (#78819, @justaugustus) [SIG API Machinery, Cluster Lifecycle, Network, Release and Testing]
- Webhooks have alpha support for network proxy (#85870, @Jefftree) [SIG API Machinery, Auth and Testing]
- When client certificate files are provided, reload files for new connections, and close connections when a certificate changes. (#79083, @jackkleeman) [SIG API Machinery, Auth, Node and Testing]
- When deleting objects using kubectl with the -force flag, you are no longer required to also specify -grace-period=0. (#87776, @brianpursley) [SIG CLI]
- Windows nodes on GCE can use virtual TPM-based authentication to the control plane. (#85466, @pjh) [SIG Cluster Lifecycle]
- You can now pass "-node-ip ::" to kubelet to indicate that it should autodetect an IPv6 address to use as the node's primary address. (#85850, @danwinship) [SIG Cloud Provider, Network and Node]
- `kubectl` now contains a `kubectl alpha debug` command. This command allows attaching an ephemeral container to a running pod for the purposes of debugging. (#88004, @verb) [SIG CLI]
- TLS Server Name overrides can now be specified in a kubeconfig file and via -tls-server-name in kubectl (#88769, @deads2k) [SIG API Machinery, Auth and CLI]

Metrics:

- Add `rest_client_rate_limiter_duration_seconds` metric to component-base to track client side rate limiter latency in seconds. Broken down by verb and URL. (#88134, @jennybuckley) [SIG API Machinery, Cluster Lifecycle and Instrumentation]
- Added two client certificate metrics for exec auth:
  - `rest_client_certificate_expiration_seconds` a gauge reporting the lifetime of the current client certificate. Reports the time of expiry in seconds since January 1, 1970 UTC.
  - `rest_client_certificate_rotation_age` a histogram reporting the age of a just rotated client certificate in seconds. (#84382, @sambdavidson) [SIG API Machinery, Auth, Cluster Lifecycle and Instrumentation]
- Controller manager serve workqueue metrics (#87967, @zhan849) [SIG API Machinery]
- Following metrics have been turned off:
  - kubelet_pod_worker_latency_microseconds
  - kubelet_pod_start_latency_microseconds
  - kubelet_cgroup_manager_latency_microseconds
  - kubelet_pod_worker_start_latency_microseconds
  - kubelet_pleg_relist_latency_microseconds
  - kubelet_pleg_relist_interval_microseconds
  - kubelet_eviction_stats_age_microseconds
  - kubelet_runtime_operations
  - kubelet_runtime_operations_latency_microseconds
  - kubelet_runtime_operations_errors

    - kubelet_device_plugin_registration_count
    - kubelet_device_plugin_alloc_latency_microseconds
    - kubelet_docker_operations
    - kubelet_docker_operations_latency_microseconds
    - kubelet_docker_operations_errors
    - kubelet_docker_operations_timeout
    - network_plugin_operations_latency_microseconds (#83841, @RainbowMango) [SIG Network and Node]
- Kube-apiserver metrics will now include request counts, latencies, and response sizes for /healthz, /livez, and /readyz requests. (#83598, @jktomer) [SIG API Machinery]
- Kubelet now exports a `server_expiration_renew_failure` and `client_expiration_renew_failure` metric counter if the certificate rotations cannot be performed. (#84614, @rphillips) [SIG API Machinery, Auth, CLI, Cloud Provider, Cluster Lifecycle, Instrumentation, Node and Release]
- Kubelet: the metric process_start_time_seconds be marked as with the ALPHA stability level. (#85446, @RainbowMango) [SIG API Machinery, Cluster Lifecycle, Instrumentation and Node]
- New metric `kubelet_pleg_last_seen_seconds` to aid diagnosis of PLEG not healthy issues. (#86251, @bboreham) [SIG Node]

Other (Bug, Cleanup or Flake)

- Fixed a regression with clients prior to 1.15 not being able to update podIP in pod status, or podCIDR in node spec, against >= 1.16 API servers (#88505, @liggitt) [SIG Apps and Network]
- Fixed "kubectl describe statefulsets.apps" printing garbage for rolling update partition (#85846, @phil9909) [SIG CLI]
- Add a event to PV when filesystem on PV does not match actual filesystem on disk (#86982, @gnufied) [SIG Storage]
- Add azure disk WriteAccelerator support (#87945, @andyzhangx) [SIG Cloud Provider and Storage]
- Add delays between goroutines for vm instance update (#88094, @aramase) [SIG Cloud Provider]
- Add init containers log to cluster dump info. (#88324, @zhouya0) [SIG CLI]
- Addons: elasticsearch discovery supports IPv6 (#85543, @SataQiu) [SIG Cluster Lifecycle and Instrumentation]
- Adds "volume.beta.kubernetes.io/migrated-to" annotation to PV's and PVC's when they are migrated to signal external provisioners to pick up those objects for Provisioning and Deleting. (#87098, @davidz627) [SIG Storage]
- All api-server log request lines in a more greppable format. (#87203, @lavalamp) [SIG API Machinery]
- Azure VMSS LoadBalancerBackendAddressPools updating has been improved with sequential-sync + concurrent-async requests. (#88699, @feiskyer) [SIG Cloud Provider]
- Azure cloud provider now obtains AAD token who audience claim will not have spn: prefix (#87590, @weinong) [SIG Cloud Provider]
- AzureFile and CephFS use the new Mount library that prevents logging of sensitive mount options. (#88684, @saad-ali) [SIG Storage]

- Bind dns-horizontal containers to linux nodes to avoid Windows scheduling on kubernetes cluster includes linux nodes and windows nodes (#83364, @wawa0210) [SIG Cluster Lifecycle and Windows]
- Bind kube-dns containers to linux nodes to avoid Windows scheduling (#83358, @wawa0210) [SIG Cluster Lifecycle and Windows]
- Bind metadata-agent containers to linux nodes to avoid Windows scheduling on kubernetes cluster includes linux nodes and windows nodes (#83363, @wawa0210) [SIG Cluster Lifecycle, Instrumentation and Windows]
- Bind metrics-server containers to linux nodes to avoid Windows scheduling on kubernetes cluster includes linux nodes and windows nodes (#83362, @wawa0210) [SIG Cluster Lifecycle, Instrumentation and Windows]
- Bug fixes: Make sure we include latest packages node #351 (@caseydavenport) (#84163, @david-tigera) [SIG Cluster Lifecycle]
- CPU limits are now respected for Windows containers. If a node is over-provisioned, no weighting is used, only limits are respected. (#86101, @PatrickLang) [SIG Node, Testing and Windows]
- Changed core_pattern on COS nodes to be an absolute path. (#86329, @mml) [SIG Cluster Lifecycle and Node]
- Client-go certificate manager rotation gained the ability to preserve optional intermediate chains accompanying issued certificates (#88744, @jackkleeman) [SIG API Machinery and Auth]
- Cloud provider config CloudProviderBackoffMode has been removed since it won't be used anymore. (#88463, @feiskyer) [SIG Cloud Provider]
- Conformance image now depends on stretch-slim instead of debian-hyperkube-base as that image is being deprecated and removed. (#88702, @dims) [SIG Cluster Lifecycle, Release and Testing]
- Deprecate -generator flag from kubectl create commands (#88655, @soltysh) [SIG CLI]
- During initialization phase (preflight), kubeadm now verifies the presence of the conntrack executable (#85857, @hnanni) [SIG Cluster Lifecycle]
- EndpointSlice should not contain endpoints for terminating pods (#89056, @andrewsykim) [SIG Apps and Network]
- Evictions due to pods breaching their ephemeral storage limits are now recorded by the `kubelet_evictions` metric and can be alerted on. (#87906, @smarterclayton) [SIG Node]
- Filter published OpenAPI schema by making nullable, required fields non-required in order to avoid kubectl to wrongly reject null values. (#85722, @sttts) [SIG API Machinery]
- Fix /readyz to return error immediately after a shutdown is initiated, before the -shutdown-delay-duration has elapsed. (#88911, @tkashem) [SIG API Machinery]
- Fix API Server potential memory leak issue in processing watch request. (#85410, @answer1991) [SIG API Machinery]
- Fix EndpointSlice controller race condition and ensure that it handles external changes to EndpointSlices. (#85703, @robscott) [SIG Apps and Network]
- Fix IPv6 addresses lost issue in pure ipv6 vsphere environment (#86001, @hubv) [SIG Cloud Provider]

- Fix LoadBalancer rule checking so that no unexpected LoadBalancer updates are made (#85990, @feiskyer) [SIG Cloud Provider]
- Fix a bug in kube-proxy that caused it to crash when using load balancers with a different IP family (#87117, @aojea) [SIG Network]
- Fix a bug in port-forward: named port not working with service (#85511, @oke-py) [SIG CLI]
- Fix a bug in the dual-stack IPVS proxier where stale IPv6 endpoints were not being cleaned up (#87695, @andrewsykim) [SIG Network]
- Fix a bug that orphan revision cannot be adopted and statefulset cannot be synced (#86801, @likakuli) [SIG Apps]
- Fix a bug where ExternalTrafficPolicy is not applied to service ExternalIPs. (#88786, @freehan) [SIG Network]
- Fix a bug where kubenet fails to parse the tc output. (#83572, @chendotjs) [SIG Network]
- Fix a regression in kubenet that prevent pods to obtain ip addresses (#85993, @chendotjs) [SIG Network and Node]
- Fix azure file AuthorizationFailure (#85475, @andyzhangx) [SIG Cloud Provider and Storage]
- Fix bug where EndpointSlice controller would attempt to modify shared objects. (#85368, @robscott) [SIG API Machinery, Apps and Network]
- Fix handling of aws-load-balancer-security-groups annotation. Security-Groups assigned with this annotation are no longer modified by kubernetes which is the expected behaviour of most users. Also no unnecessary Security-Groups are created anymore if this annotation is used. (#83446, @Elias481) [SIG Cloud Provider]
- Fix invalid VMSS updates due to incorrect cache (#89002, @ArchangelSDY) [SIG Cloud Provider]
- Fix isCurrentInstance for Windows by removing the dependency of hostname. (#89138, @feiskyer) [SIG Cloud Provider]
- Fix issue #85805 about a resource not found in azure cloud provider when LoadBalancer specified in another resource group. (#86502, @levimm) [SIG Cloud Provider]
- Fix kubectl annotate error when local=true is set (#86952, @zhouya0) [SIG CLI]
- Fix kubectl create deployment image name (#86636, @zhouya0) [SIG CLI]
- Fix `kubectl drain ignore` daemonsets and others. (#87361, @zhouya0) [SIG CLI]
- Fix missing "apiVersion" for "involvedObject" in Events for Nodes. (#87537, @uthark) [SIG Apps and Node]
- Fix nil pointer dereference in azure cloud provider (#85975, @ldx) [SIG Cloud Provider]
- Fix regression in statefulset conversion which prevents applying a statefulset multiple times. (#87706, @liggitt) [SIG Apps and Testing]
- Fix route conflicted operations when updating multiple routes together (#88209, @feiskyer) [SIG Cloud Provider]
- Fix that prevents repeated fetching of PVC/PV objects by kubelet when processing of pod volumes fails. While this prevents hammering API server in these error scenarios, it means that some errors in processing volume(s) for a pod could now take up to 2-3 minutes before retry. (#88141, @tedyu) [SIG Node and Storage]

- Fix the bug PIP's DNS is deleted if no DNS label service annotation isn't set. (#87246, @nilo19) [SIG Cloud Provider]
- Fix control plane hosts rolling upgrade causing thundering herd of LISTs on etcd leading to control plane unavailability. (#86430, @wojtek-t) [SIG API Machinery, Node and Testing]
- Fix: add azure disk migration support for CSINode (#88014, @andyzhangx) [SIG Cloud Provider and Storage]
- Fix: add non-retriable errors in azure clients (#87941, @andyzhangx) [SIG Cloud Provider]
- Fix: add remediation in azure disk attach/detach (#88444, @andyzhangx) [SIG Cloud Provider]
- Fix: azure data disk should use same key as os disk by default (#86351, @andyzhangx) [SIG Cloud Provider]
- Fix: azure disk could not mounted on Standard_DC4s/DC2s instances (#86612, @andyzhangx) [SIG Cloud Provider and Storage]
- Fix: azure file mount timeout issue (#88610, @andyzhangx) [SIG Cloud Provider and Storage]
- Fix: check disk status before disk azure disk (#88360, @andyzhangx) [SIG Cloud Provider]
- Fix: corrupted mount point in csi driver (#88569, @andyzhangx) [SIG Storage]
- Fix: get azure disk lun timeout issue (#88158, @andyzhangx) [SIG Cloud Provider and Storage]
- Fix: update azure disk max count (#88201, @andyzhangx) [SIG Cloud Provider and Storage]
- Fixed "requested device X but found Y" attach error on AWS. (#85675, @jsafrane) [SIG Cloud Provider and Storage]
- Fixed NetworkPolicy validation that `Except` values are accepted when they are outside the CIDR range. (#86578, @tnqn) [SIG Network]
- Fixed a bug in the TopologyManager. Previously, the TopologyManager would only guarantee alignment if container creation was serialized in some way. Alignment is now guaranteed under all scenarios of container creation. (#87759, @klueska) [SIG Node]
- Fixed a bug which could prevent a provider ID from ever being set for node if an error occurred determining the provider ID when the node was added. (#87043, @zjs) [SIG Apps and Cloud Provider]
- Fixed a data race in the kubelet image manager that can cause static pod workers to silently stop working. (#88915, @roycaihw) [SIG Node]
- Fixed a panic in the kubelet cleaning up pod volumes (#86277, @tedyu) [SIG Storage]
- Fixed a regression where the kubelet would fail to update the ready status of pods. (#84951, @tedyu) [SIG Node]
- Fixed an issue that could cause the kubelet to incorrectly run concurrent pod reconciliation loops and crash. (#89055, @tedyu) [SIG Node]
- Fixed block CSI volume cleanup after timeouts. (#88660, @jsafrane) [SIG Storage]
- Fixed cleaning of CSI raw block volumes. (#87978, @jsafrane) [SIG Storage]
- Fixed AWS Cloud Provider attempting to delete LoadBalancer security group it didn't provision, and fixed AWS Cloud Provider creating a default LoadBalancer security group even if annotation `service.beta.`

`kubernetes.io/aws-load-balancer-security-groups` is present because the intended behavior of aws-load-balancer-security-groups is to replace all security groups assigned to the load balancer. (#84265, @bhagwat070919) [SIG Cloud Provider]
- Fixed two scheduler metrics (pending_pods and schedule_attempts_total) not being recorded (#87692, @everpeace) [SIG Scheduling]
- Fixes an issue with kubelet-reported pod status on deleted/recreated pods. (#86320, @liggitt) [SIG Node]
- Fixes conversion error in multi-version custom resources that could cause metadata.generation to increment on no-op patches or updates of a custom resource. (#88995, @liggitt) [SIG API Machinery]
- Fixes issue where AAD token obtained by kubectl is incompatible with on-behalf-of flow and oidc. The audience claim before this fix has "spn:" prefix. After this fix, "spn:" prefix is omitted. (#86412, @weinong) [SIG API Machinery, Auth and Cloud Provider]
- Fixes an issue where you can't attach more than 15 GCE Persistent Disks to c2, n2, m1, m2 machine types. (#88602, @yuga711) [SIG Storage]
- Fixes kube-proxy when EndpointSlice feature gate is enabled on Windows. (#86016, @robscott) [SIG Auth and Network]
- Fixes kubelet crash in client certificate rotation cases (#88079, @liggitt) [SIG API Machinery, Auth and Node]
- Fixes service account token admission error in clusters that do not run the service account token controller (#87029, @liggitt) [SIG Auth]
- Fixes v1.17.0 regression in -service-cluster-ip-range handling with IPv4 ranges larger than 65536 IP addresses (#86534, @liggitt) [SIG Network]
- Fixes wrong validation result of NetworkPolicy PolicyTypes (#85747, @tnqn) [SIG Network]
- For subprotocol negotiation, both client and server protocol is required now. (#86646, @tedyu) [SIG API Machinery and Node]
- For volumes that allow attaches across multiple nodes, attach and detach operations across different nodes are now executed in parallel. (#88678, @verult) [SIG Storage]
- Garbage collector now can correctly orphan ControllerRevisions when StatefulSets are deleted with orphan propagation policy. (#84984, @cofyc) [SIG Apps]
- `Get-kube.sh` uses the gcloud's current local GCP service account for auth when the provider is GCE or GKE instead of the metadata server default (#88383, @BenTheElder) [SIG Cluster Lifecycle]
- Golang/x/net has been updated to bring in fixes for CVE-2020-9283 (#88381, @BenTheElder) [SIG API Machinery, CLI, Cloud Provider, Cluster Lifecycle and Instrumentation]
- If a serving certificate's param specifies a name that is an IP for an SNI certificate, it will have priority for replying to server connections. (#85308, @deads2k) [SIG API Machinery]
- Improved yaml parsing performance (#85458, @cjcullen) [SIG API Machinery, CLI, Cloud Provider, Cluster Lifecycle, Instrumentation and Node]
- Improves performance of the node authorizer (#87696, @liggitt) [SIG Auth]

- In GKE alpha clusters it will be possible to use the service annotation `cloud.google.com/network-tier: Standard` (#88487, @zioproto) [SIG Cloud Provider]
- Includes FSType when describing CSI persistent volumes. (#85293, @huffmanca) [SIG CLI and Storage]
- Iptables/userspace proxy: improve performance by getting local addresses only once per sync loop, instead of for every external IP (#85617, @andrewsykim) [SIG API Machinery, CLI, Cloud Provider, Cluster Lifecycle, Instrumentation and Network]
- Kube-aggregator: always sets unavailableGauge metric to reflect the current state of a service. (#87778, @p0lyn0mial) [SIG API Machinery]
- Kube-apiserver: fixed a conflict error encountered attempting to delete a pod with gracePeriodSeconds=0 and a resourceVersion precondition (#85516, @michaelgugino) [SIG API Machinery]
- Kube-proxy no longer modifies shared EndpointSlices. (#86092, @robscott) [SIG Network]
- Kube-proxy: on dual-stack mode, if it is not able to get the IP Family of an endpoint, logs it with level InfoV(4) instead of Warning, avoiding flooding the logs for endpoints without addresses (#88934, @aojea) [SIG Network]
- Kubeadm allows to configure single-stack clusters if dual-stack is enabled (#87453, @aojea) [SIG API Machinery, Cluster Lifecycle and Network]
- Kubeadm now includes CoreDNS version 1.6.7 (#86260, @rajansandeep) [SIG Cluster Lifecycle]
- Kubeadm upgrades always persist the etcd backup for stacked (#86861, @SataQiu) [SIG Cluster Lifecycle]
- Kubeadm: â€˜kubeadm alpha kubelet config download' has been removed, please use â€˜kubeadm upgrade node phase kubelet-config' instead (#87944, @SataQiu) [SIG Cluster Lifecycle]
- Kubeadm: Forward cluster name to the controller-manager arguments (#85817, @ereslibre) [SIG Cluster Lifecycle]
- Kubeadm: add support for the "ci/k8s-master" version label as a replacement for "ci-cross/*", which no longer exists. (#86609, @Pensu) [SIG Cluster Lifecycle]
- Kubeadm: apply further improvements to the tentative support for concurrent etcd member join. Fixes a bug where multiple members can receive the same hostname. Increase the etcd client dial timeout and retry timeout for add/remove/â€¦ operations. (#87505, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: don't write the kubelet environment file on "upgrade apply" (#85412, @boluisa) [SIG Cluster Lifecycle]
- Kubeadm: fix potential panic when executing "kubeadm reset" with a corrupted kubelet.conf file (#86216, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: fix the bug that â€˜kubeadm upgrade' hangs in single node cluster (#88434, @SataQiu) [SIG Cluster Lifecycle]
- Kubeadm: make sure images are pre-pulled even if a tag did not change but their contents changed (#85603, @bart0sh) [SIG Cluster Lifecycle]
- Kubeadm: remove â€˜kubeadm upgrade node config' command since it was deprecated in v1.15, please use â€˜kubeadm upgrade node phase kubelet-config' instead (#87975, @SataQiu) [SIG Cluster Lifecycle]

- Kubeadm: remove the deprecated CoreDNS feature-gate. It was set to "true" since v1.11 when the feature went GA. In v1.13 it was marked as deprecated and hidden from the CLI. (#87400, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: retry `kubeadm-config` ConfigMap creation or mutation if the apiserver is not responding. This will improve resiliency when joining new control plane nodes. (#85763, @ereslibre) [SIG Cluster Lifecycle]
- Kubeadm: tolerate whitespace when validating certificate authority PEM data in kubeconfig files (#86705, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: use bind-address option to configure the kube-controller-manager and kube-scheduler http probes (#86493, @aojea) [SIG Cluster Lifecycle]
- Kubeadm: uses the api-server AdvertiseAddress IP family to choose the etcd endpoint IP family for non external etcd clusters (#85745, @aojea) [SIG Cluster Lifecycle]
- Kubectl cluster-info dump -output-directory=xxx now generates files with an extension depending on the output format. (#82070, @olivierlemasle) [SIG CLI]
- `Kubectl describe <type>` and `kubectl top pod` will return a message saying `"No resources found"` or `"No resources found in <namespace> namespace"` if there are no results to display. (#87527, @brianpursley) [SIG CLI]
- `Kubectl drain node --dry-run` will list pods that would be evicted or deleted (#82660, @sallyom) [SIG CLI]
- `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) [SIG CLI]
- Kubelet metrics gathered through metrics-server or prometheus should no longer timeout for Windows nodes running more than 3 pods. (#87730, @marosset) [SIG Node, Testing and Windows]
- Kubelet metrics have been changed to buckets. For example the `exec/{podNamespace}/{podID}/{containerName}` is now just exec. (#87913, @cheftako) [SIG Node]
- Kubelets perform fewer unnecessary pod status update operations on the API server. (#88591, @smarterclayton) [SIG Node and Scalability]
- Kubernetes will try to acquire the iptables lock every 100 msec during 5 seconds instead of every second. This is especially useful for environments using kube-proxy in iptables mode with a high churn rate of services. (#85771, @aojea) [SIG Network]
- Limit number of instances in a single update to GCE target pool to 1000. (#87881, @wojtek-t) [SIG Cloud Provider, Network and Scalability]
- Make Azure clients only retry on specified HTTP status codes (#88017, @feiskyer) [SIG Cloud Provider]
- Make error message and service event message more clear (#86078, @feiskyer) [SIG Cloud Provider]
- Minimize AWS NLB health check timeout when externalTrafficPolicy set to Local (#73363, @kellycampbell) [SIG Cloud Provider]
- Pause image contains "Architecture" in non-amd64 images (#87954, @BenTheElder) [SIG Release]

- Pause image upgraded to 3.2 in kubelet and kubeadm. (#88173, @BenTheElder) [SIG CLI, Cluster Lifecycle, Node and Testing]
- Plugin/PluginConfig and Policy APIs are mutually exclusive when running the scheduler (#88864, @alculquicondor) [SIG Scheduling]
- Remove `FilteredNodesStatuses` argument from `PreScore`'s interface. (#88189, @skilxn-go) [SIG Scheduling and Testing]
- Resolved a performance issue in the node authorizer index maintenance. (#87693, @liggitt) [SIG Auth]
- Resolved regression in admission, authentication, and authorization webhook performance in v1.17.0-rc.1 (#85810, @liggitt) [SIG API Machinery and Testing]
- Resolves performance regression in `kubectl get all` and in client-go discovery clients constructed using `NewDiscoveryClientForConfig` or `NewDiscoveryClientForConfigOrDie`. (#86168, @liggitt) [SIG API Machinery]
- Reverted a kubectl azure auth module change where oidc claim spn: prefix was omitted resulting a breaking behavior with existing Azure AD OIDC enabled api-server (#87507, @weinong) [SIG API Machinery, Auth and Cloud Provider]
- Shared informers are now more reliable in the face of network disruption. (#86015, @squeed) [SIG API Machinery]
- Specifying PluginConfig for the same plugin more than once fails scheduler startup. Specifying extenders and configuring .ignoredResources for the NodeResourcesFit plugin fails (#88870, @alculquicondor) [SIG Scheduling]
- Terminating a restartPolicy=Never pod no longer has a chance to report the pod succeeded when it actually failed. (#88440, @smarterclayton) [SIG Node and Testing]
- The CSR signing cert/key pairs will be reloaded from disk like the kube-apiserver cert/key pairs (#86816, @deads2k) [SIG API Machinery, Apps and Auth]
- The EventRecorder from k8s.io/client-go/tools/events will now create events in the default namespace (instead of kube-system) when the related object does not have it set. (#88815, @enj) [SIG API Machinery]
- The audit event sourceIPs list will now always end with the IP that sent the request directly to the API server. (#87167, @tallclair) [SIG API Machinery and Auth]
- The sample-apiserver aggregated conformance test has updated to use the Kubernetes v1.17.0 sample apiserver (#84735, @liggitt) [SIG API Machinery, Architecture, CLI and Testing]
- To reduce chances of throttling, VM cache is set to nil when Azure node provisioning state is deleting (#87635, @feiskyer) [SIG Cloud Provider]
- VMSS cache is added so that less chances of VMSS GET throttling (#85885, @nilo19) [SIG Cloud Provider]
- Wait for kubelet & kube-proxy to be ready on Windows node within 10s (#85228, @YangLu1031) [SIG Cluster Lifecycle]
- `kubectl apply -f <file> --prune -n <namespace>` should prune all resources not defined in the file in the cli specified namespace. (#85613, @MartinKaburu) [SIG CLI]
- `kubectl create clusterrolebinding` creates rbac.authorization.k8s.io/v1 object (#85889, @oke-py) [SIG CLI]

- `kubectl diff` now returns 1 only on diff finding changes, and >1 on kubectl errors. The "exit status code 1" message has also been muted. (#87437, @apelisse) [SIG CLI and Testing]

Dependencies

- Update Calico to v3.8.4 (#84163, @david-tigera)[SIG Cluster Lifecycle]
- Update aws-sdk-go dependency to v1.28.2 (#87253, @SaranBalaji90) [SIG API Machinery and Cloud Provider]
- Update CNI version to v0.8.5 (#78819, @justaugustus)[SIG Release, Testing, Network, Cluster Lifecycle and API Machinery]
- Update cri-tools to v1.17.0 (#86305, @saschagrunert)[SIG Release and Cluster Lifecycle]
- Pause image upgraded to 3.2 in kubelet and kubeadm (#88173, @BenTheElder)[SIG CLI, Node, Testing and Cluster Lifecycle]
- Update CoreDNS version to 1.6.7 in kubeadm (#86260, @rajansandeep)[SIG Cluster Lifecycle]
- Update golang.org/x/crypto to fix CVE-2020-9283 (#8838, @BenTheElder)[SIG CLI, Instrumentation, API Machinery, CLuster Lifecycle and Cloud Provider]
- Update Go to 1.13.8 (#87648, @ialidzhikov)[SIG Release and Testing]
- Update Cluster-Autoscaler to 1.18.0 (#89095, @losipiuk)[SIG Autoscaling and Cluster Lifecycle]

v1.18.0-rc.1

Documentation

Downloads for v1.18.0-rc.1

filename	sha512 hash
kubernetes.tar.gz	c17231d5de2e0677e8af8259baa11a388625821c79b86362049f2edb366
kubernetes-src.tar.gz	e84ffad57c301f5d6e90f916b996d5abb0c987928c3ca6b1565f7b04258

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	1aea99923d492436b3eb91aaecffac94e5d0aa2b38a0930d266fda85c
kubernetes-client-darwin-amd64.tar.gz	07fa7340a959740bd52b83ff44438bbd988e235277dad1e43f125f08a
kubernetes-client-linux-386.tar.gz	48cebd26448fdd47aa36257baa4c716a98fda055bbf6a05230f2a3fe3
kubernetes-client-linux-amd64.tar.gz	c3a5fedf263f07a07f59c01fea6c63c1e0b76ee8dc67c45b6c134255c

filename	sha512 hash
kubernetes-client-linux-arm.tar.gz	a6b11a55bd38583bbaac14931a6862f8ce6493afe30947ba29e555665
kubernetes-client-linux-arm64.tar.gz	9e15331ac8010154a9b64f5488969fc8ee2f21059639896cb84c5cf4f
kubernetes-client-linux-ppc64le.tar.gz	f828fe6252678de9d4822e482f5873309ae9139b2db87298ab3273ce4
kubernetes-client-linux-s390x.tar.gz	19da4b45f0666c063934af616f3e7ed3caa99d4ee1e46d53efadc7a8a
kubernetes-client-windows-386.tar.gz	775c9afb6cb3e7c4ba53e9f48a5df2cf207234a33059bd74448bc9f17
kubernetes-client-windows-amd64.tar.gz	208d2595a5b57ac97aac75b4a2a6130f0c937f781a030bde1a432daf4

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	dcf832eae04f9f52ff473754ef5cfe697b35f4dc1a282622c94fa10943c8c
kubernetes-server-linux-arm.tar.gz	a04e34bea28eb1c8b492e8b1dd3c0dd87ebee71a7dbbef72be10a335e5533
kubernetes-server-linux-arm64.tar.gz	a6af086b07a8c2e498f32b43e6511bf6a5e6baf358c572c6910c8df17cd6ca
kubernetes-server-linux-ppc64le.tar.gz	5a960ef5ba0c255f587f2ac0b028cd03136dc91e4efc5d1becab46417852e5
kubernetes-server-linux-s390x.tar.gz	0f32c7d9b14bc238b9a5764d8f00edc4d3bf36bcf06b340b81061424e6070

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	27d8955d535d14f3f4dca501fd27e4f06fad84c6da878ea5332a5c83b6955
kubernetes-node-linux-arm.tar.gz	0d56eccad63ba608335988e90b377fe8ae978b177dc836cdb803a5c99d99e8
kubernetes-node-linux-arm64.tar.gz	79bb9be66f9e892d866b28e5cc838245818edb9706981fab6ccbff493181b

filename	sha512 hash
kubernetes-node-linux-ppc64le.tar.gz	3e9e2c6f9a2747d828069511dce8b4034c773c2d122f005f4508e22518055
kubernetes-node-linux-s390x.tar.gz	4f96e018c336fa13bb6df6f7217fe46a2b5c47f806f786499c429604ccba2
kubernetes-node-windows-amd64.tar.gz	ab110d76d506746af345e5897ef4f6993d5f53ac818ba69a334f364104735

Changelog since v1.18.0-beta.2

Changes by Kind

API Change

- Removes ConfigMap as suggestion for IngressClass parameters (#89093, @robscott) [SIG Network]

Other (Bug, Cleanup or Flake)

- EndpointSlice should not contain endpoints for terminating pods (#89056, @andrewsykim) [SIG Apps and Network]
- Fix a bug where ExternalTrafficPolicy is not applied to service ExternalIPs. (#88786, @freehan) [SIG Network]
- Fix invalid VMSS updates due to incorrect cache (#89002, @ArchangelSDY) [SIG Cloud Provider]
- Fix isCurrentInstance for Windows by removing the dependency of hostname. (#89138, @feiskyer) [SIG Cloud Provider]
- Fixed a data race in kubelet image manager that can cause static pod workers to silently stop working. (#88915, @roycaihw) [SIG Node]
- Fixed an issue that could cause the kubelet to incorrectly run concurrent pod reconciliation loops and crash. (#89055, @tedyu) [SIG Node]
- Kube-proxy: on dual-stack mode, if it is not able to get the IP Family of an endpoint, logs it with level InfoV(4) instead of Warning, avoiding flooding the logs for endpoints without addresses (#88934, @aojea) [SIG Network]
- Update Cluster Autoscaler to 1.18.0; changelog: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.18.0 (#89095, @losipiuk) [SIG Autoscaling and Cluster Lifecycle]

v1.18.0-beta.2

Documentation

Downloads for v1.18.0-beta.2

filename	sha512 hash
kubernetes.tar.gz	3017430ca17f8a3523669b4a02c39cedfc6c48b07281bc0a67a9fbe9d76
kubernetes-src.tar.gz	c5fd60601380a99efff4458b1c9cf4dc02195f6f756b36e590e54dff68f

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	7e49ede167b9271d4171e477fa21d267b2fb35f80869337d5b323198d
kubernetes-client-darwin-amd64.tar.gz	3f5cdf0e85eee7d0773e0ae2df1c61329dea90e0da92b02dae1ffd101
kubernetes-client-linux-386.tar.gz	b67b41c11bfecb88017c33feee21735c56f24cf6f7851b63c752495fc
kubernetes-client-linux-amd64.tar.gz	1fef2197cb80003e3a5c26f05e889af9d85fbbc23e27747944d2997ac
kubernetes-client-linux-arm.tar.gz	84e5f4d9776490219ee94a84adccd5dfc7c0362eb330709771afcde95
kubernetes-client-linux-arm64.tar.gz	ba613b114e0cca32fa21a3d10f845aa2f215d3af54e775f917ff93919
kubernetes-client-linux-ppc64le.tar.gz	502a6938d8c4bbe04abbd19b59919d86765058ff72334848be4012cec
kubernetes-client-linux-s390x.tar.gz	c24700e0ed2ef5c1d2dd282d638c88d90392ae90ea420837b39fd8e1c
kubernetes-client-windows-386.tar.gz	0d4c5a741b052f790c8b0923c9586ee9906225e51cf4dc8a56fc303d4
kubernetes-client-windows-amd64.tar.gz	841ef2e306c0c9593f04d9528ee019bf3b667761227d9afc1d6ca8bf1

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	b373df2e6ef55215e712315a5508e85a39126bd81b7b93c6b6305238919a8
kubernetes-server-linux-arm.tar.gz	b8103cb743c23076ce8dd7c2da01c8dd5a542fbac8480e82dc673139c8ee5
kubernetes-server-linux-arm64.tar.gz	8f8f05cf64fb9c8d80cdcb4935b2d3e3edc48bdd303231ae12f93e3f4d979

filename	sha512 hash
kubernetes-server-linux-ppc64le.tar.gz	b313b911c46f2ec129537407af3f165f238e48caeb4b9e530783ffa365930
kubernetes-server-linux-s390x.tar.gz	a1b6b06571141f507b12e5ef98efb88f4b6b9aba924722b2a74f11278d29a

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	20e02ca327543cddb2568ead3d5de164cbfb2914ab6416106d906bf12fcfb
kubernetes-node-linux-arm.tar.gz	ecd817ef05d6284f9c6592b84b0a48ea31cf4487030c9fb36518474b2a33d
kubernetes-node-linux-arm64.tar.gz	0020d32b7908ffd5055c8b26a8b3033e4702f89efcfffe3f6fcdb8a9921fa
kubernetes-node-linux-ppc64le.tar.gz	e065411d66d486e7793449c1b2f5a412510b913bf7f4e728c0a20e275642b
kubernetes-node-linux-s390x.tar.gz	082ee90413beaaea41d6cbe9a18f7d783a95852607f3b94190e0ca12aacdd
kubernetes-node-windows-amd64.tar.gz	fb5aca0cc36be703f9d4033eababd581bac5de8399c50594db087a99ed4cb

Changelog since v1.18.0-beta.1

Urgent Upgrade Notes

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

- `kubectl` no longer defaults to `http://localhost:8080`. If you own one of these legacy clusters, you are *strongly- encouraged to secure your server. If you cannot secure your server, you can set `KUBERNETES_MASTER` if you were relying on that behavior and you're a client-go user. Set `--server`, `--kubeconfig` or `KUBECONFIG` to make it work in `kubectl`. (#86173, @soltysh) [SIG API Machinery, CLI and Testing]

Changes by Kind

Deprecation

- AlgorithmSource is removed from v1alpha2 Scheduler ComponentConfig (#87999, @damemi) [SIG Scheduling]
- Kube-proxy: deprecate `--healthz-port` and `--metrics-port` flag, please use `--healthz-bind-address` and `--metrics-bind-address` instead (#88512, @SataQiu) [SIG Network]
- Kubeadm: deprecate the usage of the experimental flag â€˜-use-api' under the â€˜kubeadm alpha certs renew' command. (#88827, @neolit123) [SIG Cluster Lifecycle]

API Change

- A new IngressClass resource has been added to enable better Ingress configuration. (#88509, @robscott) [SIG API Machinery, Apps, CLI, Network, Node and Testing]
- Added GenericPVCDataSource feature gate to enable using arbitrary custom resources as the data source for a PVC. (#88636, @bswartz) [SIG Apps and Storage]
- Allow user to specify fsgroup permission change policy for pods (#88488, @gnufied) [SIG Apps and Storage]
- BlockVolume and CSIBlockVolume features are now GA. (#88673, @jsafrane) [SIG Apps, Node and Storage]
- CustomResourceDefinition schemas that use `x-kubernetes-list-map-keys` to specify properties that uniquely identify list items must make those properties required or have a default value, to ensure those properties are present for all list items. See https://kubernetes.io/docs/reference/using-api/api-concepts/#merge-strategy for details. (#88076, @eloyekunle) [SIG API Machinery and Testing]
- Fixes a regression with clients prior to 1.15 not being able to update podIP in pod status, or podCIDR in node spec, against >= 1.16 API servers (#88505, @liggitt) [SIG Apps and Network]
- Ingress: Add Exact and Prefix maching to Ingress PathTypes (#88587, @cmluciano) [SIG Apps, Cluster Lifecycle and Network]
- Ingress: Add alternate backends via TypedLocalObjectReference (#88775, @cmluciano) [SIG Apps and Network]
- Ingress: allow wildcard hosts in IngressRule (#88858, @cmluciano) [SIG Network]
- Kube-controller-manager and kube-scheduler expose profiling by default to match the kube-apiserver. Use `--enable-profiling=false` to disable. (#88663, @deads2k) [SIG API Machinery, Cloud Provider and Scheduling]
- Move TaintBasedEvictions feature gates to GA (#87487, @skilxn-go) [SIG API Machinery, Apps, Node, Scheduling and Testing]
- New flag -endpointslice-updates-batch-period in kube-controller-manager can be used to reduce number of endpointslice updates generated by pod changes. (#88745, @mborsz) [SIG API Machinery, Apps and Network]

- Scheduler Extenders can now be configured in the v1alpha2 component config (#88768, @damemi) [SIG Release, Scheduling and Testing]
- The apiserver/v1alph1#EgressSelectorConfiguration API is now beta. (#88502, @caesarxuchao) [SIG API Machinery]
- The storage.k8s.io/CSIDriver has moved to GA, and is now available for use. (#84814, @huffmanca) [SIG API Machinery, Apps, Auth, Node, Scheduling, Storage and Testing]
- VolumePVCDataSource moves to GA in 1.18 release (#88686, @j-griffith) [SIG Apps, CLI and Cluster Lifecycle]

Feature

- Add `rest_client_rate_limiter_duration_seconds` metric to component-base to track client side rate limiter latency in seconds. Broken down by verb and URL. (#88134, @jennybuckley) [SIG API Machinery, Cluster Lifecycle and Instrumentation]
- Allow user to specify resource using -filename flag when invoking kubectl exec (#88460, @soltysh) [SIG CLI and Testing]
- Apiserver add a new flag -goaway-chance which is the fraction of requests that will be closed gracefully(GOAWAY) to prevent HTTP/2 clients from getting stuck on a single apiserver. After the connection closed(received GOAWAY), the client's other in-flight requests won't be affected, and the client will reconnect. The flag min value is 0 (off), max is .02 (1/50 requests); .001 (1/1000) is a recommended starting point. Clusters with single apiservers, or which don't use a load balancer, should NOT enable this. (#88567, @answer1991) [SIG API Machinery]
- Azure: add support for single stack IPv6 (#88448, @aramase) [SIG Cloud Provider]
- DefaultConstraints can be specified for the PodTopologySpread plugin in the component config (#88671, @alculquicondor) [SIG Scheduling]
- Kubeadm: support Windows specific kubelet flags in kubeadm-flags.env (#88287, @gab-satchi) [SIG Cluster Lifecycle and Windows]
- Kubectl cluster-info dump changed to only display a message telling you the location where the output was written when the output is not standard output. (#88765, @brianpursley) [SIG CLI]
- Print NotReady when pod is not ready based on its conditions. (#88240, @soltysh) [SIG CLI]
- Scheduler Extender API is now located under k8s.io/kube-scheduler/extender (#88540, @damemi) [SIG Release, Scheduling and Testing]
- Signatures on scale client methods have been modified to accept `context.Context` as a first argument. Signatures of Get, Update, and Patch methods have been updated to accept GetOptions, UpdateOptions and PatchOptions respectively. (#88599, @julianvmodesto) [SIG API Machinery, Apps, Autoscaling and CLI]
- Signatures on the dynamic client methods have been modified to accept `context.Context` as a first argument. Signatures of Delete and DeleteCollection methods now accept DeleteOptions by value instead of by reference. (#88906, @liggitt) [SIG API Machinery, Apps, CLI, Cluster Lifecycle, Storage and Testing]
- Signatures on the metadata client methods have been modified to accept `context.Context` as a first argument. Signatures of Delete and

DeleteCollection methods now accept DeleteOptions by value instead of by reference. (#88910, @liggitt) [SIG API Machinery, Apps and Testing]
- Webhooks will have alpha support for network proxy (#85870, @Jefftree) [SIG API Machinery, Auth and Testing]
- When client certificate files are provided, reload files for new connections, and close connections when a certificate changes. (#79083, @jackkleeman) [SIG API Machinery, Auth, Node and Testing]
- When deleting objects using kubectl with the -force flag, you are no longer required to also specify -grace-period=0. (#87776, @brianpursley) [SIG CLI]
- `kubectl` now contains a `kubectl alpha debug` command. This command allows attaching an ephemeral container to a running pod for the purposes of debugging. (#88004, @verb) [SIG CLI]

Documentation

- Update Japanese translation for kubectl help (#86837, @inductor) [SIG CLI and Docs]
- `kubectl plugin` now prints a note how to install krew (#88577, @corneliusweig) [SIG CLI]

Other (Bug, Cleanup or Flake)

- Azure VMSS LoadBalancerBackendAddressPools updating has been improved with squential-sync + concurrent-async requests. (#88699, @feiskyer) [SIG Cloud Provider]
- AzureFile and CephFS use new Mount library that prevents logging of sensitive mount options. (#88684, @saad-ali) [SIG API Machinery, CLI, Cloud Provider, Cluster Lifecycle, Instrumentation and Storage]
- Build: Enable kube-cross image-building on K8s Infra (#88562, @justaugustus) [SIG Release and Testing]
- Client-go certificate manager rotation gained the ability to preserve optional intermediate chains accompanying issued certificates (#88744, @jackkleeman) [SIG API Machinery and Auth]
- Conformance image now depends on stretch-slim instead of debian-hyperkube-base as that image is being deprecated and removed. (#88702, @dims) [SIG Cluster Lifecycle, Release and Testing]
- Deprecate -generator flag from kubectl create commands (#88655, @soltysh) [SIG CLI]
- FIX: prevent apiserver from panicking when failing to load audit webhook config file (#88879, @JoshVanL) [SIG API Machinery and Auth]
- Fix /readyz to return error immediately after a shutdown is initiated, before the -shutdown-delay-duration has elapsed. (#88911, @tkashem) [SIG API Machinery]
- Fix a bug where kubenet fails to parse the tc output. (#83572, @chendotjs) [SIG Network]
- Fix describe ingress annotations not sorted. (#88394, @zhouya0) [SIG CLI]
- Fix handling of aws-load-balancer-security-groups annotation. Security-Groups assigned with this annotation are no longer modified by

kubernetes which is the expected behaviour of most users. Also no unnecessary Security-Groups are created anymore if this annotation is used. (#83446, @Elias481) [SIG Cloud Provider]
- Fix kubectl create deployment image name (#86636, @zhouya0) [SIG CLI]
- Fix missing "apiVersion" for "involvedObject" in Events for Nodes. (#87537, @uthark) [SIG Apps and Node]
- Fix that prevents repeated fetching of PVC/PV objects by kubelet when processing of pod volumes fails. While this prevents hammering API server in these error scenarios, it means that some errors in processing volume(s) for a pod could now take up to 2-3 minutes before retry. (#88141, @tedyu) [SIG Node and Storage]
- Fix: azure file mount timeout issue (#88610, @andyzhangx) [SIG Cloud Provider and Storage]
- Fix: corrupted mount point in csi driver (#88569, @andyzhangx) [SIG Storage]
- Fixed a bug in the TopologyManager. Previously, the TopologyManager would only guarantee alignment if container creation was serialized in some way. Alignment is now guaranteed under all scenarios of container creation. (#87759, @klueska) [SIG Node]
- Fixed block CSI volume cleanup after timeouts. (#88660, @jsafrane) [SIG Node and Storage]
- Fixes issue where you can't attach more than 15 GCE Persistent Disks to c2, n2, m1, m2 machine types. (#88602, @yuga711) [SIG Storage]
- For volumes that allow attaches across multiple nodes, attach and detach operations across different nodes are now executed in parallel. (#88678, @verult) [SIG Apps, Node and Storage]
- Hide kubectl.kubernetes.io/last-applied-configuration in describe command (#88758, @soltysh) [SIG Auth and CLI]
- In GKE alpha clusters it will be possible to use the service annotation `cloud.google.com/network-tier: Standard` (#88487, @zioproto) [SIG Cloud Provider]
- Kubelets perform fewer unnecessary pod status update operations on the API server. (#88591, @smarterclayton) [SIG Node and Scalability]
- Plugin/PluginConfig and Policy APIs are mutually exclusive when running the scheduler (#88864, @alculquicondor) [SIG Scheduling]
- Specifying PluginConfig for the same plugin more than once fails scheduler startup.

Specifying extenders and configuring .ignoredResources for the NodeResourcesFit plugin fails (#88870, @alculquicondor) [SIG Scheduling] - Support TLS Server Name overrides in kubeconfig file and via -tls-server-name in kubectl (#88769, @deads2k) [SIG API Machinery, Auth and CLI] - Terminating a restartPolicy=Never pod no longer has a chance to report the pod succeeded when it actually failed. (#88440, @smarterclayton) [SIG Node and Testing] - The EventRecorder from k8s.io/client-go/tools/events will now create events in the default namespace (instead of kube-system) when the related object does not have it set. (#88815, @enj) [SIG API Machinery] - The audit event sourceIPs list will now always end with the IP that sent the request directly to the API server. (#87167, @tallclair) [SIG API Machinery and Auth] - Update to use golang 1.13.8 (#87648, @ialidzhikov) [SIG Release and Testing] - Validate kube-proxy flags -ipvs-tcp-

timeout, -ipvs-tcpfin-timeout, -ipvs-udp-timeout (#88657, @chendotjs) [SIG Network]

v1.18.0-beta.1

Documentation

Downloads for v1.18.0-beta.1

filename	sha512 hash
kubernetes.tar.gz	7c182ca905b3a31871c01ab5fdaf46f074547536c7975e069ff230af0d4
kubernetes-src.tar.gz	d104b8c792b1517bd730787678c71c8ee3b259de81449192a49a1c6e37a

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	bc337bb8f200a789be4b97ce99b9d7be78d35ebd64746307c28339dc4
kubernetes-client-darwin-amd64.tar.gz	38dfa5e0b0cfff39942c913a6bcb2ad8868ec43457d35cffba08217bb
kubernetes-client-linux-386.tar.gz	8e63ec7ce29c69241120c037372c6c779e3f16253eabd612c7cbe6aa8
kubernetes-client-linux-amd64.tar.gz	c1be9f184a7c3f896a785c41cd6ece9d90d8cb9b1f6088bdfb5557d88
kubernetes-client-linux-arm.tar.gz	8eab02453cfd9e847632a774a0e0cf3a33c7619fb4ced7f1840e1f714
kubernetes-client-linux-arm64.tar.gz	f7df0ec02d2e7e63278d5386e8153cfe2b691b864f17b6452cc824a5f
kubernetes-client-linux-ppc64le.tar.gz	36dd5b10addca678a518e6d052c9d6edf473e3f87388a2f03f714c93c
kubernetes-client-linux-s390x.tar.gz	5bdbb44b996ab4ccf3a383780270f5cfdbf174982c300723c8bddf0a4
kubernetes-client-windows-386.tar.gz	5dea3d4c4e91ef889850143b361974250e99a3c526f5efee23ff9ccdc
kubernetes-client-windows-amd64.tar.gz	db298e698391368703e6aea7f4345aec5a4b8c69f9d8ff6c99fb5804a

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	c6284929dd5940e750b48db72ffbc09f73c5ec31ab3db283babb8e4e07cd8

filename	sha512 hash
kubernetes-server-linux-arm.tar.gz	6fc9552cf082c54cc0833b19876117c87ba7feb5a12c7e57f71b52208daf0
kubernetes-server-linux-arm64.tar.gz	b794b9c399e548949b5bfb2fe71123e86c2034847b2c99aca34b6de718a35
kubernetes-server-linux-ppc64le.tar.gz	fddaed7a54f97046a91c29534645811c6346e973e22950b2607b8c119c237
kubernetes-server-linux-s390x.tar.gz	65951a534bb55069c7419f41cbcdfe2fae31541d8a3f9eca11fc2489addf28

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	992059efb5cae7ed0ef55820368d854bad1c6d13a70366162cd3b5111ce24
kubernetes-node-linux-arm.tar.gz	c63ae0f8add5821ad267774314b8c8c1ffe3b785872bf278e721fd5dfdad1
kubernetes-node-linux-arm64.tar.gz	47adb9ddf6eaf8f475b89f59ee16fbd5df183149a11ad1574eaa645b47a6d5
kubernetes-node-linux-ppc64le.tar.gz	a3bc4a165567c7b76a3e45ab7b102d6eb3ecf373eb048173f921a4964cf9b
kubernetes-node-linux-s390x.tar.gz	109ddf37c748f69584c829db57107c3518defe005c11fcd2a1471845c15aa
kubernetes-node-windows-amd64.tar.gz	a3a75d2696ad3136476ad7d811e8eabaff5111b90e592695e651d6111f819

Changelog since v1.18.0-beta.0

Urgent Upgrade Notes

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

- The StreamingProxyRedirects feature and `--redirect-container-streaming` flag are deprecated, and will be removed in a future release. The default behavior (proxy streaming requests through the kubelet) will be the only supported option. If you are setting `--redirect-container-streaming=true`, then you must migrate off this

configuration. The flag will no longer be able to be enabled starting in v1.20. If you are not setting the flag, no action is necessary. (#88290, @tallclair) [SIG API Machinery and Node]

- Yes.

Feature Name: Support using network resources (VNet, LB, IP, etc.) in different AAD Tenant and Subscription than those for the cluster.

Changes in Pull Request:

```
1. Add properties `networkResourceTenantID` and
`networkResourceSubscriptionID` in cloud provider auth config
section, which indicates the location of network resources.
2. Add function `GetMultiTenantServicePrincipalToken` to fetch
multi-tenant service principal token, which will be used by
Azure VM/VMSS Clients in this feature.
3. Add function `GetNetworkResourceServicePrincipalToken` to
fetch network resource service principal token, which will be
used by Azure Network Resource (Load Balancer, Public IP, Route
Table, Network Security Group and their sub level resources)
Clients in this feature.
4. Related unit tests.
```

None.

User Documentation: In PR https://github.com/kubernetes-sigs/cloud-provider-azure/pull/301 (#88384, @bowen5) [SIG Cloud Provider]

Changes by Kind

Deprecation

- Azure service annotation service.beta.kubernetes.io/azure-load-balancer-disable-tcp-reset has been deprecated. Its support would be removed in a future release. (#88462, @feiskyer) [SIG Cloud Provider]

API Change

- API additions to apiserver types (#87179, @Jefftree) [SIG API Machinery, Cloud Provider and Cluster Lifecycle]
- Add Scheduling Profiles to kubescheduler.config.k8s.io/v1alpha2 (#88087, @alculquicondor) [SIG Scheduling and Testing]
- Added support for multiple sizes huge pages on a container level (#84051, @bart0sh) [SIG Apps, Node and Storage]
- AppProtocol is a new field on Service and Endpoints resources, enabled with the ServiceAppProtocol feature gate. (#88503, @robscott) [SIG Apps and Network]
- Fixed missing validation of uniqueness of list items in lists with `x-kubernetes-list-type: map` or x-kubernetes-list-type: set` in CustomResources. (#84920, @sttts) [SIG API Machinery]

- Introduces optional -detect-local flag to kube-proxy. Currently the only supported value is "cluster-cidr", which is the default if not specified. (#87748, @satyasm) [SIG Cluster Lifecycle, Network and Scheduling]
- Kube-scheduler can run more than one scheduling profile. Given a pod, the profile is selected by using its `.spec.SchedulerName`. (#88285, @alculquicondor) [SIG Apps, Scheduling and Testing]
- Moving Windows RunAsUserName feature to GA (#87790, @marosset) [SIG Apps and Windows]

Feature

- Add -dry-run to kubectl delete, taint, replace (#88292, @julianvmodesto) [SIG CLI and Testing]
- Add huge page stats to Allocated resources in "kubectl describe node" (#80605, @odinuge) [SIG CLI]
- Kubeadm: The ClusterStatus struct present in the kubeadm-config ConfigMap is deprecated and will be removed on a future version. It is going to be maintained by kubeadm until it gets removed. The same information can be found on `etcd` and `kube-apiserver` pod annotations, `kubeadm.kubernetes.io/etcd.advertise-client-urls` and `kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint` respectively. (#87656, @ereslibre) [SIG Cluster Lifecycle]
- Kubeadm: add the experimental feature gate PublicKeysECDSA that can be used to create a cluster with ECDSA certificates from "kubeadm init". Renewal of existing ECDSA certificates is also supported using "kubeadm alpha certs renew", but not switching between the RSA and ECDSA algorithms on the fly or during upgrades. (#86953, @rojkov) [SIG API Machinery, Auth and Cluster Lifecycle]
- Kubeadm: on kubeconfig certificate renewal, keep the embedded CA in sync with the one on disk (#88052, @neolit123) [SIG Cluster Lifecycle]
- Kubeadm: upgrade supports fallback to the nearest known etcd version if an unknown k8s version is passed (#88373, @SataQiu) [SIG Cluster Lifecycle]
- New flag `--show-hidden-metrics-for-version` in kube-scheduler can be used to show all hidden metrics that deprecated in the previous minor release. (#84913, @serathius) [SIG Instrumentation and Scheduling]
- Scheduler framework permit plugins now run at the end of the scheduling cycle, after reserve plugins. Waiting on permit will remain in the beginning of the binding cycle. (#88199, @mateuszlitwin) [SIG Scheduling]
- The kubelet and the default docker runtime now support running ephemeral containers in the Linux process namespace of a target container. Other container runtimes must implement this feature before it will be available in that runtime. (#84731, @verb) [SIG Node]

Other (Bug, Cleanup or Flake)

- Add delays between goroutines for vm instance update (#88094, @aramase) [SIG Cloud Provider]

- Add init containers log to cluster dump info. (#88324, @zhouya0) [SIG CLI]
- CPU limits are now respected for Windows containers. If a node is over-provisioned, no weighting is used - only limits are respected. (#86101, @PatrickLang) [SIG Node, Testing and Windows]
- Cloud provider config CloudProviderBackoffMode has been removed since it won't be used anymore. (#88463, @feiskyer) [SIG Cloud Provider]
- Evictions due to pods breaching their ephemeral storage limits are now recorded by the `kubelet_evictions` metric and can be alerted on. (#87906, @smarterclayton) [SIG Node]
- Fix: add remediation in azure disk attach/detach (#88444, @andyzhangx) [SIG Cloud Provider]
- Fix: check disk status before disk azure disk (#88360, @andyzhangx) [SIG Cloud Provider]
- Fixed cleaning of CSI raw block volumes. (#87978, @jsafrane) [SIG Storage]
- Get-kube.sh uses the gcloud's current local GCP service account for auth when the provider is GCE or GKE instead of the metadata server default (#88383, @BenTheElder) [SIG Cluster Lifecycle]
- Golang/x/net has been updated to bring in fixes for CVE-2020-9283 (#88381, @BenTheElder) [SIG API Machinery, CLI, Cloud Provider, Cluster Lifecycle and Instrumentation]
- Kubeadm now includes CoreDNS version 1.6.7 (#86260, @rajansandeep) [SIG Cluster Lifecycle]
- Kubeadm: fix the bug that â€˜kubeadm upgrade' hangs in single node cluster (#88434, @SataQiu) [SIG Cluster Lifecycle]
- Optimize kubectl version help info (#88313, @zhouya0) [SIG CLI]
- Removes the deprecated command `kubectl rolling-update` (#88057, @julianvmodesto) [SIG Architecture, CLI and Testing]

v1.18.0-alpha.5

Documentation

Downloads for v1.18.0-alpha.5

filename	sha512 hash
kubernetes.tar.gz	6452cac2b80721e9f577cb117c29b9ac6858812b4275c2becbf7431256
kubernetes-src.tar.gz	e41d9d4dd6910a42990051fcdca4bf5d3999df46375abd27ffc56aae9b4

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	5c95935863492b31d4aaa6be93260088dafea27663eb91edca980ca3a

filename	sha512 hash
kubernetes-client-darwin-amd64.tar.gz	868faa578b3738604d8be62fae599ccc556799f1ce54807f1fe72599f
kubernetes-client-linux-386.tar.gz	76a89d1d30b476b47f8fb808e342f89608e5c1c1787c4c06f2d7e763f
kubernetes-client-linux-amd64.tar.gz	07ad96a09b44d1c707d7c68312c5d69b101a3424bf1e6e9400b2e7a3f
kubernetes-client-linux-arm.tar.gz	c04fed9fa370a75c1b8e18b2be0821943bb9befcc784d14762ea3278e
kubernetes-client-linux-arm64.tar.gz	4199147dea9954333df26d34248a1cb7b02ebbd6380ffcd42d9f9ed5f
kubernetes-client-linux-ppc64le.tar.gz	4f6d4d61d1c52d3253ca19031ebcd4bad06d19b68bbaaab5c8e8c5907
kubernetes-client-linux-s390x.tar.gz	e2a454151ae5dd891230fb516a3f73f73ab97832db66fd3d12e7f1657
kubernetes-client-windows-386.tar.gz	14b262ba3b71c41f545db2a017cf1746075ada5745a858d2a62bc9df7
kubernetes-client-windows-amd64.tar.gz	26353c294755a917216664364b524982b7f5fc6aa832ce90134bb178d

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	ba77e0e7c610f59647c1b2601f82752964a0f54b7ad609a89b00fcfd553d0
kubernetes-server-linux-arm.tar.gz	45e87b3e844ea26958b0b489e8c9b90900a3253000850f5ff9e87ffdcafba
kubernetes-server-linux-arm64.tar.gz	155e136e3124ead69c594eead3398d6cfdbb8f823c324880e8a7bbd1b570b
kubernetes-server-linux-ppc64le.tar.gz	3fa0fb8221da19ad9d03278961172b7fa29a618b30abfa55e7243bb937ded
kubernetes-server-linux-s390x.tar.gz	db3199c3d7ba0b326d71dc8b80f50b195e79e662f71386a3b2976d47d13d7

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	addcdfbad7f12647e6babb8eadf853a374605c8f18bf63f416fa4d3bf1b90

filename	sha512 hash
kubernetes-node-linux-arm.tar.gz	b2ac54e0396e153523d116a2aaa32c919d6243931e0104cd47a23f546d710
kubernetes-node-linux-arm64.tar.gz	7aab36f2735cba805e4fd109831a1af0f586a88db3f07581b6dc2a2aab900
kubernetes-node-linux-ppc64le.tar.gz	a579936f07ebf86f69f297ac50ba4c34caf2c0b903f73190eb581c78382b0
kubernetes-node-linux-s390x.tar.gz	58fa0359ddd48835192fab1136a2b9b45d1927b04411502c269cda07cb8a8
kubernetes-node-windows-amd64.tar.gz	9086c03cd92b440686cea6d8c4e48045cc46a43ab92ae0e70350b3f51804b

Changelog since v1.18.0-alpha.3

Deprecation

- Kubeadm: command line option "kubelet-version" for `kubeadm upgrade node` has been deprecated and will be removed in a future release. (#87942, @SataQiu) [SIG Cluster Lifecycle]

API Change

- Kubelet podresources API now provides the information about active pods only. (#79409, @takmatsu) [SIG Node]
- Remove deprecated fields from .leaderElection in kubescheduler.config.k8s.io/v1alpha2 (#87904, @alculquicondor) [SIG Scheduling]
- Signatures on generated clientset methods have been modified to accept `context.Context` as a first argument. Signatures of generated Create, Update, and Patch methods have been updated to accept CreateOptions, UpdateOptions and PatchOptions respectively. Clientsets that with the previous interface have been added in new "deprecated" packages to allow incremental migration to the new APIs. The deprecated packages will be removed in the 1.21 release. (#87299, @mikedanese) [SIG API Machinery, Apps, Auth, Autoscaling, CLI, Cloud Provider, Cluster Lifecycle, Instrumentation, Network, Node, Scheduling, Storage, Testing and Windows]
- The k8s.io/node-api component is no longer updated. Instead, use the RuntimeClass types located within k8s.io/api, and the generated clients located within k8s.io/client-go (#87503, @liggitt) [SIG Node and Release]

Feature

- Add indexer for storage cacher (#85445, @shaloulcy) [SIG API Machinery]
- Add support for mount options to the FC volume plugin (#87499, @ejweber) [SIG Storage]
- Added a config-mode flag in azure auth module to enable getting AAD token without spn: prefix in audience claim. When it's not specified, the default behavior doesn't change. (#87630, @weinong) [SIG API Machinery, Auth, CLI and Cloud Provider]
- Introduced BackoffManager interface for backoff management (#87829, @zhan849) [SIG API Machinery]
- PodTopologySpread plugin now excludes terminatingPods when making scheduling decisions. (#87845, @Huang-Wei) [SIG Scheduling]
- Promote CSIMigrationOpenStack to Beta (off by default since it requires installation of the OpenStack Cinder CSI Driver) The in-tree AWS OpenStack Cinder "kubernetes.io/cinder" was already deprecated a while ago and will be removed in 1.20. Users should enable CSIMigration + CSIMigrationOpenStack features and install the OpenStack Cinder CSI Driver (https://github.com/kubernetes-sigs/cloud-provider-openstack) to avoid disruption to existing Pod and PVC objects at that time. Users should start using the OpenStack Cinder CSI Driver directly for any new volumes. (#85637, @dims) [SIG Cloud Provider]

Design

- The scheduler Permit extension point doesn't return a boolean value in its Allow() and Reject() functions. (#87936, @Huang-Wei) [SIG Scheduling]

Other (Bug, Cleanup or Flake)

- Adds "volume.beta.kubernetes.io/migrated-to" annotation to PV's and PVC's when they are migrated to signal external provisioners to pick up those objects for Provisioning and Deleting. (#87098, @davidz627) [SIG Apps and Storage]
- Fix a bug in the dual-stack IPVS proxier where stale IPv6 endpoints were not being cleaned up (#87695, @andrewsykim) [SIG Network]
- Fix kubectl drain ignore daemonsets and others. (#87361, @zhouya0) [SIG CLI]
- Fix: add azure disk migration support for CSINode (#88014, @andyzhangx) [SIG Cloud Provider and Storage]
- Fix: add non-retriable errors in azure clients (#87941, @andyzhangx) [SIG Cloud Provider]
- Fixed NetworkPolicy validation that Except values are accepted when they are outside the CIDR range. (#86578, @tnqn) [SIG Network]
- Improves performance of the node authorizer (#87696, @liggitt) [SIG Auth]
- Iptables/userspace proxy: improve performance by getting local addresses only once per sync loop, instead of for every external IP

(#85617, @andrewsykim) [SIG API Machinery, CLI, Cloud Provider, Cluster Lifecycle, Instrumentation and Network]
- Kube-aggregator: always sets unavailableGauge metric to reflect the current state of a service. (#87778, @p0lyn0mial) [SIG API Machinery]
- Kubeadm allows to configure single-stack clusters if dual-stack is enabled (#87453, @aojea) [SIG API Machinery, Cluster Lifecycle and Network]
- Kubeadm: â€˜kubeadm alpha kubelet config download' has been removed, please use â€˜kubeadm upgrade node phase kubelet-config' instead (#87944, @SataQiu) [SIG Cluster Lifecycle]
- Kubeadm: remove â€˜kubeadm upgrade node config' command since it was deprecated in v1.15, please use â€˜kubeadm upgrade node phase kubelet-config' instead (#87975, @SataQiu) [SIG Cluster Lifecycle]
- Kubectl describe and kubectl top pod will return a message saying "No resources found" or "No resources found in namespace" if there are no results to display. (#87527, @brianpursley) [SIG CLI]
- Kubelet metrics gathered through metrics-server or prometheus should no longer timeout for Windows nodes running more than 3 pods. (#87730, @marosset) [SIG Node, Testing and Windows]
- Kubelet metrics have been changed to buckets. For example the exec/{podNamespace}/{podID}/{containerName} is now just exec. (#87913, @cheftako) [SIG Node]
- Limit number of instances in a single update to GCE target pool to 1000. (#87881, @wojtek-t) [SIG Cloud Provider, Network and Scalability]
- Make Azure clients only retry on specified HTTP status codes (#88017, @feiskyer) [SIG Cloud Provider]
- Pause image contains "Architecture" in non-amd64 images (#87954, @BenTheElder) [SIG Release]
- Pods that are considered for preemption and haven't started don't produce an error log. (#87900, @alculquicondor) [SIG Scheduling]
- Prevent error message from being displayed when running kubectl plugin list and your path includes an empty string (#87633, @brianpursley) [SIG CLI]
- `kubectl create clusterrolebinding` creates rbac.authorization.k8s.io/v1 object (#85889, @oke-py) [SIG CLI]

v1.18.0-alpha.4

Documentation

Important note about manual tag

Due to a tagging bug in our Release Engineering tooling during `v1.18.0-alpha.3`, we needed to push a manual tag (`v1.18.0-alpha.4`).

No binaries have been produced or will be provided for `v1.18.0-alpha.4`.

The changelog for `v1.18.0-alpha.4` is included as part of the [changelog since v1.18.0-alpha.3][#changelog-since-v1180-alpha3] section.

v1.18.0-alpha.3

Documentation

Downloads for v1.18.0-alpha.3

filename	sha512 hash
kubernetes.tar.gz	60bf3bfc23b428f53fd853bac18a4a905b980fcc0bacd35ccd6357a89cf
kubernetes-src.tar.gz	8adf1016565a7c93713ab6fa4293c2d13b4f6e4e1ec4dcba60bd71e218b

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	abb32e894e8280c772e96227b574da81cd1eac374b8d29158b7f222ed
kubernetes-client-darwin-amd64.tar.gz	5e4b1a993264e256ec1656305de7c306094cae9781af8f1382df4ce4e
kubernetes-client-linux-386.tar.gz	68da39c2ae101d2b38f6137ceda07eb0c2124794982a62ef483245dbf
kubernetes-client-linux-amd64.tar.gz	dc236ffa8ad426620e50181419e9bebe3c161e953dbfb8a019f61b112
kubernetes-client-linux-arm.tar.gz	ab0a8bd6dc31ea160b731593cdc490b3cc03668b1141cf95310bd7060
kubernetes-client-linux-arm64.tar.gz	159ea083c601710d0d6aea423eeb346c99ffaf2abd137d35a53e87a07
kubernetes-client-linux-ppc64le.tar.gz	16b0459adfa26575d13be49ab53ac7f0ffd05e184e4e13d2dfbfe725d
kubernetes-client-linux-s390x.tar.gz	d5aa1f5d89168995d2797eb839a04ce32560f405b38c1c0baaa0e313e
kubernetes-client-windows-386.tar.gz	374e16a1e52009be88c94786f80174d82dff66399bf294c9bee18a215
kubernetes-client-windows-amd64.tar.gz	5a94c1068c19271f810b994adad8e62fae03b3d4473c7c9e6d056995f

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	a677bec81f0eba75114b92ff955bac74512b47e53959d56a685dae5edd527

filename	sha512 hash
kubernetes-server-linux-arm.tar.gz	2fb696f86ff13ebeb5f3cf2b254bf41303644c5ea84a292782eac61235507
kubernetes-server-linux-arm64.tar.gz	738e95da9cfb8f1309479078098de1c38cef5e1dd5ee1129b77651a936a41
kubernetes-server-linux-ppc64le.tar.gz	7a85bfcbb2aa636df60c41879e96e788742ecd72040cb0db2a93418439c125
kubernetes-server-linux-s390x.tar.gz	1f1cdb2efa3e7cac857203d8845df2fdaa5cf1f20df764effff29371945e

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	4ccfced3f5ba4adfa58f4a9d1b2c5bdb3e89f9203ab0e27d11eb1c325ac32
kubernetes-node-linux-arm.tar.gz	d695a69d18449062e4c129e54ec8384c573955f8108f4b78adc2ec929719f
kubernetes-node-linux-arm64.tar.gz	21df1da88c89000abc22f97e482c3aaa5ce53ec9628d83dda2e04a1d86c4d
kubernetes-node-linux-ppc64le.tar.gz	ff77e3aacb6ed9d89baed92ef542c8b5cec83151b6421948583cf608bca3b
kubernetes-node-linux-s390x.tar.gz	57d75b7977ec1a0f6e7ed96a304dbb3b8664910f42ca19aab319a9ec33535
kubernetes-node-windows-amd64.tar.gz	63fdbb71773cfd73a914c498e69bb9eea3fc314366c99ffb8bd42ec5b4dae

Changelog since v1.18.0-alpha.2

Deprecation

- Remove all the generators from kubectl run. It will now only create pods. Additionally, deprecates all the flags that are not relevant anymore. (#87077, @soltysh) [SIG Architecture, SIG CLI, and SIG Testing]
- kubeadm: kube-dns is deprecated and will not be supported in a future version (#86574, @SataQiu) [SIG Cluster Lifecycle]

API Change

- Add kubescheduler.config.k8s.io/v1alpha2 (#87628, @alculquicondor) [SIG Scheduling]
- -enable-cadvisor-endpoints is now disabled by default. If you need access to the cAdvisor v1 Json API please enable it explicitly in the kubelet command line. Please note that this flag was deprecated in 1.15 and will be removed in 1.19. (#87440, @dims) [SIG Instrumentation, SIG Node, and SIG Testing]
- The following feature gates are removed, because the associated features were unconditionally enabled in previous releases: CustomResourceValidation, CustomResourceSubresources, CustomResourceWebhookConversion, CustomResourcePublishOpenAPI, CustomResourceDefaulting (#87475, @liggitt) [SIG API Machinery]

Feature

- aggragation api will have alpha support for network proxy (#87515, @Sh4d1) [SIG API Machinery]
- API request throttling (due to a high rate of requests) is now reported in client-go logs at log level 2. The messages are of the form

Throttling request took 1.50705208s, request: GET:

The presence of these messages, may indicate to the administrator the need to tune the cluster accordingly. (#87740, @jennybuckley) [SIG API Machinery] - kubeadm: reject a node joining the cluster if a node with the same name already exists (#81056, @neolit123) [SIG Cluster Lifecycle] - disableAvailabilitySetNodes is added to avoid VM list for VMSS clusters. It should only be used when vmType is "vmss" and all the nodes (including masters) are VMSS virtual machines. (#87685, @feiskyer) [SIG Cloud Provider] - The kubectl -dry-run flag now accepts the values â€˜client', â€˜server', and â€˜none', to support client-side and server-side dry-run strategies. The boolean and unset values for the -dry-run flag are deprecated and a value will be required in a future version. (#87580, @julianvmodesto) [SIG CLI] - Add support for pre-allocated hugepages for more than one page size (#82820, @odinuge) [SIG Apps] - Update CNI version to v0.8.5 (#78819, @justaugustus) [SIG API Machinery, SIG Cluster Lifecycle, SIG Network, SIG Release, and SIG Testing] - Skip default spreading scoring plugin for pods that define TopologySpreadConstraints (#87566, @skilxn-go) [SIG Scheduling] - Added more details to taint toleration errors (#87250, @starizard) [SIG Apps, and SIG Scheduling] - Scheduler: Add DefaultBinder plugin (#87430, @alculquicondor) [SIG Scheduling, and SIG Testing] - Kube-apiserver metrics will now include request counts, latencies, and response sizes for /healthz, /livez, and /readyz requests. (#83598, @jktomer) [SIG API Machinery]

Other (Bug, Cleanup or Flake)

- Fix the masters rolling upgrade causing thundering herd of LISTs on etcd leading to control plane unavailability. (#86430, @wojtek-t) [SIG API Machinery, SIG Node, and SIG Testing]
- `kubectl diff` now returns 1 only on diff finding changes, and >1 on kubectl errors. The "exit status code 1" message as also been muted. (#87437, @apelisse) [SIG CLI, and SIG Testing]
- To reduce chances of throttling, VM cache is set to nil when Azure node provisioning state is deleting (#87635, @feiskyer) [SIG Cloud Provider]
- Fix regression in statefulset conversion which prevented applying a statefulset multiple times. (#87706, @liggitt) [SIG Apps, and SIG Testing]
- fixed two scheduler metrics (pending_pods and schedule_attempts_total) not being recorded (#87692, @everpeace) [SIG Scheduling]
- Resolved a performance issue in the node authorizer index maintenance. (#87693, @liggitt) [SIG Auth]
- Removed the â€˜client' label from apiserver_request_total. (#87669, @logicalhan) [SIG API Machinery, and SIG Instrumentation]
- `(*"k8s.io/client-go/rest".Request).{Do,DoRaw,Stream,Watch}` now require callers to pass a `context.Context` as an argument. The context is used for timeout and cancellation signaling and to pass supplementary information to round trippers in the wrapped transport chain. If you don't need any of this functionality, it is sufficient to pass a context created with `context.Background()` to these functions. The `(*"k8s.io/client-go/rest".Request).Context` method is removed now that all methods that execute a request accept a context directly. (#87597, @mikedanese) [SIG API Machinery, SIG Apps, SIG Auth, SIG Autoscaling, SIG CLI, SIG Cloud Provider, SIG Cluster Lifecycle, SIG Instrumentation, SIG Network, SIG Node, SIG Scheduling, SIG Storage, and SIG Testing]
- For volumes that allow attaches across multiple nodes, attach and detach operations across different nodes are now executed in parallel. (#87258, @verult) [SIG Apps, SIG Node, and SIG Storage]
- kubeadm: apply further improvements to the tentative support for concurrent etcd member join. Fixes a bug where multiple members can receive the same hostname. Increase the etcd client dial timeout and retry timeout for add/remove/â€¦ operations. (#87505, @neolit123) [SIG Cluster Lifecycle]
- Reverted a kubectl azure auth module change where oidc claim spn: prefix was omitted resulting a breaking behavior with existing Azure AD OIDC enabled api-server (#87507, @weinong) [SIG API Machinery, SIG Auth, and SIG Cloud Provider]
- Update cri-tools to v1.17.0 (#86305, @saschagrunert) [SIG Cluster Lifecycle, and SIG Release]
- kubeadm: remove the deprecated CoreDNS feature-gate. It was set to "true" since v1.11 when the feature went GA. In v1.13 it was marked as deprecated and hidden from the CLI. (#87400, @neolit123) [SIG Cluster Lifecycle]
- Shared informers are now more reliable in the face of network disruption. (#86015, @squeed) [SIG API Machinery]

- the CSR signing cert/key pairs will be reloaded from disk like the kube-apiserver cert/key pairs (#86816, @deads2k) [SIG API Machinery, SIG Apps, and SIG Auth]
- "kubectl describe statefulsets.apps" prints garbage for rolling update partition (#85846, @phil9909) [SIG CLI]

v1.18.0-alpha.2

Documentation

Downloads for v1.18.0-alpha.2

filename	sha512 hash
kubernetes.tar.gz	7af83386b4b35353f0aa1bdaf73599eb08b1d1ca11ecc2c606854aff754
kubernetes-src.tar.gz	a14b02a0a0bde97795a836a8f5897b0ee6b43e010e13e43dd4cca80a5b9

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	427f214d47ded44519007de2ae87160c56c2920358130e474b7682997
kubernetes-client-darwin-amd64.tar.gz	861fd81ac3bd45765575bedf5e002a2294aba48ef9e15980fc7d67839
kubernetes-client-linux-386.tar.gz	7d59b05d6247e2606a8321c72cd239713373d876dbb43b0fb7f1cb857
kubernetes-client-linux-amd64.tar.gz	7cdefb4e32bad9d2df5bb8e7e0a6f4dab2ae6b7afef5d801ac5c342d4
kubernetes-client-linux-arm.tar.gz	6212bbf0fa1d01ced77dcca2c4b76b73956cd3c6b70e0701c1fe0df5f
kubernetes-client-linux-arm64.tar.gz	1f0d9990700510165ee471acb2f88222f1b80e8f6deb351ce14cf50a7
kubernetes-client-linux-ppc64le.tar.gz	77e00ba12a32db81e96f8de84609de93f32c61bb3f53875a57496d213
kubernetes-client-linux-s390x.tar.gz	a39ec2044bed5a4570e9c83068e0fc0ce923ccffa44380f8bbc324742
kubernetes-client-windows-386.tar.gz	1a0ab88f9b7e34b60ab31d5538e97202a256ad8b7b7ed5070cae5f2f1
kubernetes-client-windows-amd64.tar.gz	1966eb5dfb78c1bc33aaa6389f32512e3aa92584250a0164182f3566c

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	f814d6a3872e4572aa4da297c29def4c1fad8eba0903946780b6bf9788c72
kubernetes-server-linux-arm.tar.gz	56aa08225e546c92c2ff88ac57d3db7dd5e63640772ea72a429f080f70698
kubernetes-server-linux-arm64.tar.gz	fb87128d905211ba097aa860244a376575ae2edbaca6e51402a24bc296485
kubernetes-server-linux-ppc64le.tar.gz	6d21fbf39b9d3a0df9642407d6f698fabdc809aca83af197bceb58a81b258
kubernetes-server-linux-s390x.tar.gz	ddcda4dc360ca97705f71bf2a18ddacd7b7ddf77535b62e699e97a1b2dd24

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	78915a9bde35c70c67014f0cea8754849db4f6a84491a3ad9678fd3bc0203
kubernetes-node-linux-arm.tar.gz	3218e811abcb0cb09d80742def339be3916db5e9bbc62c0dc8e6d87085f7e
kubernetes-node-linux-arm64.tar.gz	fa22de9c4440b8fb27f4e77a5a63c5e1c8aa8aa30bb79eda843b0f40498c2
kubernetes-node-linux-ppc64le.tar.gz	bbda9b5cc66e8f13d235703b2a85e2c4f02fa16af047be4d27a3e198e11eb
kubernetes-node-linux-s390x.tar.gz	b2ed1eda013069adce2aac00b86d75b84e006cfce9bafac0b5a2bafcb60f8
kubernetes-node-windows-amd64.tar.gz	bd8eb23dba711f31b5148257076b1bbe9629f2a75de213b2c779bd5b29279

Changelog since v1.18.0-alpha.1

Other notable changes

- Bump golang/mock version to v1.3.1 (#87326, @wawa0210)

- fix a bug that orphan revision cannot be adopted and statefulset cannot be synced (#86801, @likakuli)
- Azure storage clients now suppress requests on throttling (#87306, @feiskyer)
- Introduce Alpha field `Immutable` in both Secret and ConfigMap objects to mark their contents as immutable. The implementation is hidden behind feature gate `ImmutableEphemeralVolumes` (currently in Alpha stage). (#86377, @wojtek-t)
- EndpointSlices will now be enabled by default. A new `EndpointSlicePr oxying` feature gate determines if kube-proxy will use EndpointSlices, this is disabled by default. (#86137, @robscott)
- kubeadm upgrades always persist the etcd backup for stacked (#86861, @SataQiu)
- Fix the bug PIP's DNS is deleted if no DNS label service annotation isn't set. (#87246, @nilo19)
- New flag `--show-hidden-metrics-for-version` in kube-controller-manager can be used to show all hidden metrics that deprecated in the previous minor release. (#85281, @RainbowMango)
- Azure network and VM clients now suppress requests on throttling (#87122, @feiskyer)
- `kubectl apply -f <file> --prune -n <namespace>` should prune all resources not defined in the file in the cli specified namespace. (#85613, @MartinKaburu)
- Fixes service account token admission error in clusters that do not run the service account token controller (#87029, @liggitt)
- CustomResourceDefinition status fields are no longer required for client validation when submitting manifests. (#87213, @hasheddan)
- All apiservers log request lines in a more greppable format. (#87203, @lavalamp)
- provider/azure: Network security groups can now be in a separate resource group. (#87035, @CecileRobertMichon)
- Cleaned up the output from `kubectl describe CSINode <name>`. (#85283, @huffmanca)
- Fixed the following (#84265, @bhagwat070919)
  - - AWS Cloud Provider attempts to delete LoadBalancer security group it didn't provision
  - - AWS Cloud Provider creates default LoadBalancer security group even if annotation [service.beta.kubernetes.io/aws-load-balancer-security-groups] is present
- kubelet: resource metrics endpoint `/metrics/resource/v1alpha1` as well as all metrics under this endpoint have been deprecated. (#86282, @RainbowMango)
  - Please convert to the following metrics emitted by endpoint `/metrics/resource`:
  - - scrape_error -> scrape_error
  - - node_cpu_usage_seconds_total -> node_cpu_usage_seconds
  - - node_memory_working_set_bytes -> node_memory_working_set_bytes
  - - container_cpu_usage_seconds_total -> container_cpu_usage_seconds
  - - container_memory_working_set_bytes -> container_memory_working_set_bytes

    - - scrape_error -> scrape_error
- You can now pass "-node-ip ::" to kubelet to indicate that it should autodetect an IPv6 address to use as the node's primary address. (#85850, @danwinship)
- kubeadm: support automatic retry after failing to pull image (#86899, @SataQiu)
- TODO (#87044, @jennybuckley)
- Improved yaml parsing performance (#85458, @cjcullen)
- Fixed a bug which could prevent a provider ID from ever being set for node if an error occurred determining the provider ID when the node was added. (#87043, @zjs)
- fix a regression in kubenet that prevent pods to obtain ip addresses (#85993, @chendotjs)
- Bind kube-dns containers to linux nodes to avoid Windows scheduling (#83358, @wawa0210)
- The following features are unconditionally enabled and the corresponding `--feature-gates` flags have been removed: `PodPriority`, `TaintNodesByCondition`, `ResourceQuotaScopeSelectors` and `ScheduleDaemonSetPods` (#86210, @draveness)
- Bind dns-horizontal containers to linux nodes to avoid Windows scheduling on kubernetes cluster includes linux nodes and windows nodes (#83364, @wawa0210)
- fix kubectl annotate error when local=true is set (#86952, @zhouya0)
- Bug fixes: (#84163, @david-tigera)
    - Make sure we include latest packages node #351 (@caseydavenport)
- fix kuebctl apply set-last-applied namespaces error (#86474, @zhouya0)
- Add VolumeBinder method to FrameworkHandle interface, which allows user to get the volume binder when implementing scheduler framework plugins. (#86940, @skilxn-go)
- elasticsearch supports automatically setting the advertise address (#85944, @SataQiu)
- If a serving certificates param specifies a name that is an IP for an SNI certificate, it will have priority for replying to server connections. (#85308, @deads2k)
- kube-proxy: Added dual-stack IPv4/IPv6 support to the iptables proxier. (#82462, @vllry)
- Azure VMSS/VMSSVM clients now suppress requests on throttling (#86740, @feiskyer)
- New metric kubelet_pleg_last_seen_seconds to aid diagnosis of PLEG not healthy issues. (#86251, @bboreham)
- For subprotocol negotiation, both client and server protocol is required now. (#86646, @tedyu)
- kubeadm: use bind-address option to configure the kube-controller-manager and kube-scheduler http probes (#86493, @aojea)
- Marked scheduler's metrics scheduling_algorithm_predicate_evaluation_seconds and (#86584, @xiaoanyunfei)
    - scheduling_algorithm_priority_evaluation_seconds as deprecated. Those are replaced by framework_extension_point_duration_seconds[extenstion_point="Filter"]

and framework_extension_point_duration_seconds[extenstion_point="Score"] respectively.
- Marked scheduler's scheduling_duration_seconds Summary metric as deprecated (#86586, @xiaoanyunfei)
- Add instructions about how to bring up e2e test cluster (#85836, @YangLu1031)
- If a required flag is not provided to a command, the user will only see the required flag error message, instead of the entire usage menu. (#86693, @sallyom)
- kubeadm: tolerate whitespace when validating certificate authority PEM data in kubeconfig files (#86705, @neolit123)
- kubeadm: add support for the "ci/k8s-master" version label as a replacement for "ci-cross/*", which no longer exists. (#86609, @Pensu)
- Fix EndpointSlice controller race condition and ensure that it handles external changes to EndpointSlices. (#85703, @robscott)
- Fix nil pointer dereference in azure cloud provider (#85975, @ldx)
- fix: azure disk could not mounted on Standard_DC4s/DC2s instances (#86612, @andyzhangx)
- Fixes v1.17.0 regression in -service-cluster-ip-range handling with IPv4 ranges larger than 65536 IP addresses (#86534, @liggitt)
- Adds back support for AlwaysCheckAllPredicates flag. (#86496, @ahg-g)
- Azure global rate limit is switched to per-client. A set of new rate limit configure options are introduced, including routeRateLimit, SubnetsRateLimit, InterfaceRateLimit, RouteTableRateLimit, LoadBalancerRateLimit, PublicIPAddressRateLimit, SecurityGroupRateLimit, VirtualMachineRateLimit, StorageAccountRateLimit, DiskRateLimit, SnapshotRateLimit, VirtualMachineScaleSetRateLimit and VirtualMachineSizeRateLimit. (#86515, @feiskyer)
  - The original rate limit options would be default values for those new client's rate limiter.
- Fix issue #85805 about resource not found in azure cloud provider when lb specified in other resource group. (#86502, @levimm)
- `AlwaysCheckAllPredicates` is deprecated in scheduler Policy API. (#86369, @Huang-Wei)
- Kubernetes KMS provider for data encryption now supports disabling the in-memory data encryption key (DEK) cache by setting cachesize to a negative value. (#86294, @enj)
- option `preConfiguredBackendPoolLoadBalancerTypes` is added to azure cloud provider for the pre-configured load balancers, possible values: `""`, `"internal"`, "external"`,"all"` (#86338, @gossion)
- Promote StartupProbe to beta for 1.18 release (#83437, @matthyx)
- Fixes issue where AAD token obtained by kubectl is incompatible with on-behalf-of flow and oidc. (#86412, @weinong)
  - The audience claim before this fix has "spn:" prefix. After this fix, "spn:" prefix is omitted.
- change CounterVec to Counter about PLEGDiscardEvent (#86167, @yiyang5055)
- hollow-node do not use remote CRI anymore (#86425, @jkaniuk)
- hollow-node use fake CRI (#85879, @gongguan)

v1.18.0-alpha.1

Documentation

Downloads for v1.18.0-alpha.1

filename	sha512 hash
kubernetes.tar.gz	0c4904efc7f4f1436119c91dc1b6c93b3bd9c7490362a394bff10099c18
kubernetes-src.tar.gz	0a50fc6816c730ca5ae4c4f26d5ad7b049607d29f6a782a4e5b4b05ac50

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	c6d75f7f3f20bef17fc7564a619b54e6f4a673d041b7c9ec93663763a
kubernetes-client-darwin-amd64.tar.gz	ca1f19db289933beace6daee6fc30af19b0e260634ef6e89f773464a0
kubernetes-client-linux-386.tar.gz	af2e673653eb39c3f24a54efc68e1055f9258bdf6cf8fea42faf42c05
kubernetes-client-linux-amd64.tar.gz	9009032c3f94ac8a78c1322a28e16644ce3b20989eb762685a1819148
kubernetes-client-linux-arm.tar.gz	afba9595b37a3f2eead6e3418573f7ce093b55467dce4da0b8de86002
kubernetes-client-linux-arm64.tar.gz	04fc3b2fe3f271807f0bc6c61be52456f26a1af904964400be819b791
kubernetes-client-linux-ppc64le.tar.gz	04c7edab874b33175ff7bebfff5b3a032bc6eb088fcd7387ffcd5b3fa
kubernetes-client-linux-s390x.tar.gz	499287dbbc33399a37b9f3b35e0124ff20b17b6619f25a207ee9c606e
kubernetes-client-windows-386.tar.gz	cf84aeddf00f126fb13c0436b116dd0464a625659e44c84bf863517db
kubernetes-client-windows-amd64.tar.gz	69f20558ccd5cd6dbaccf29307210db4e687af21f6d71f68c69d3a397

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	3f29df2ce904a0f10db4c1d7a425a36f420867b595da3fa158ae430bfead90
kubernetes-server-linux-arm.tar.gz	4a21073b2273d721fbf062c254840be5c8471a010bcc0c731b101729e36e61

filename	sha512 hash
kubernetes-server-linux-arm64.tar.gz	7f1cb6d721bedc90e28b16f99bea7e59f5ad6267c31ef39c14d34db6ad6aa
kubernetes-server-linux-ppc64le.tar.gz	8f2b552030b5274b1c2c7c166eacd5a14b0c6ca0f23042f4c52efe87e22a1
kubernetes-server-linux-s390x.tar.gz	8d9f2c96f66edafb7c8b3aa90960d29b41471743842aede6b47b3b2e61f43

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	84194cb081d1502f8ca68143569f9707d96f1a28fcf0c574ebd203321463a
kubernetes-node-linux-arm.tar.gz	0091e108ab94fd8683b89c597c4fdc2fbf4920b007cfcd5297072c44bc3a2
kubernetes-node-linux-arm64.tar.gz	b7e85682cc2848a35d52fd6f01c247f039ee1b5dd03345713821ea10a7fa9
kubernetes-node-linux-ppc64le.tar.gz	cd1f0849e9c62b5d2c93ff0cebf58843e178d8a88317f45f76de0db5ae020
kubernetes-node-linux-s390x.tar.gz	e1e697a34424c75d75415b613b81c8af5f64384226c5152d869f12fd7db1a
kubernetes-node-windows-amd64.tar.gz	c725a19a4013c74e22383ad3fb4cb799b3e161c4318fdad066daf806730a8

Changelog since v1.17.0

Action Required

- action required (#85363, @immutableT)
  - 1. Currently, if users were to explicitly specify CacheSize of 0 for KMS provider, they would end-up with a provider that caches up to 1000 keys. This PR changes this behavior.
  - Post this PR, when users supply 0 for CacheSize this will result in a validation error.
  - 2. CacheSize type was changed from int32 to *int32. This allows defaulting logic to differentiate between cases where users explicitly supplied 0 vs. not supplied any value.

    - 3. KMS Provider's endpoint (path to Unix socket) is now validated when the EncryptionConfiguration files is loaded. This used to be handled by the GRPCService.

Other notable changes

- fix: azure data disk should use same key as os disk by default (#86351, @andyzhangx)
- New flag `--show-hidden-metrics-for-version` in kube-proxy can be used to show all hidden metrics that deprecated in the previous minor release. (#85279, @RainbowMango)
- Remove cluster-monitoring addon (#85512, @serathius)
- Changed core_pattern on COS nodes to be an absolute path. (#86329, @mml)
- Track mount operations as uncertain if operation fails with non-final error (#82492, @gnufied)
- add kube-proxy flags -ipvs-tcp-timeout, -ipvs-tcpfin-timeout, -ipvs-udp-timeout to configure IPVS connection timeouts. (#85517, @andrewsykim)
- The sample-apiserver aggregated conformance test has updated to use the Kubernetes v1.17.0 sample apiserver (#84735, @liggitt)
- The underlying format of the `CPUManager` state file has changed. Upgrades should be seamless, but any third-party tools that rely on reading the previous format need to be updated. (#84462, @klueska)
- kubernetes will try to acquire the iptables lock every 100 msec during 5 seconds instead of every second. This specially useful for environments using kube-proxy in iptables mode with a high churn rate of services. (#85771, @aojea)
- Fixed a panic in the kubelet cleaning up pod volumes (#86277, @tedyu)
- azure cloud provider cache TTL is configurable, list of the azure cloud provider is as following: (#86266, @zqingqing1)
    - - "availabilitySetNodesCacheTTLInSeconds"
    - - "vmssCacheTTLInSeconds"
    - - "vmssVirtualMachinesCacheTTLInSeconds"
    - - "vmCacheTTLInSeconds"
    - - "loadBalancerCacheTTLInSeconds"
    - - "nsgCacheTTLInSeconds"
    - - "routeTableCacheTTLInSeconds"
- Fixes kube-proxy when EndpointSlice feature gate is enabled on Windows. (#86016, @robscott)
- Fixes wrong validation result of NetworkPolicy PolicyTypes (#85747, @tnqn)
- Fixes an issue with kubelet-reported pod status on deleted/recreated pods. (#86320, @liggitt)
- kube-apiserver no longer serves the following deprecated APIs: (#85903, @liggitt) * All resources under `apps/v1beta1` and `apps/v1beta2` - use `apps/v1` instead * `daemonsets`, `deployments`, `replicasets` resources under `extensions/v1beta1` - use `apps/v1` instead * `networkpolicies` resources under `extensions/v1beta1` - use `networking.k8s.io/v1` instead * `podsecuritypolicies` resources under `extensions/v1beta1` - use `policy/v1beta1` instead

- kubeadm: fix potential panic when executing "kubeadm reset" with a corrupted kubelet.conf file (#86216, @neolit123)
- Fix a bug in port-forward: named port not working with service (#85511, @oke-py)
- kube-proxy no longer modifies shared EndpointSlices. (#86092, @robscott)
- allow for configuration of CoreDNS replica count (#85837, @pickledrick)
- Fixed a regression where the kubelet would fail to update the ready status of pods. (#84951, @tedyu)
- Resolves performance regression in client-go discovery clients constructed using `NewDiscoveryClientForConfig` or `NewDiscoveryClientForConfigOrDie`. (#86168, @liggitt)
- Make error message and service event message more clear (#86078, @feiskyer)
- e2e-test-framework: add e2e test namespace dump if all tests succeed but the cleanup fails. (#85542, @schrodit)
- SafeSysctlWhitelist: add net.ipv4.ping_group_range (#85463, @AkihiroSuda)
- kubelet: the metric process_start_time_seconds be marked as with the ALPHA stability level. (#85446, @RainbowMango)
- API request throttling (due to a high rate of requests) is now reported in the kubelet (and other component) logs by default. The messages are of the form (#80649, @RobertKrawitz)
    - Throttling request took 1.50705208s, request: GET:
    - The presence of large numbers of these messages, particularly with long delay times, may indicate to the administrator the need to tune the cluster accordingly.
- Fix API Server potential memory leak issue in processing watch request. (#85410, @answer1991)
- Verify kubelet & kube-proxy can recover after being killed on Windows nodes (#84886, @YangLu1031)
- Fixed an issue that the scheduler only returns the first failure reason. (#86022, @Huang-Wei)
- kubectl/drain: add skip-wait-for-delete-timeout option. (#85577, @michaelgugino)
    - If pod DeletionTimestamp older than N seconds, skip waiting for the pod. Seconds must be greater than 0 to skip.
- Following metrics have been turned off: (#83841, @RainbowMango)
    - - kubelet_pod_worker_latency_microseconds
    - - kubelet_pod_start_latency_microseconds
    - - kubelet_cgroup_manager_latency_microseconds
    - - kubelet_pod_worker_start_latency_microseconds
    - - kubelet_pleg_relist_latency_microseconds
    - - kubelet_pleg_relist_interval_microseconds
    - - kubelet_eviction_stats_age_microseconds
    - - kubelet_runtime_operations
    - - kubelet_runtime_operations_latency_microseconds
    - - kubelet_runtime_operations_errors
    - - kubelet_device_plugin_registration_count
    - - kubelet_device_plugin_alloc_latency_microseconds
    - - kubelet_docker_operations

    - - kubelet_docker_operations_latency_microseconds
    - - kubelet_docker_operations_errors
    - - kubelet_docker_operations_timeout
    - - network_plugin_operations_latency_microseconds
- - Renamed Kubelet metric certificate_manager_server_expiration_seconds to certificate_manager_server_ttl_seconds and changed to report the second until expiration at read time rather than absolute time of expiry. (#85874, @sambdavidson)
    - - Improved accuracy of Kubelet metric rest_client_exec_plugin_ttl_seconds.
- Bind metadata-agent containers to linux nodes to avoid Windows scheduling on kubernetes cluster includes linux nodes and windows nodes (#83363, @wawa0210)
- Bind metrics-server containers to linux nodes to avoid Windows scheduling on kubernetes cluster includes linux nodes and windows nodes (#83362, @wawa0210)
- During initialization phase (preflight), kubeadm now verifies the presence of the conntrack executable (#85857, @hnanni)
- VMSS cache is added so that less chances of VMSS GET throttling (#85885, @nilo19)
- Update go-winio module version from 0.4.11 to 0.4.14 (#85739, @wawa0210)
- Fix LoadBalancer rule checking so that no unexpected LoadBalancer updates are made (#85990, @feiskyer)
- kubectl drain node -dry-run will list pods that would be evicted or deleted (#82660, @sallyom)
- Windows nodes on GCE can use TPM-based authentication to the master. (#85466, @pjh)
- kubectl/drain: add disable-eviction option. (#85571, @michaelgugino)
    - Force drain to use delete, even if eviction is supported. This will bypass checking PodDisruptionBudgets, and should be used with caution.
- kubeadm now errors out whenever a not supported component config version is supplied for the kubelet and kube-proxy (#85639, @rosti)
- Fixed issue with addon-resizer using deprecated extensions APIs (#85793, @bskiba)
- Includes FSType when describing CSI persistent volumes. (#85293, @huffmanca)
- kubelet now exports a "server_expiration_renew_failure" and "client_expiration_renew_failure" metric counter if the certificate rotations cannot be performed. (#84614, @rphillips)
- kubeadm: don't write the kubelet environment file on "upgrade apply" (#85412, @boluisa)
- fix azure file AuthorizationFailure (#85475, @andyzhangx)
- Resolved regression in admission, authentication, and authorization webhook performance in v1.17.0-rc.1 (#85810, @liggitt)
- kubeadm: uses the apiserver AdvertiseAddress IP family to choose the etcd endpoint IP family for non external etcd clusters (#85745, @aojea)
- kubeadm: Forward cluster name to the controller-manager arguments (#85817, @ereslibre)

- Fixed "requested device X but found Y" attach error on AWS. (#85675, @jsafrane)
- addons: elasticsearch discovery supports IPv6 (#85543, @SataQiu)
- kubeadm: retry `kubeadm-config` ConfigMap creation or mutation if the apiserver is not responding. This will improve resiliency when joining new control plane nodes. (#85763, @ereslibre)
- Update Cluster Autoscaler to 1.17.0; changelog: https://github.com/kubernetes/autoscaler/releases/tag/cluster-autoscaler-1.17.0 (#85610, @losipiuk)
- Filter published OpenAPI schema by making nullable, required fields non-required in order to avoid kubectl to wrongly reject null values. (#85722, @sttts)
- kubectl set resources will no longer return an error if passed an empty change for a resource. (#85490, @sallyom)
    - kubectl set subject will no longer return an error if passed an empty change for a resource.
- kube-apiserver: fixed a conflict error encountered attempting to delete a pod with gracePeriodSeconds=0 and a resourceVersion precondition (#85516, @michaelgugino)
- kubeadm: add a upgrade health check that deploys a Job (#81319, @neolit123)
- kubeadm: make sure images are pre-pulled even if a tag did not change but their contents changed (#85603, @bart0sh)
- kube-apiserver: Fixes a bug that hidden metrics can not be enabled by the command-line option `--show-hidden-metrics-for-version`. (#85444, @RainbowMango)
- kubeadm now supports automatic calculations of dual-stack node cidr masks to kube-controller-manager. (#85609, @Arvinderpal)
- Fix bug where EndpointSlice controller would attempt to modify shared objects. (#85368, @robscott)
- 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)
- kubeadm: reset raises warnings if it cannot delete folders (#85265, @SataQiu)
- Wait for kubelet & kube-proxy to be ready on Windows node within 10s (#85228, @YangLu1031)

Feedback

Page last modified on March 31, 2020 at 2:04 AM PST by update release notes to v1.18.0 (#19923) (Page History)

Edit This Page

Kubernetes version and version skew support policy

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

Supported versions

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

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

Applicable fixes, including security fixes, may be backported to those three release branches, depending on severity and feasibility. Patch releases are cut from those branches at a regular cadence, or as needed. This decision is owned by the patch release 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:
>
> - they must be upgraded within one minor version of `kube-apiserver` before the control plane can be upgraded
> - it increases the likelihood of running `kubelet` versions older than the three maintained minor releases

Feedback

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: `kube ctl 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 v1.18.0, you would run the following:

```
minikube start --kubernetes-version v1.18.0
```

Specifying the VM driver

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

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

Minikube supports the following drivers:

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

- virtualbox
- vmwarefusion
- docker (EXPERIMENTAL)
- 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)
- parallels (driver installation)
- 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
```

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

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

Or you can use the extended version:

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

Use local images by re-using the Docker daemon

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

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

To work with the Docker daemon on your Mac/Linux host, run the last line from `minikube docker-env`.

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

```
docker ps
```

> **Note:**
>
> On Centos 7, Docker may report the following error:
>
> ```
> Could not read CA certificate "/etc/docker/ca.pem":
> open /etc/docker/ca.pem: no such file or directory
> ```
>
> You can fix this by updating /etc/sysconfig/docker to ensure that Minikube's environment changes are respected:
>
> ```
> < DOCKER_CERT_PATH=/etc/docker
> ---
> > if [ -z "${DOCKER_CERT_PATH}" ]; then
> >   DOCKER_CERT_PATH=/etc/docker
> > fi
> ```

Configuring Kubernetes

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

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

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

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

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

Examples

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

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

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

Stopping a Cluster

The `minikube stop` command can be used to stop your cluster. This command shuts down the Minikube Virtual Machine, but preserves all cluster state and data. Starting the cluster again will restore it to its previous state.

Deleting a Cluster

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

Upgrading Minikube

If you are using macOS, see Upgrading Minikube to upgrade your existing minikube installation.

Interacting with Your Cluster

Kubectl

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

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

`kubectl config use-context minikube`,

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

Dashboard

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

```
minikube dashboard
```

Services

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

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

Networking

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

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

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

Persistent Volumes

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

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

- `/data`
- `/var/lib/minikube`
- `/var/lib/docker`

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

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

Mounted Host Folders

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

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

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

Private Container Registries

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

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

Add-ons

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

Using Minikube with an HTTP Proxy

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

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

For example:

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

If your Virtual Machine address is 192.168.99.100, then chances are your proxy settings will prevent `kubectl` from directly reaching it. To by-pass

proxy configuration for this IP address, you should modify your no_proxy settings. You can do so with:

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

Known Issues

Features that require multiple nodes will not work in Minikube.

Design

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

For more information about Minikube, see the proposal.

Additional Links

- **Goals and Non-Goals**: For the goals and non-goals of the Minikube project, please see our roadmap.
- **Development Guide**: See CONTRIBUTING.md for an overview of how to send pull requests.
- **Building Minikube**: For instructions on how to build/test Minikube from source, see the build guide.
- **Adding a New Dependency**: For instructions on how to add a new dependency to Minikube, see the adding dependencies guide.
- **Adding a New Addon**: For instructions on how to add a new addon for Minikube, see the adding an addon guide.
- **MicroK8s**: Linux users wishing to avoid running a virtual machine may consider MicroK8s as an alternative.

Community

Contributions, questions, and comments are all welcomed and encouraged! Minikube developers hang out on Slack in the #minikube channel (get an invitation here). We also have the kubernetes-dev Google Groups mailing list. If you are posting to the list please prefix your subject with "minikube: ".

Feedback

Was this page helpful?

Yes No

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

Analytics

Create an Issue Edit This Page
Page last modified on March 27, 2020 at 2:06 AM PST by Update the Upgrading Minikube topic (#19862) (Page History)

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.

Analytics

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:

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

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

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

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

Applicability

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

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

Cgroup drivers

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

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

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

> **Caution:** Changing the cgroup driver of a Node that has joined a cluster is highly unrecommended. If the kubelet has created Pods

> using the semantics of one cgroup driver, changing the container runtime to another cgroup driver can cause errors when trying to re-create the PodSandbox for such existing Pods. Restarting the kubelet may not solve such errors. The recommendation is to drain the Node from its workloads, remove it from the cluster and re-join it.

Docker

On each of your machines, install Docker. Version 19.03.8 is recommended, but 1.13.1, 17.03, 17.06, 17.09, 18.06 and 18.09 are known to work as well. Keep track of the latest verified Docker version in the Kubernetes release notes.

Use the following commands to install Docker on your system:

- Ubuntu 16.04+
- CentOS/RHEL 7.4+

```
# Install Docker CE
## Set up the repository:
### Install packages to allow apt to use a repository over HTTPS
apt-get update && apt-get install -y \
  apt-transport-https ca-certificates curl software-properties-
common gnupg2

### Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-
key add -

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

## Install Docker CE.
apt-get update && apt-get install -y \
  containerd.io=1.2.13-1 \
  docker-ce=5:19.03.8~3-0~ubuntu-$(lsb_release -cs) \
  docker-ce-cli=5:19.03.8~3-0~ubuntu-$(lsb_release -cs)

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

```
  "storage-driver": "overlay2"
}
EOF

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

# Restart docker.
systemctl daemon-reload
systemctl restart docker
```

```
# Install Docker CE
## Set up the repository
### Install required packages.
yum install -y yum-utils device-mapper-persistent-data lvm2

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

## Install Docker CE.
yum update -y && yum install -y \
  containerd.io-1.2.13 \
  docker-ce-19.03.8 \
  docker-ce-cli-19.03.8

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

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

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

# Restart Docker
systemctl daemon-reload
systemctl restart docker
```

Refer to the official Docker installation guides for more information.

CRI-O

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

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

> **Note:** The CRI-O major and minor versions must match the Kubernetes major and minor versions. For more information, see the CRI-O compatiblity matrix.

Prerequisites

```
modprobe overlay
modprobe br_netfilter

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

sysctl --system
```

- Debian
- Ubuntu 18.04, 19.04 and 19.10
- CentOS/RHEL 7.4+
- openSUSE Tumbleweed

```
# Debian Unstable/Sid
echo 'deb http://download.opensuse.org/repositories/devel:/
kubic:/libcontainers:/stable/Debian_Unstable/ /' > /etc/apt/
sources.list.d/devel:kubic:libcontainers:stable.list
wget -nv https://download.opensuse.org/repositories/
devel:kubic:libcontainers:stable/Debian_Unstable/Release.key -O-
| sudo apt-key add -

# Debian Testing
echo 'deb http://download.opensuse.org/repositories/devel:/
kubic:/libcontainers:/stable/Debian_Testing/ /' > /etc/apt/
sources.list.d/devel:kubic:libcontainers:stable.list
wget -nv https://download.opensuse.org/repositories/
devel:kubic:libcontainers:stable/Debian_Testing/Release.key -O-
| sudo apt-key add -

# Debian 10
echo 'deb http://download.opensuse.org/repositories/devel:/
kubic:/libcontainers:/stable/Debian_10/ /' > /etc/apt/
sources.list.d/devel:kubic:libcontainers:stable.list
wget -nv https://download.opensuse.org/repositories/
```

```
devel:kubic:libcontainers:stable/Debian_10/Release.key -O- |
sudo apt-key add -

# Raspbian 10
echo 'deb http://download.opensuse.org/repositories/devel:/
kubic:/libcontainers:/stable/Raspbian_10/ /' > /etc/apt/
sources.list.d/devel:kubic:libcontainers:stable.list
wget -nv https://download.opensuse.org/repositories/
devel:kubic:libcontainers:stable/Raspbian_10/Release.key -O- |
sudo apt-key add -

# Install CRI-O
sudo apt-get install cri-o-1.17
```

```
# Setup repository
. /etc/os-release
sudo sh -c "echo 'deb http://download.opensuse.org/repositories/
devel:/kubic:/libcontainers:/stable/x${NAME}_${VERSION_ID}/ /'
> /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list"
wget -nv https://download.opensuse.org/repositories/
devel:kubic:libcontainers:stable/x${NAME}_${VERSION_ID}/
Release.key -O- | sudo apt-key add -
sudo apt-get update

# Install CRI-O
sudo apt-get install cri-o-1.17
```

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

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

```
sudo zypper install cri-o
```

Start CRI-O

```
systemctl daemon-reload
systemctl start crio
```

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

Containerd

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

Use the following commands to install Containerd on your system:

Prerequisites

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

modprobe overlay
modprobe br_netfilter

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

sysctl --system
```

Install containerd

- Ubuntu 16.04
- CentOS/RHEL 7.4+

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

### Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-
key add -

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

## Install containerd
apt-get update && apt-get install -y containerd.io

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

# Restart containerd
systemctl restart containerd
```

```
# Install containerd
## Set up the repository
### Install required packages
yum install -y yum-utils device-mapper-persistent-data lvm2

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

## Install containerd
yum update -y && yum install -y containerd.io

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

# Restart containerd
systemctl restart containerd
```

systemd

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

Other CRI runtimes: frakti

Refer to the Frakti QuickStart guide for more information.

Feedback

Was this page helpful?

Yes No

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

Installing kubeadm

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

Before you begin

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

Verify the MAC address and product_uuid are unique for every node

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

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

Check network adapters

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

Letting iptables see bridged traffic

As a requirement for your Linux Node's iptables to correctly see bridged traffic, you should ensure `net.bridge.bridge-nf-call-iptables` is set to 1 in your `sysctl` config, e.g.

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

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

For more details please see the Network Plugin Requirements page.

Check required ports

Control-plane node(s)

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

Worker node(s)

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

â€ Default port range for NodePort Services.

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

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

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

Installing runtime

To run containers in Pods, Kubernetes uses a container runtimeThe container runtime is the software that is responsible for running containers.
.

- Linux nodes
- other operating systems

By default, Kubernetes uses the Container Runtime InterfaceAn API for container runtimes to integrate with kubelet (CRI) to interface with your chosen container runtime.

If you don't specify a runtime, kubeadm automatically tries to detect an installed container runtime by scanning through a list of well known Unix domain sockets. The following table lists container runtimes and their associated socket paths:

Runtime	Path to Unix domain socket
Docker	`/var/run/docker.sock`
containerd	`/run/containerd/containerd.sock`
CRI-O	`/var/run/crio/crio.sock`

If both Docker and containerd are detected, Docker takes precedence. This is needed because Docker 18.09 ships with containerd and both are detectable even if you only installed Docker. If any other two or more runtimes are detected, kubeadm exits with an error.

The kubelet integrates with Docker through the built-in `dockershim` CRI implementation.

See container runtimes for more information.

By default, kubeadm uses DockerDocker is a software technology providing operating-system-level virtualization also known as containers. as the container runtime. The kubelet integrates with Docker through the built-in `dockershim` CRI implementation.

See container runtimes for more information.

Installing kubeadm, kubelet and kubectl

You will install these packages on all of your machines:

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

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

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

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

For more information on version skews, see:

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

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

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

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-
el7-$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF

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

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

systemctl enable --now kubelet
```

Note:

- Setting SELinux in permissive mode by running `setenforce 0` and `sed ...` effectively disables it. This is required to allow containers to access the host filesystem, which is needed by pod networks for example. You have to do this until SELinux support is improved in the kubelet.

Install CNI plugins (required for most pod network):

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

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

```
CRICTL_VERSION="v1.17.0"
mkdir -p /opt/bin
curl -L "https://github.com/kubernetes-sigs/cri-tools/releases/download/${CRICTL_VERSION}/crictl-${CRICTL_VERSION}-linux-amd64.tar.gz" | tar -C /opt/bin -xz
```

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

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

mkdir -p /opt/bin
cd /opt/bin
curl -L --remote-name-all https://storage.googleapis.com/kubernetes-release/release/${RELEASE}/bin/linux/amd64/{kubeadm,kubelet,kubectl}
chmod +x {kubeadm,kubelet,kubectl}

RELEASE_VERSION="v0.2.7"
curl -sSL "https://raw.githubusercontent.com/kubernetes/release/${RELEASE_VERSION}/cmd/kubepkg/templates/latest/deb/kubelet/lib/systemd/system/kubelet.service" | sed "s:/usr/bin:/opt/bin:g" > /etc/systemd/system/kubelet.service
mkdir -p /etc/systemd/system/kubelet.service.d
curl -sSL "https://raw.githubusercontent.com/kubernetes/release/${RELEASE_VERSION}/cmd/kubepkg/templates/latest/deb/kubeadm/10-kubeadm.conf" | sed "s:/usr/bin:/opt/bin:g" > /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

Enable and start `kubelet`:

```
systemctl enable --now kubelet
```

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

Configure cgroup driver used by kubelet on control-plane node

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

If you are using a different CRI, you have to modify the file `/etc/default/kubelet` (`/etc/sysconfig/kubelet` for CentOS, RHEL, Fedora) with your `cgroup-driver` value, like so:

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

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

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

Restarting the kubelet is required:

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

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

Troubleshooting

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

What's next

- Using kubeadm to Create a Cluster

Feedback

Was this page helpful?

Yes No

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

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.

Not possible to join a v1.18 Node to a v1.17 cluster due to missing RBAC

In v1.18 kubeadm added prevention for joining a Node in the cluster if a Node with the same name already exists. This required adding RBAC for the bootstrap-token user to be able to GET a Node object.

However this causes an issue where `kubeadm join` from v1.18 cannot join a cluster created by kubeadm v1.17.

To workaround the issue you have two options:

Execute `kubeadm init phase bootstrap-token` on a control-plane node using kubeadm v1.18. Note that this enables the rest of the bootstrap-token permissions as well.

or

Apply the following RBAC manually using `kubectl apply -f ...`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kubeadm:get-nodes
rules:
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kubeadm:get-nodes
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kubeadm:get-nodes
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:bootstrappers:kubeadm:default-node-token
```

`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 `if config` 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 `node-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 `node-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 node-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 v1.18.

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
```

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

To deploy Cilium you just need to run:

```
kubectl create -f https://raw.githubusercontent.com/cilium/cilium/v1.6/install/kubernetes/quick-install.yaml
```

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

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

The output is similar to this:

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

Cilium can be used as a replacement for kube-proxy, see Kubernetes without kube-proxy.

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

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

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

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

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

Make sure that your firewall rules allow UDP ports 8285 and 8472 traffic for all hosts participating in the overlay network. The Firewall section of Flannel's troubleshooting guide explains about this in more detail.

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

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml
```

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

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

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

For information on using the `kubeadm` tool to set up a Kubernetes cluster with Kube-router, please see the official setup guide.

For more information on setting up your Kubernetes cluster with Weave Net, please see [Integrating Kubernetes via the Addon]((https://www.weave.works/docs/net/latest/kube-addon/).

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

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

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

If your network is not working or CoreDNS is not in the `Running` state, check out the troubleshooting guide for `kubeadm`.

Control plane node isolation

By default, your cluster will not schedule Pods on the control-plane node for security reasons. If you want to be able to schedule Pods on the control-plane node, for example for a single-machine Kubernetes cluster for development, run:

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

With output looking something like:

```
node "test-01" untainted
taint "node-role.kubernetes.io/master:" not found
taint "node-role.kubernetes.io/master:" not found
```

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

Joining your nodes

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

- SSH to the machine
- Become root (e.g. `sudo su -`)

- Run the command that was output by `kubeadm init`. For example:

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

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

```
kubeadm token list
```

The output is similar to this:

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

```
authentication,  The default bootstrap  system:

signing          token generated by     bootstrappers:

   'kubeadm init'.        kubeadm:

                          default-node-token
```

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

```
kubeadm token create
```

The output is similar to this:

```
5didvk.d09sbcov8ph2amjw
```

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

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

The output is similar to:

```
8cb2de97839780a412b93877f8507ad6c94f73add17d5d7058e91741c9d5ec78
```

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

The output should look something like:

```
[preflight] Running pre-flight checks

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

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

Run 'kubectl get nodes' on control-plane to see this machine
join.
```

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

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

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

```
scp root@<control-plane-host>:/etc/kubernetes/admin.conf .
kubectl --kubeconfig ./admin.conf get nodes
```

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

(Optional) Proxying API Server to localhost

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

```
scp root@<control-plane-host>:/etc/kubernetes/admin.conf .
kubectl --kubeconfig ./admin.conf proxy
```

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

Clean up

If you used disposable servers for your cluster, for testing, you can switch those off and do no further clean up. You can use `kubectl config delete-cluster` to delete your local references to the cluster.

However, if you want to deprovision your cluster more cleanly, you should first drain the node and make sure that the node is empty, then deconfigure the node.

Remove the node

Talking to the control-plane node with the appropriate credentials, run:

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

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

```
kubeadm reset
```

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

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

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

```
ipvsadm -C
```

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

Clean up the control plane

You can use `kubeadm reset` on the control plane host to trigger a best-effort clean up.

See the `kubeadm reset` reference documentation for more information about this subcommand and its options.

What's next

- Verify that your cluster is running properly with Sonobuoy
- See Upgrading kubeadm clusters for details about upgrading your cluster using `kubeadm`.
- Learn about advanced `kubeadm` usage in the kubeadm reference documentation
- Learn more about Kubernetes concepts and `kubectl`.
- See the Cluster Networking page for a bigger list of Pod network add-ons.
- See the list of add-ons to explore other add-ons, including tools for logging, monitoring, network policy, visualization & control of your Kubernetes cluster.
- Configure how your cluster handles logs for cluster events and from applications running in Pods. See Logging Architecture for an overview of what is involved.

Feedback

- For bugs, visit the kubeadm GitHub issue tracker
- For support, visit the #kubeadm Slack channel
- General SIG Cluster Lifecycle development Slack channel: #sig-cluster-lifecycle

- SIG Cluster Lifecycle SIG information
- SIG Cluster Lifecycle mailing list: kubernetes-sig-cluster-lifecycle

Version skew policy

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

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

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

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

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

Limitations

Cluster resilience

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

Workarounds:

- Regularly back up etcd. The etcd data directory configured by kubeadm is at `/var/lib/etcd` on the control-plane node.

- Use multiple control-plane nodes. You can read Options for Highly Available topology to pick a cluster topology that provides higher availabilty.

Platform compatibility

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

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

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

Troubleshooting

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

Feedback

Was this page helpful?

Yes No

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

Analytics

Create an Issue Edit This Page
Page last modified on March 16, 2020 at 11:20 AM PST by improvements to the br_netfilter documentation for kubeadm (#19610) (Page History)

Edit This Page

Customizing control plane configuration with kubeadm

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

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

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

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

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

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

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

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

APIServer flags

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

Example usage:

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: v1.16.0
apiServer:
  extraArgs:
    advertise-address: 192.168.0.103
    anonymous-auth: "false"
    enable-admission-plugins: AlwaysPullImages,DefaultStorageClas
s
    audit-log-path: /home/johndoe/audit.log
```

ControllerManager flags

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

Example usage:

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: v1.16.0
controllerManager:
  extraArgs:
    cluster-signing-key-file: /home/johndoe/keys/ca.key
    bind-address: 0.0.0.0
    deployment-controller-sync-period: "50"
```

Scheduler flags

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

Example usage:

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

```
kubernetesVersion: v1.16.0
scheduler:
  extraArgs:
    address: 0.0.0.0
    config: /home/johndoe/schedconfig.yaml
    kubeconfig: /home/johndoe/kubeconfig.yaml
```

Feedback

Was this page helpful?

Yes No

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

Analytics

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

Edit This Page

Options for Highly Available topology

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

You can set up an HA cluster:

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

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

Stacked etcd topology

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

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

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

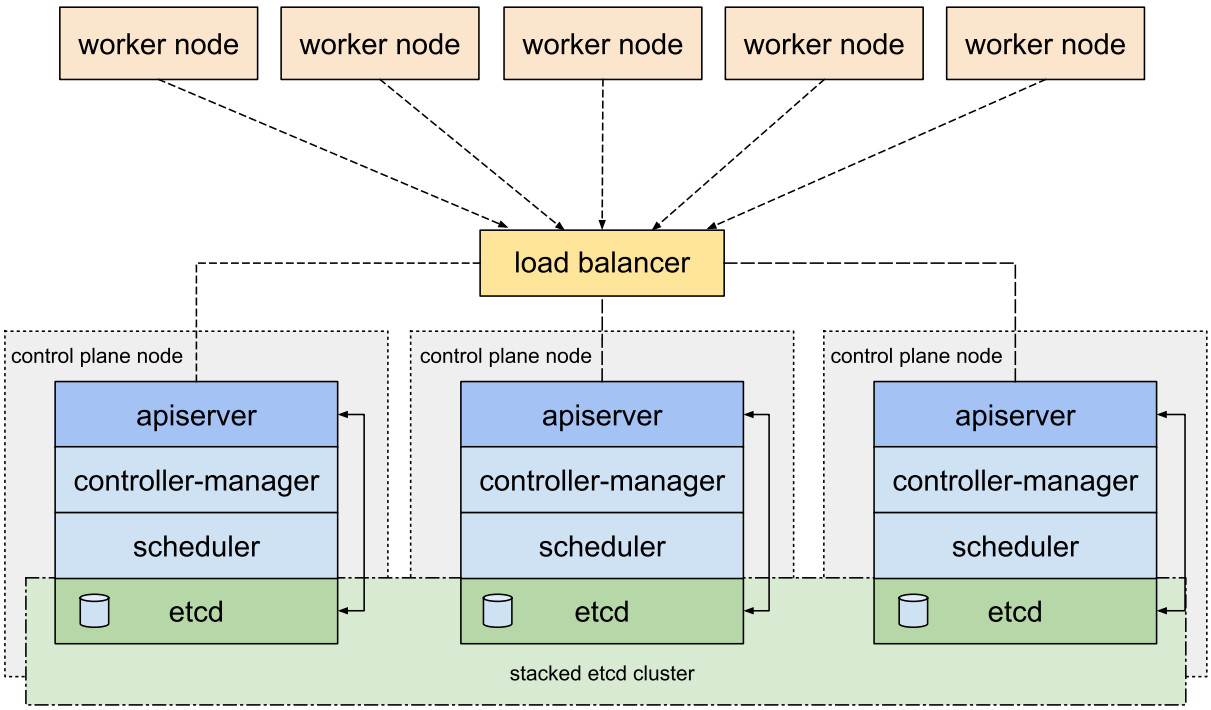
This topology couples the control planes and etcd members on the same nodes. It is simpler to set up than a cluster with external etcd nodes, and simpler to manage for replication.

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

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

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

kubeadm HA topology - stacked etcd



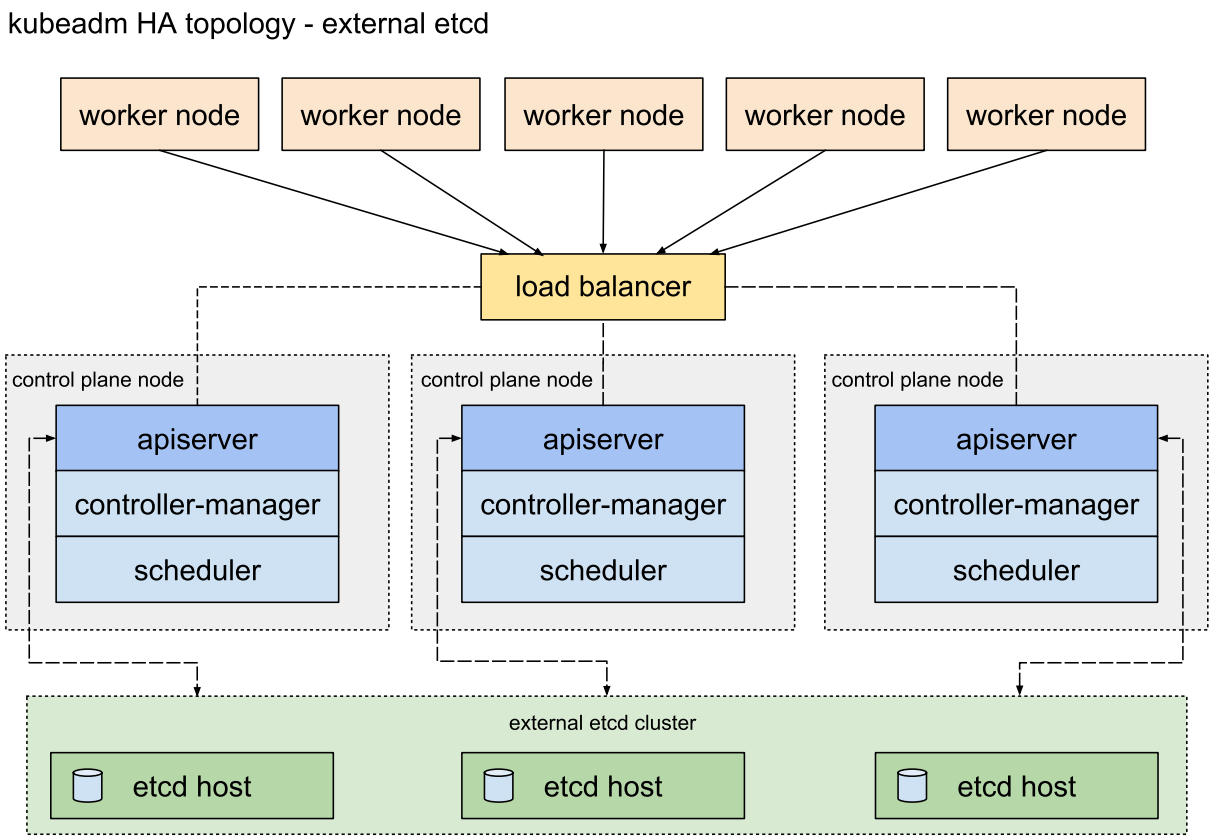
External etcd topology

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

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

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

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



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
```

```
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26aec866
```

Manual certificate distribution

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

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

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

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

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

2. Add your SSH identity to the session:

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

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

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

     ```
     ssh -A 10.0.0.7
     ```

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

     ```
     sudo -E -s
     ```

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

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

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

```
    scp /etc/kubernetes/pki/etcd/ca.crt "${USER}"@$host:etcd-
ca.crt
    # 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.

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/${HO
ST0}/kubeadmcfg.yaml
root@HOST1 $ kubeadm init phase etcd local --config=/home/
ubuntu/kubeadmcfg.yaml
root@HOST2 $ kubeadm init phase etcd local --config=/home/
ubuntu/kubeadmcfg.yaml
```

8. Optional: Check the cluster health

```
docker run --rm -it \
--net host \
-v /etc/kubernetes:/etc/kubernetes 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:

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

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

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

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

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

Kubelet configuration patterns

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

Propagating cluster-level configuration to each kubelet

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

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

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

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

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

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

Providing instance-specific configuration details

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

- The path to the DNS resolution file, as specified by the `--resolv-conf` kubelet configuration flag, may differ among operating systems, or depending on whether you are using `systemd-resolved`. If this path is wrong, DNS resolution will fail on the Node whose kubelet is configured incorrectly.
- The Node API object `.metadata.name` is set to the machine's hostname by default, unless you are using a cloud provider. You can use the `--hostname-override` flag to override the default behavior if you need to specify a Node name different from the machine's hostname.
- Currently, the kubelet cannot automatically detects the cgroup driver used by the CRI runtime, but the value of `--cgroup-driver` must match the cgroup driver used by the CRI runtime to ensure the health of the kubelet.
- Depending on the CRI runtime your cluster uses, you may need to specify different flags to the kubelet. For instance, when using Docker, you need to specify flags such as `--network-plugin=cni`, but if you are using an external runtime, you need to specify `--container-runtime=remote` and specify the CRI endpoint using the `--container-runtime-endpoint=<path>`.

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

Configure kubelets using kubeadm

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

By calling `kubeadm config print init-defaults --component-configs KubeletConfiguration` you can see all the default values for this structure.

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

Workflow when using `kubeadm init`

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

configuration file points to the client certificates that allow the kubelet to communicate with the API server. This addresses the need to propagate cluster-level configuration to each kubelet.

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

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

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

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

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

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

Workflow when using `kubeadm join`

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

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

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

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

The kubelet drop-in file for systemd

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

This configuration file installed by the `kubeadm` DEB or RPM package is written to `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` and is used by systemd. It augments the basic `kubelet.service` for RPM (resp. `kubelet.service` for DEB)):

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

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

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

Kubernetes binaries and package contents

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

Package name	Description
`kubeadm`	Installs the `/usr/bin/kubeadm` CLI tool and the kubelet drop-in file for the kubelet.
`kubelet`	Installs the `/usr/bin/kubelet` binary.
`kubectl`	Installs the `/usr/bin/kubectl` binary.
`kubernetes-cni`	Installs the official CNI binaries into the `/opt/cni/bin` directory.

Package name	Description
`cri-tools`	Installs the `/usr/bin/crictl` binary from the cri-tools git repository.

Feedback

Was this page helpful?

Yes No

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

Create an Issue Edit This Page
Page last modified on November 28, 2019 at 1:17 PM PST by Fixed the incorrect parameter of specifying the CRI endpoint for kubelet and corrected Default CNs (#17839) (Page History)

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

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

Analytics

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

Edit This Page

Installing Kubernetes with 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

- 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/$(c
url -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 following portion of the command with the specific kops version.

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

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
```

Download the latest release with the command:

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

To download a specific version of kops, replace the following portion of the command with the specific kops version.

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

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

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

Make the kops binary executable

```
chmod +x kops-linux-amd64
```

Move the kops binary in to your PATH.

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

You can also install kops using Homebrew.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(4/5) Build your cluster configuration

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

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

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

It prints commands you can use to explore further:

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

If this is your first time using kops, do spend a few minutes to try those out! An instance group is a set of instances, which will be registered as kubernetes nodes. On AWS this is implemented via auto-scaling-groups. You

can have several instance groups, for example if you wanted nodes that are a mix of spot and on-demand instances, or GPU and non-GPU instances.

(5/5) Create the cluster in AWS

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

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

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

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

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

Explore other add-ons

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

Cleanup

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

What's next

- Learn more about Kubernetes concepts and `kubectl`.
- Learn more about `kops` advanced usage for tutorials, best practices and advanced configuration options.
- Follow `kops` community discussions on Slack: community discussions
- Contribute to `kops` by addressing or raising an issue 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 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.

Analytics

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

Edit This Page

Installing Kubernetes with 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
  ◦ Custom Deployments

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.

Running Kubernetes on AWS EC2

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

- Support Level
- Further reading

Before you begin

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

Supported Production Grade Tools

- conjure-up is an open-source installer for Kubernetes that creates Kubernetes clusters with native AWS integrations on Ubuntu.

- Kubernetes Operations - Production Grade K8s Installation, Upgrades, and Management. Supports running Debian, Ubuntu, CentOS, and RHEL in AWS.

- CoreOS Tectonic includes the open-source Tectonic Installer that creates Kubernetes clusters with Container Linux nodes on AWS.

- CoreOS originated and the Kubernetes Incubator maintains a CLI tool, kube-aws, that creates and manages Kubernetes clusters with Container Linux nodes, using AWS tools: EC2, CloudFormation and Autoscaling.

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

Getting started with your cluster

Command line administration tool: kubectl

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

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

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

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

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

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

Examples

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

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

For more complete applications, please look in the examples directory

Scaling the cluster

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

Tearing down the cluster

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

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

Support Level

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

Further reading

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

Feedback

Was this page helpful?

Yes No

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

Running Kubernetes on Azure

Azure Kubernetes Service (AKS)

The Azure Kubernetes Service offers simple deployments for Kubernetes clusters.

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

Microsoft Azure Kubernetes Service

Custom Deployments: AKS-Engine

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

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

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

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

CoreOS Tectonic for Azure

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

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

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

Feedback

Was this page helpful?

Yes No

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

Running Kubernetes on CenturyLink Cloud

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.

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 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.

Feedback

Was this page helpful?

Yes No

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

Running Kubernetes on Tencent Kubernetes Engine

Tencent Kubernetes Engine

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

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

Custom Deployment

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

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

What's Next

To learn more, see the TKE documentation.

Feedback

Was this page helpful?

Yes No

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

Cloudstack

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

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

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

Prerequisites

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

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

```
yum install libselinux-python
```

cs is a python module for the CloudStack API.

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

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

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

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

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

Clone the playbook

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

Create a Kubernetes cluster

You simply need to run the playbook.

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

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

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

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

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

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

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

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

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

And you can list the machines in your cluster:

```
fleetctl list-machines
```

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

Support Level

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

Feedback

Was this page helpful?

Yes No

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

Analytics

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

Edit This Page

Kubernetes on DC/OS

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

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

- Official Mesosphere Guide

Official Mesosphere Guide

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

Feedback

Was this page helpful?

Yes No

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

Analytics

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

Edit This Page

oVirt

oVirt is a virtual datacenter manager that delivers powerful management of multiple virtual machines on multiple hosts. Using KVM and libvirt, oVirt

can be installed on Fedora, CentOS, or Red Hat Enterprise Linux hosts to set up and manage your virtual data center.

oVirt Cloud Provider Deployment

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

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

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

Using the oVirt Cloud Provider

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

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

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

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

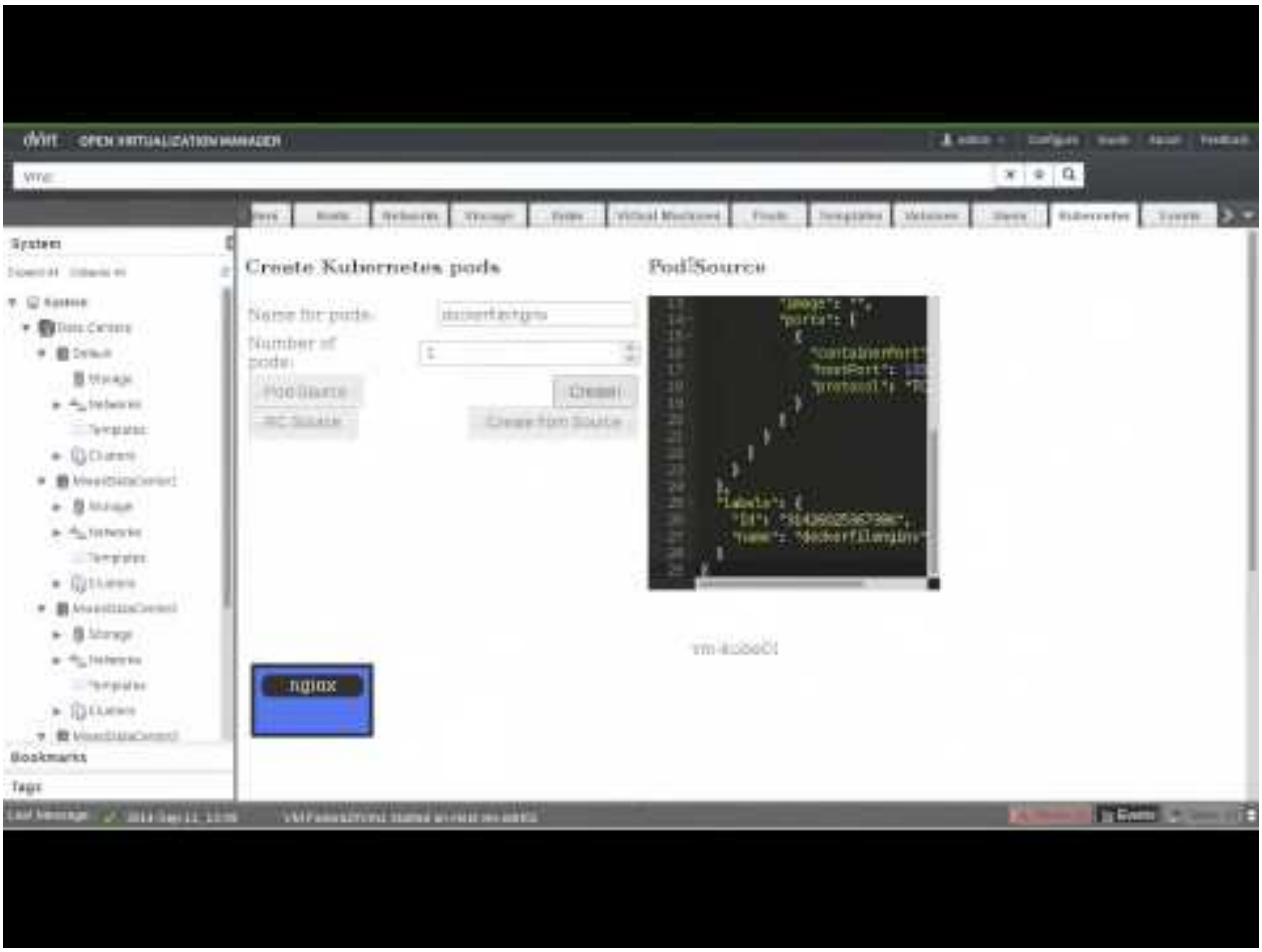
In the above example all the VMs tagged with the `kubernetes` label will be reported as nodes to Kubernetes.

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

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

oVirt Cloud Provider Screencast

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



Support Level

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

Feedback

Was this page helpful?

Yes No

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

Analytics

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

FEATURE STATE: `Kubernetes v1.14` stable
This feature is *stable*, meaning:

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

Docker EE-basic 18.09+ is the recommended container runtime for Windows Server 2019 / 1809 nodes running Kubernetes. This works with the dockershim code included in the kubelet.

CRI-ContainerD

FEATURE STATE: `Kubernetes v1.18` alpha
This feature is currently in a *alpha* state, meaning:

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

ContainerD is an OCI-compliant runtime that works with Kubernetes on Linux. Kubernetes v1.18 adds support for ContainerDA container runtime with an emphasis on simplicity, robustness and portability on Windows. Progress for ContainerD on Windows can be tracked at enhancements#1001.

Caution: ContainerD on Windows in Kubernetes v1.18 has the following known shortcomings:

- ContainerD does not have an official release with support for Windows; all development in Kubernetes has been performed against active ContainerD development branches. Production deployments should always use official releases that have been fully tested and are supported with security fixes.
- Group-Managed Service Accounts are not implemented when using ContainerD - see containerd/cri#1276.

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:

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

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

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

Networking

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

the management of containers whereas HNS is responsible for the management of networking resources such as:

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

The following service spec types are supported:

- NodePort
- ClusterIP
- LoadBalancer
- ExternalName

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

Network Driver	**Description**	**Container Packet Modifications**	**Network Plugins**	**Network Plugin Characteristics**
L2bridge	Containers are attached to an external vSwitch. Containers are attached to the underlay network, although the physical network doesn't need to learn the container MACs because they are rewritten on ingress/egress. Inter-container traffic is bridged inside the container host.	MAC is rewritten to host MAC, IP remains the same.	win-bridge, Azure-CNI, Flannel host-gateway uses win-bridge	win-bridge uses L2bridge network mode, connects containers to the underlay of hosts, offering best performance. Requires user-defined routes (UDR) for inter-node connectivity.

Network Driver	Description	Container Packet Modifications	Network Plugins	Network Plugin Characteristics
L2Tunnel	This is a special case of l2bridge, but only used on Azure. All packets are sent to the virtualization host where SDN policy is applied.	MAC rewritten, IP visible on the underlay network	Azure-CNI	Azure-CNI allows integration of containers with Azure vNET, and allows them to leverage the set of capabilities that Azure Virtual Network provides. For example, securely connect to Azure services or use Azure NSGs. See azure-cni for some examples
Overlay (Overlay networking for Windows in Kubernetes is in *alpha* stage)	Containers are given a vNIC connected to an external vSwitch. Each overlay network gets its own IP subnet, defined by a custom IP prefix.The overlay network driver uses VXLAN encapsulation.	Encapsulated with an outer header.	Win-overlay, Flannel VXLAN (uses win-overlay)	win-overlay should be used when virtual container networks are desired to be isolated from underlay of hosts (e.g. for security reasons). Allows for IPs to be re-used for different overlay networks (which have different VNID tags) if you are restricted on IPs in your datacenter. This option requires KB4489899 on Windows Server 2019.

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

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

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

- Pod -> Pod (IP)
- Pod -> Pod (Name)

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

The following IPAM options are supported on Windows:

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

Limitations

Control Plane

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

Compute

Resource management and process isolation

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

Operating System Restrictions

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

Feature Restrictions

- TerminationGracePeriod: not implemented
- Single file mapping: to be implemented with CRI-ContainerD
- Termination message: to be implemented with CRI-ContainerD

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

Memory Reservations and Handling

Windows does not have an out-of-memory process killer as Linux does. Windows always treats all user-mode memory allocations as virtual, and pagefiles are mandatory. The net effect is that Windows won't reach out of memory conditions the same way Linux does, and processes page to disk instead of being subject to out of memory (OOM) termination. If memory is over-provisioned and all physical memory is exhausted, then paging can slow down performance.

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

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

The behavior of the flags behave differently as described below:

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

Storage

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

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

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

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

Networking

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

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

The following networking functionality is not supported on Windows nodes

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

These features were added in Kubernetes v1.15:

- `kubectl port-forward`

CNI Plugins

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

1. Node-pod connectivity isn't possible by design. It's only possible for local pods with Flannel PR 1096
2. We are restricted to using VNI 4096 and UDP port 4789. The VNI limitation is being worked on and will be overcome in a future release (open-source flannel changes). See the official Flannel VXLAN backend docs for more details on these parameters.

DNS

- ClusterFirstWithHostNet is not supported for DNS. Windows treats all names with a â€˜.' as a FQDN and skips PQDN resolution

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

Security

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

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

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

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

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

API

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

At a high level, these OS concepts are different:

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

setting a path or command line that's interpreted inside a container, `\` may be needed.
- Signals - Windows interactive apps handle termination differently, and can implement one or more of these:
  - A UI thread handles well-defined messages including WM_CLOSE
  - Console apps handle ctrl-c or ctrl-break using a Control Handler
  - Services register a Service Control Handler function that can accept SERVICE_CONTROL_STOP control codes

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

V1.Container

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

- V1.Container.terminationMessagePath - this has some limitations in that Windows doesn't support mapping single files. The default value is /dev/termination-log, which does work because it does not exist on Windows by default.

V1.Pod

- V1.Pod.hostIPC, v1.pod.hostpid - host namespace sharing is not possible on Windows
- V1.Pod.hostNetwork - There is no Windows OS support to share the host network
- V1.Pod.dnsPolicy - ClusterFirstWithHostNet - is not supported because Host Networking is not supported on Windows.
- V1.Pod.podSecurityContext - see V1.PodSecurityContext below
- V1.Pod.shareProcessNamespace - this is a beta feature, and depends on Linux namespaces which are not implemented on Windows. Windows cannot share process namespaces or the container's root filesystem. Only the network can be shared.
- V1.Pod.terminationGracePeriodSeconds - this is not fully implemented in Docker on Windows, see: reference. The behavior today is that the ENTRYPOINT process is sent CTRL_SHUTDOWN_EVENT, then Windows waits 5 seconds by default, and finally shuts down all processes using the normal Windows shutdown behavior. The 5 second default is actually in the Windows registry inside the container, so it can be overridden when the container is built.
- V1.Pod.volumeDevices - this is a beta feature, and is not implemented on Windows. Windows cannot attach raw block devices to pods.
- V1.Pod.volumes - EmptyDir, Secret, ConfigMap, HostPath - all work and have tests in TestGrid
    - V1.emptyDirVolumeSource - the Node default medium is disk on Windows. Memory is not supported, as Windows does not have a built-in RAM disk.
- V1.VolumeMount.mountPropagation - mount propagation is not supported on Windows.

V1.PodSecurityContext

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

- V1.PodSecurityContext.SELinuxOptions - SELinux is not available on Windows
- V1.PodSecurityContext.RunAsUser - provides a UID, not available on Windows
- V1.PodSecurityContext.RunAsGroup - provides a GID, not available on Windows
- V1.PodSecurityContext.RunAsNonRoot - Windows does not have a root user. The closest equivalent is ContainerAdministrator which is an identity that doesn't exist on the node.
- V1.PodSecurityContext.SupplementalGroups - provides GID, not available on Windows

- V1.PodSecurityContext.Sysctls - these are part of the Linux sysctl interface. There's no equivalent on Windows.

Getting Help and Troubleshooting

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

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

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

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

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

   1. As native Windows Services

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

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

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

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

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

      # Query the service status
      ```

```
Get-Service kubelet
Get-Service kube-proxy
```

2. Using nssm.exe

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

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

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

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

```
# Register flanneld.exe
nssm install flanneld C:\flannel\flanneld.exe
nssm set flanneld AppParameters --kubeconfig-file=c:\k\c
onfig --iface=<ManagementIP> --ip-masq=1 --kube-subnet-
mgr=1
nssm set flanneld AppEnvironmentExtra
NODE_NAME=<hostname>
nssm set flanneld AppDirectory C:\flannel
nssm start flanneld

# Register kubelet.exe
# Microsoft releases the pause infrastructure container
at mcr.microsoft.com/k8s/core/pause:1.2.0
nssm install kubelet C:\k\kubelet.exe
nssm set kubelet AppParameters --hostname-
override=<hostname> --v=6 --pod-infra-container-
image=mcr.microsoft.com/k8s/core/pause:1.2.0 --resolv-
conf="" --allow-privileged=true --enable-debugging-
handlers --cluster-dns=<DNS-service-IP> --cluster-
domain=cluster.local --kubeconfig=c:\k\config --hairpin-
```

```
mode=promiscuous-bridge --image-pull-progress-
deadline=20m --cgroups-per-qos=false  --log-dir=<log
directory> --logtostderr=false --enforce-node-
allocatable="" --network-plugin=cni --cni-bin-dir=c:\k\c
ni --cni-conf-dir=c:\k\cni\config
nssm set kubelet AppDirectory C:\k
nssm start kubelet

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

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

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

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

For additional details, see official nssm usage docs.

3. My Windows Pods do not have network connectivity

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

4. My Windows Pods cannot ping external resources

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

   If you are still facing problems, most likely your network configuration in cni.conf deserves some extra attention. You can always edit this

static file. The configuration update will apply to any newly created Kubernetes resources.

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

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

5. My Windows node cannot access NodePort service

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

6. vNICs and HNS endpoints of containers are being deleted

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

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

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

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

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

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

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

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

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

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

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

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

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

11. No network adapter is found when starting kubelet

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

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

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

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

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

    Starting with Kubernetes v1.14, Microsoft releases the pause infrastructure container at `mcr.microsoft.com/k8s/core/pause:1.2.0`.

13. DNS resolution is not properly working

    Check the DNS limitations for Windows in this section.

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

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

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

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

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

16. What is a `pause` container?

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

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

Further investigation

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

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

Reporting Issues and Feature Requests

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

great avenue to get some initial support and troubleshooting ideas prior to creating a ticket.

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

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

What's next

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

Hyper-V isolation

Hyper-V isolation is requried to enable the following use cases for Windows containers in Kubernetes:

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

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

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

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

Deployment with kubeadm and cluster API

Kubeadm is becoming the de facto standard for users to deploy a Kubernetes cluster. Windows node support in kubeadm is currently a work-in-progress but a guide is available here. We are also making investments in cluster API to ensure Windows nodes are properly provisioned.

A few other key features

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

Feedback

Was this page helpful?

Yes No

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

Guide for 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.

Observability

Capturing logs from workloads

Logs are an important element of observability; they enable users to gain insights into the operational aspect of workloads and are a key ingredient to troubleshooting issues. Because Windows containers and workloads inside Windows containers behave differently from Linux containers, users had a hard time collecting logs, limiting operational visibility. Windows workloads for example are usually configured to log to ETW (Event Tracing for Windows) or push entries to the application event log. LogMonitor, an open source tool by Microsoft, is the recommended way to monitor configured log sources inside a Windows container. LogMonitor supports monitoring event logs, ETW providers, and custom application logs, piping them to STDOUT for consumption by `kubectl logs <pod>`.

Follow the instructions in the LogMonitor GitHub page to copy its binaries and configuration files to all your containers and add the necessary entrypoints for LogMonitor to push your logs to STDOUT.

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

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 March 26, 2020 at 12:30 AM PST by Official 1.18 Release Docs (#19116) (Page History)

Edit This Page

Running in multiple zones

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

Introduction

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

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

Functionality

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

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

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

Limitations

There are some important limitations of the multizone support:

- We assume that the different zones are located close to each other in the network, so we don't perform any zone-aware routing. In particular, traffic that goes via services might cross zones (even if some pods backing that service exist in the same zone as the client), and this may incur additional latency and cost.

- Volume zone-affinity will only work with a `PersistentVolume`, and will not work if you directly specify an EBS volume in the pod spec (for example).

- Clusters cannot span clouds or regions (this functionality will require full federation support).

- Although your nodes are in multiple zones, kube-up currently builds a single master node by default. While services are highly available and can tolerate the loss of a zone, the control plane is located in a single zone. Users that want a highly available control plane should follow the high availability instructions.

Volume limitations

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

- StatefulSet volume zone spreading when using dynamic provisioning is currently not compatible with pod affinity or anti-affinity policies.

- If the name of the StatefulSet contains dashes ("-"), volume zone spreading may not provide a uniform distribution of storage across zones.

- When specifying multiple PVCs in a Deployment or Pod spec, the StorageClass needs to be configured for a specific single zone, or the PVs need to be statically provisioned in a specific zone. Another workaround is to use a StatefulSet, which will ensure that all the volumes for a replica are provisioned in the same zone.

Walkthrough

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

Bringing up your cluster

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

GCE:

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

AWS:

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

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

Nodes are labeled

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

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

The output is similar to this:

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

Add more nodes in a second zone

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

GCE:

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

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

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

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

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

The output is similar to this:

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

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

Volume affinity

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

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

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

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

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

The output is similar to this:

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

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

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

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

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

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

And check node labels:

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

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

Pods are spread across zones

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

GCE:

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

AWS:

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

Verify that you now have nodes in 3 zones:

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

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

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

The pods should be spread across all 3 zones:

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

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

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

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

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

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

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

The output is similar to this:

```
LoadBalancer Ingress:   130.211.126.21
```

Set the above IP:

```
export IP=130.211.126.21
```

Explore with curl via IP:

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

The output is similar to this:

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

Again, explore multiple times:

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

The output is similar to this:

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

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

Shutting down the cluster

When you're done, clean up:

GCE:

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

AWS:

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

Feedback

Was this page helpful?

Yes No

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

Building large clusters

Support

At v1.18, 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.

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

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.

Analytics

Create an Issue Edit This Page
Page last modified on January 15, 2020 at 3:11 AM PST by fix-up 404 urls (#18008) (Page History)

Edit This Page

PKI certificates and requirements

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

- How certificates are used by your cluster
- Where certificates are stored

- Configure certificates manually
- Configure certificates for user accounts

How certificates are used by your cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

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

etcd also implements mutual TLS to authenticate clients and peers.

Where certificates are stored

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

Configure certificates manually

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

Single root CA

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

Required CAs:

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

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`, `kubern etes.default`, `kubernetes.default.svc`, `kubernetes.default.svc.cluste r`, `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:
>
> - The scenario where you are copying to your cluster CA certificates without private keys is referred as external CA in the kubeadm documentation.

- If you are comparing the above list with a kubeadm generated PKI, please be aware that `kube-etcd`, `kube-etcd-peer` and `kube-etcd-healthcheck-client` certificates are not generated in case of external etcd.

Certificate paths

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

Default CN	recommended key path	recommended cert path	command	key argument	cert argument
etcd-ca	etcd/ca.key	etcd/ca.crt	kube-apiserver		-etcd-cafile
kube-apiserver-etcd-client	apiserver-etcd-client.key	apiserver-etcd-client.crt	kube-apiserver	-etcd-keyfile	-etcd-certfile
kubernetes-ca	ca.key	ca.crt	kube-apiserver		-client-ca-file
kubernetes-ca	ca.key	ca.crt	kube-controller-manager	-cluster-signing-key-file	-client-ca-file, -root-ca-file, -cluster-signing-cert-file
kube-apiserver	apiserver.key	apiserver.crt	kube-apiserver	-tls-private-key-file	-tls-cert-file
kube-apiserver-kubelet-client	apiserver-kubelet-client.key	apiserver-kubelet-client.crt	kube-apiserver	-kubelet-client-key	-kubelet-client-certificate
front-proxy-ca	front-proxy-ca.key	front-proxy-ca.crt	kube-apiserver		-requestheader-client-ca-file
front-proxy-ca	front-proxy-ca.key	front-proxy-ca.crt	kube-controller-manager		-requestheader-client-ca-file
front-proxy-client	front-proxy-client.key	front-proxy-client.crt	kube-apiserver	-proxy-client-key-file	-proxy-client-cert-file
etcd-ca	etcd/ca.key	etcd/ca.crt	etcd		-trusted-ca-file, -peer-trusted-ca-file
kube-etcd	etcd/server.key	etcd/server.crt	etcd	-key-file	-cert-file
kube-etcd-peer	etcd/peer.key	etcd/peer.crt	etcd	-peer-key-file	-peer-cert-file
etcd-ca		etcd/ca.crt	etcdctl		-cacert

Default CN	recommended key path	recommended cert path	command	key argument	cert argument
kube-etcd-healthcheck-client	etcd/ healthcheck-client.key	etcd/ healthcheck-client.crt	etcdctl	-key	-cert

Same considerations apply for the service account key pair:

private key path	public key path	command	argument
sa.key		kube-controller-manager	service-account-private
	sa.pub	kube-apiserver	service-account-key

Configure certificates for user accounts

You must manually configure these administrator account and service accounts:

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

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

1. For each config, generate an x509 cert/key pair with the given CN and O.

2. Run `kubectl` as follows for each config:

```
KUBECONFIG=<filename> kubectl config set-cluster default-cluster --server=https://<host ip>:6443 --certificate-authority <path-to-kubernetes-ca> --embed-certs
KUBECONFIG=<filename> kubectl config set-credentials <credential-name> --client-key <path-to-key>.pem --client-certificate <path-to-cert>.pem --embed-certs
KUBECONFIG=<filename> kubectl config set-context default-system --cluster default-cluster --user <credential-name>
KUBECONFIG=<filename> kubectl config use-context default-system
```

These files are used as follows:

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

Feedback

Was this page helpful?

Yes No

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