Edit This Page

Setup

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

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

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

Local-machine Solutions

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

You should pick a local solution if you want to:

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

Pick a local-machine solution.

Hosted Solutions

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

You should pick a hosted solution if you:

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

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

Pick a hosted solution.

Turnkey – Cloud Solutions

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

You should pick a turnkey cloud solution if you:

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

Pick a turnkey cloud solution

Turnkey – On-Premises Solutions

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

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

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

Pick an on-prem turnkey cloud solution.

Custom Solutions

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

Pick a custom solution.

What's next

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

Feedback

Was this page helpful?

Yes

No

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

Edit This Page

Page last modified on October 04, 2018 at 3:37 AM PST by Trivial typo fixes (#10386) (Page History)

Edit This Page

Picking the Right Solution

Kubernetes can run on various platforms: from your laptop, to VMs on a cloud provider, to a rack of bare metal servers. The effort required to set up a cluster varies from running a single command to crafting your own customized cluster. Use this guide to choose a solution that fits your needs.

If you just want to "kick the tires" on Kubernetes, use the local Docker-based solutions.

When you are ready to scale up to more machines and higher availability, a hosted solution is the easiest to create and maintain.

Turnkey cloud solutions require only a few commands to create and cover a wide range of cloud providers. On-Premises turnkey cloud solutions have the simplicity of the turnkey cloud solution combined with the security of your own private network.

If you already have a way to configure hosting resources, use kubeadm to easily bring up a cluster with a single command per machine.

Custom solutions vary from step-by-step instructions to general advice for setting up a Kubernetes cluster from scratch.

- Local-machine Solutions
- Hosted Solutions
- Turnkey Cloud Solutions
- On-Premises turnkey cloud solutions

- Custom Solutions
- Table of Solutions

Local-machine Solutions

Community Supported Tools

- Minikube is a method for creating a local, single-node Kubernetes cluster for development and testing. Setup is completely automated and doesn't require a cloud provider account.
- Kubeadm-dind is a multi-node (while minikube is single-node) Kubernetes cluster which only requires a docker daemon. It uses docker-in-docker technique to spawn the Kubernetes cluster.
- Kubernetes IN Docker is a tool for running local Kubernetes clusters using Docker container "nodes". It is primarily designed for testing Kubernetes 1.11+. You can use it to create multi-node or multi-control-plane Kubernetes clusters.

Ecosystem Tools

- CDK on LXD supports a nine-instance deployment on localhost with LXD containers.
- Docker Desktop is an easy-to-install application for your Mac or Windows environment that enables you to start coding and deploying in containers in minutes on a single-node Kubernetes cluster.
- Minishift installs the community version of the Kubernetes enterprise platform OpenShift for local development & testing. It offers an all-in-one VM (`minishift start`) for Windows, macOS, and Linux. The container start is based on `oc cluster up` (Linux only). You can also install the included add-ons.
- MicroK8s provides a single command installation of the latest Kubernetes release on a local machine for development and testing. Setup is quick, fast (~30 sec) and supports many plugins including Istio with a single command.
- IBM Cloud Private-CE (Community Edition) can use VirtualBox on your machine to deploy Kubernetes to one or more VMs for development and test scenarios. Scales to full multi-node cluster.
- IBM Cloud Private-CE (Community Edition) on Linux Containers is a Terraform/Packer/BASH based Infrastructure as Code (IaC) scripts to create a seven node (1 Boot, 1 Master, 1 Management, 1 Proxy and 3 Workers) LXD cluster on Linux Host.

- k3s is a lightweight production-grade Kubernetes distribution. With a super-simple installation process and a binary footprint around 40MB, it is ideal for local-machine development.
- Ubuntu on LXD supports a nine-instance deployment on localhost.

Hosted Solutions

- AppsCode.com provides managed Kubernetes clusters for various public clouds, including AWS and Google Cloud Platform.
- APPUiO runs an OpenShift public cloud platform, supporting any Kubernetes workload. Additionally APPUiO offers Private Managed OpenShift Clusters, running on any public or private cloud.
- Amazon Elastic Container Service for Kubernetes offers managed Kubernetes service.
- Azure Kubernetes Service offers managed Kubernetes clusters.
- Containership Kubernetes Engine (CKE) intuitive Kubernetes cluster provisioning and management on GCP, Azure, AWS, Packet, and DigitalOcean. Seamless version upgrades, autoscaling, metrics, workload creation, and more.
- DigitalOcean Kubernetes offers managed Kubernetes service.
- Giant Swarm offers managed Kubernetes clusters in their own datacenter, on-premises, or on public clouds.
- Google Kubernetes Engine offers managed Kubernetes clusters.
- IBM Cloud Kubernetes Service offers managed Kubernetes clusters with isolation choice, operational tools, integrated security insight into images and containers, and integration with Watson, IoT, and data.
- Kubermatic provides managed Kubernetes clusters for various public clouds, including AWS and Digital Ocean, as well as on-premises with OpenStack integration.
- Kublr offers enterprise-grade secure, scalable, highly reliable Kubernetes clusters on AWS, Azure, GCP, and on-premise. It includes out-of-the-box backup and disaster recovery, multi-cluster centralized logging and monitoring, and built-in alerting.
- KubeSail is an easy, free way to try Kubernetes.
- Madcore.Ai is devops-focused CLI tool for deploying Kubernetes infrastructure in AWS. Master, auto-scaling group nodes with spot-instances, ingress-ssl-lego, Heapster, and Grafana.

- Nutanix Karbon is a multi-cluster, highly available Kubernetes management and operational platform that simplifies the provisioning, operations, and lifecycle management of Kubernetes.
- OpenShift Dedicated offers managed Kubernetes clusters powered by OpenShift.
- OpenShift Online provides free hosted access for Kubernetes applications.
- Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE) is a fully-managed, scalable, and highly available service that you can use to deploy your containerized applications to the cloud.
- Platform9 offers managed Kubernetes service that works on-premises or on any public cloud, with 99.9% SLA guarantee.
- Stackpoint.io provides Kubernetes infrastructure automation and management for multiple public clouds.
- SysEleven MetaKube offers managed Kubernetes as a service powered on our OpenStack public cloud. It includes lifecycle management, administration dashboards, monitoring, autoscaling and much more.
- VEXXHOST VEXXHOST proudly offers Certified Kubernetes on their public cloud, which also happens to be the largest OpenStack public cloud in Canada.
- VMware Cloud PKS is an enterprise Kubernetes-as-a-Service offering in the VMware Cloud Services portfolio that provides easy to use, secure by default, cost effective, SaaS-based Kubernetes clusters.

Turnkey Cloud Solutions

These solutions allow you to create Kubernetes clusters on a range of Cloud IaaS providers with only a few commands. These solutions are actively developed and have active community support.

- Agile Stacks
- Alibaba Cloud
- APPUiO
- AWS
- Azure
- CenturyLink Cloud
- Conjure-up Kubernetes with Ubuntu on AWS, Azure, Google Cloud, Oracle Cloud
- Containership
- Docker Enterprise
- Gardener
- Giant Swarm

- Google Compute Engine (GCE)
- IBM Cloud
- k3s
- Kontena Pharos
- Kubermatic
- Kublr
- Madcore.Ai
- Nirmata
- Nutanix Karbon
- Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE)
- Pivotal Container Service
- Platform9 Managed Kubernetes as a Service
- Rancher
- Stackpoint.io
- Supergiant.io
- VEXXHOST
- VMware Cloud PKS
- VMware Enterprise PKS

On-Premises turnkey cloud solutions

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

- Agile Stacks
- APPUiO
- Docker Enterprise
- Giant Swarm
- GKE On-Prem | Google Cloud
- IBM Cloud Private
- k3s
- Kontena Pharos
- Kubermatic
- Kublr
- Mirantis Cloud Platform
- Nirmata
- OpenShift Container Platform (OCP) by Red Hat
- Pivotal Container Service
- Platform9 Managed Kubernetes as a Service
- Rancher
- SUSE CaaS Platform
- SUSE Cloud Application Platform
- VMware Enterprise PKS

Custom Solutions

Kubernetes can run on a wide range of Cloud providers and bare-metal environments, and with many base operating systems.

If you can find a guide below that matches your needs, use it.

Universal

If you already have a way to configure hosting resources, use kubeadm to bring up a cluster with a single command per machine.

Cloud

These solutions are combinations of cloud providers and operating systems not covered by the above solutions.

- Cloud Foundry Container Runtime (CFCR)
- Gardener
- Kublr
- Kubernetes on Ubuntu
- Kubespray
- Rancher Kubernetes Engine (RKE)
- VMware Essential PKS

On-Premises VMs

- Cloud Foundry Container Runtime (CFCR)
- CloudStack (uses Ansible)
- Fedora (Multi Node) (uses Fedora and flannel)
- Nutanix AHV
- OpenShift Container Platform (OCP) Kubernetes platform by Red Hat
- oVirt
- Platform9 Managed Kubernetes as a Service works on any infrastructure: on-premises, bare metal, or private/hybrid cloud.
- VMware Essential PKS
- VMware vSphere
- VMware vSphere, OpenStack, or Bare Metal (uses Juju, Ubuntu and flannel)

Bare Metal

- Digital Rebar
- Docker Enterprise

- Fedora (Single Node)
- Fedora (Multi Node)
- k3s
- Kubernetes on Ubuntu
- Kubernetes on Ubuntu
- OpenShift Container Platform (OCP) Kubernetes platform by Red Hat
- Platform9 Managed Kubernetes as a Service works on any infrastructure: on-premises, bare metal, or private/hybrid cloud.
- VMware Essential PKS

Integrations

These solutions provide integration with third-party schedulers, resource managers, and/or lower level platforms.

- DCOS
  - Community Edition DCOS uses AWS
  - Enterprise Edition DCOS supports cloud hosting, on-premises VMs, and bare metal

Table of Solutions

Below is a table of all of the solutions listed above.

IaaS Provider	Config. Mgmt.	OS
Agile Stacks	Terraform	CoreOS
Alibaba Cloud Container Service For Kubernetes	ROS	CentOS
any	any	multi-su
any	any	any
any	any	any
any	RKE	multi-su
any	Gardener Cluster-Operator	multi-su
AppsCode.com	Saltstack	Debian
AWS	CoreOS	CoreOS
AWS	Saltstack	Debian
AWS	kops	Debian
AWS	Juju	Ubuntu
Azure	Juju	Ubuntu
Azure (IaaS)		Ubuntu
Azure Kubernetes Service		Ubuntu
Bare-metal	custom	CentOS
Bare-metal	custom	Fedora
Bare-metal	custom	Fedora
Bare Metal	Juju	Ubuntu

IaaS Provider	Config. Mgmt.	OS
Bare-metal	custom	Ubuntu
CloudStack	Ansible	CoreOS
DCOS	Marathon	CoreOS
Digital Rebar	kubeadm	any
Docker Enterprise	custom	multi-su
Giant Swarm		CoreOS
GCE	CoreOS	CoreOS
GCE	Juju	Ubuntu
GCE	Saltstack	Debian
Google Kubernetes Engine		
IBM Cloud Kubernetes Service		Ubuntu
IBM Cloud Kubernetes Service		Ubuntu
IBM Cloud Private	Ansible	multi-su
Kublr	custom	multi-su
Kubermatic		multi-su
KVM	custom	Fedora
libvirt	custom	Fedora
lxd	Juju	Ubuntu
Madcore.Ai	Jenkins DSL	Ubuntu
Mirantis Cloud Platform	Salt	Ubuntu
Oracle Cloud Infrastructure	Juju	Ubuntu
Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE)		
oVirt		
Platform9		multi-su
Rackspace	custom	CoreOS
Red Hat OpenShift	Ansible & CoreOS	RHEL &
Stackpoint.io		multi-su
Vagrant	CoreOS	CoreOS
VMware vSphere	any	multi-su
VMware vSphere	Juju	Ubuntu
VMware Cloud PKS		Photon
VMware Enterprise PKS	BOSH	Ubuntu
VMware Essential PKS	any	multi-su

Definition of columns

- **IaaS Provider** is the product or organization which provides the virtual or physical machines (nodes) that Kubernetes runs on.
- **OS** is the base operating system of the nodes.
- **Config. Mgmt.** is the configuration management system that helps install and maintain Kubernetes on the nodes.
- **Networking** is what implements the networking model. Those with networking type *none* may not support more than a single node, or may

support multiple VM nodes in a single physical node.

- **Support Levels**
  - **Project**: Kubernetes committers regularly use this configuration, so it usually works with the latest release of Kubernetes.
  - **Commercial**: A commercial offering with its own support arrangements.
  - **Community**: Actively supported by community contributions. May not work with recent releases of Kubernetes.
  - **Inactive**: Not actively maintained. Not recommended for first-time Kubernetes users, and may be removed.
- **Notes** has other relevant information, such as the version of Kubernetes used.

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 April 22, 2019 at 1:53 PM PST by fix(pick-right-solution.md): Remove conflicts and correct possible Grammar error (#13930) (Page History)

Edit This Page

Building a release

You can either build a release from source or download a pre-built release. If you do not plan on developing Kubernetes itself, we suggest using a pre-built version of the current release, which can be found in the Release Notes.

The Kubernetes source code can be downloaded from the kubernetes/kubernetes repo.

- Building from source

Building from source

If you are simply building a release from source there is no need to set up a full golang environment as all building happens in a Docker container.

Building a release is simple.

```
git clone https://github.com/kubernetes/kubernetes.git
cd kubernetes
make release
```

For more details on the release process see the kubernetes/kubernetes `build` directory.

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 08, 2019 at 5:04 AM PST by Update building-from-source.md (#12946) (Page History)

Edit This Page

Building a release

You can either build a release from source or download a pre-built release. If you do not plan on developing Kubernetes itself, we suggest using a pre-built version of the current release, which can be found in the Release Notes.

The Kubernetes source code can be downloaded from the kubernetes/kubernetes repo.

- Building from source

Building from source

If you are simply building a release from source there is no need to set up a full golang environment as all building happens in a Docker container.

Building a release is simple.

```
git clone https://github.com/kubernetes/kubernetes.git
cd kubernetes
make release
```

For more details on the release process see the kubernetes/kubernetes `build` directory.

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 08, 2019 at 5:04 AM PST by Update building-from-source.md (#12946) (Page History)

Edit This Page

v1.14 Release Notes

- v1.14.0
  - Downloads for v1.14.0
    - Client Binaries
    - Server Binaries
    - Node Binaries
- Kubernetes v1.14 Release Notes
  - 1.14 What's New
  - Known Issues
  - Urgent Upgrade Notes
    - (No, really, you MUST read this before you upgrade)

  - Deprecations
  - Removed and deprecated metrics
    - Removed metrics
    - Deprecated metrics
  - Notable Features
  - API Changes
  - Detailed Bug Fixes And Changes
    - API Machinery
    - Apps
    - Auth
    - AWS
    - Azure
    - CLI
    - Cloud Provider
    - Cluster Lifecycle
    - GCP
    - Network
    - Node
    - Scheduling
    - Storage
    - Testing
    - VMware
    - Windows
  - External Dependencies
- v1.14.0-rc.1
  - Downloads for v1.14.0-rc.1
    - Client Binaries
    - Server Binaries
    - Node Binaries
  - Changelog since v1.14.0-beta.2
    - Action Required
    - Other notable changes
- v1.14.0-beta.2
  - Downloads for v1.14.0-beta.2
    - Client Binaries
    - Server Binaries
    - Node Binaries
  - Changelog since v1.14.0-beta.1
    - Action Required
    - Other notable changes
- v1.14.0-beta.1
  - Downloads for v1.14.0-beta.1
    - Client Binaries
    - Server Binaries
    - Node Binaries
  - Changelog since v1.14.0-alpha.3

        * Action Required
        * Other notable changes
- v1.14.0-alpha.3
  - Downloads for v1.14.0-alpha.3
    * Client Binaries
    * Server Binaries
    * Node Binaries
  - Changelog since v1.14.0-alpha.2
    * Action Required
    * Other notable changes
- v1.14.0-alpha.2
  - Downloads for v1.14.0-alpha.2
    * Client Binaries
    * Server Binaries
    * Node Binaries
  - Changelog since v1.14.0-alpha.1
    * Action Required
    * Other notable changes
- v1.14.0-alpha.1
  - Downloads for v1.14.0-alpha.1
    * Client Binaries
    * Server Binaries
    * Node Binaries
  - Changelog since v1.13.0
    * Action Required
    * Other notable changes

v1.14.0

Documentation

Downloads for v1.14.0

filename	sha512 hash
kubernetes.tar.gz	`0ad264a46f185a9ff4db0393508a9598dab146f438b2cfdc7527592eb422870b8f26ade7`
kubernetes-src.tar.gz	`c5175439decc1c5f54254572bfec3c9f61f39d6bd1cbc28d1f771f8f931b98f0c305f187`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	`68bdba50a2b0be755e73e34ffc758fd419940adace096b1ddebd44a0`

filename	sha512 hash
kubernetes-client-darwin-amd64.tar.gz	255bd93082b3ac5d69bd4e45c75c9f19efee50ad6add50837ff2987c
kubernetes-client-linux-386.tar.gz	2bd115ad2503fdfe5482e4592fcc0c8a2aee36be5205220a13c8050c
kubernetes-client-linux-amd64.tar.gz	a551adf8019b17fce5aff2b379fab3627588978a2d628b64ba1af6f3
kubernetes-client-linux-arm.tar.gz	24e771cd4074786330e07f5537259a28d0932102639326230d9161f1
kubernetes-client-linux-arm64.tar.gz	16204f2345ab3523bbe3c868f04806a97c111d940b2594aaff67cf73
kubernetes-client-linux-ppc64le.tar.gz	cd9ce829d585dd3331c53d35015d4017026d5efd24b9bc2f34299524
kubernetes-client-linux-s390x.tar.gz	482c0a8e53b27f8922f58d89fb81842ddd9c3ffd120e635838992dc9
kubernetes-client-windows-386.tar.gz	4446d666f999e979a7245e1b7ebf4817f7bd23aa247a38853a63b9cd
kubernetes-client-windows-amd64.tar.gz	97f4789f21d10fd3df446e55bc489472dcd534c623bb40dc3cb20fe1

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	25739802a641517a8bbb933b69000a943e8dd38e616b8778149dd01387
kubernetes-server-linux-arm.tar.gz	c1dbba77a4ff5661eb36c55182a753b88ccc9b89ca31e162b066721267
kubernetes-server-linux-arm64.tar.gz	ad346bbe2a053c1106b51e5125698737dc7b76fa3bf439e14d4b4ba1c2
kubernetes-server-linux-ppc64le.tar.gz	49f9bd1c751620ecf4b5c152f287d72b36abca21fd1dfe99443d984473
kubernetes-server-linux-s390x.tar.gz	d6be847f2a0358755a69dea26181e5fc1a80ac4939b8b04a3875e1f669

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	75dc99919d1084d7d471a53ab60c743dc399145c99e83f37c6ba3c241
kubernetes-node-linux-arm.tar.gz	49013a4f01be8086fff332099d94903082688b9b295d2f3446846265
kubernetes-node-linux-arm64.tar.gz	f8c0cb0c089cd1d1977c049002620b8cf748d193c1b76dd1d3aac01f
kubernetes-node-linux-ppc64le.tar.gz	48fc02c856a192388877189a43eb1cda531e548bb035f9dfe6a1e3c8
kubernetes-node-linux-s390x.tar.gz	d7c5f52cf602fd0c0d0f72d4cfe1ceaa4bad70a42f37f21c103f17c34
kubernetes-node-windows-amd64.tar.gz	120afdebe844b06a7437bb9788c3e7ea4fc6352aa18cc6a00e70f44f5

Kubernetes v1.14 Release Notes

1.14 What's New

Support for Windows Nodes is Graduating to Stable (#116)

- Support for Windows Server 2019 for worker nodes and containers
- Support for out of tree networking with Azure-CNI, OVN-Kubernetes and Flannel

- Improved support for pods, service types, workload controllers and metrics/quotas to closely match the capabilities offered for Linux containers kubernetes/enhancements: #116 [kep]

Updated Plugin Mechanism for kubectl is Graduating to Stable (#579)

- Extends functionality to kubectl to support extensions adding new commands as well as overriding specific subcommands (at any depth).
- Documentation fixes kubernetes/enhancements: #579 [kep]

Durable Local Storage Management is Now GA (#121)

- Makes locally attached (non-network attached) storage available as a persistent volume source.
- Allows users to take advantage of the typically cheaper and improved performance of persistent local storage kubernetes/kubernetes: #73525, #74391, #74769 kubernetes/enhancements: #121 [kep]

Pid Limiting is Graduating to Beta (#757)

- Prevents a pod from starving pid resource
- Ability to isolate pid resources pod-to-pod and node-to-pod kubernetes/kubernetes: #73651 kubernetes/enhancements: #757 [kep]

Pod Priority and Preemption in Kubernetes (#564)

- Pod priority and preemption enables Kubernetes scheduler to schedule more important Pods first and when cluster is out of resources, it removes less important pods to create room for more important ones. The importance is specified by priority. kubernetes/kubernetes: #73498, #73555, #74465 kubernetes/enhancements: #564 [kep]

Pod Ready++ (#580)

- Introduces extension point for external feedback on pod readiness. kubernetes/kubernetes: #74434, kubernetes/enhancements: #580 [kep]

Kubeadm: Automate certificate copy between control planes in HA setups

- Joining control plane nodes to a HA cluster can now be simplified by enabling the optional automatic copy of certificates from an existing control plane node.
- You can now use `kubeadm init --experimental-upload-certs` and `kubeadm join --experimental-control-plane --certificate-key`. kubernetes/kubeadm: #1373 kubernetes/enhancements: #357 [kep]

Kubeadm: Expose the `kubeadm join` workflow as phases

- The `kubeadm join` command can now be used in phases. Similar to the work that was done for `kubeadm init` in 1.13, in 1.14 the `join` phases can be now executed step-by-step/selectively using the `kubeadm join phase` sub-command. This makes it possible to further customize the

workflow of joining nodes to the cluster. kubernetes/kubeadm: #1204 kubernetes/enhancements: kep

Known Issues

- There is a known issue coredns/coredns#2629 in CoreDNS 1.3.1, wherein if the Kubernetes API shuts down while CoreDNS is connected, CoreDNS will crash. The issue is fixed in CoreDNS 1.4.0 in coredns/coredns#2529.
- Kubelet might fail to restart if an existing flexvolume mounted pvc contains a large number of directories, or is full. #75019

Urgent Upgrade Notes

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

- kube-apiserver:
  - Default RBAC policy no longer grants access to discovery and permission-checking APIs (used by `kubectl auth can-i`) to *unauthenticated* users. Upgraded clusters preserve prior behavior, but cluster administrators wishing to grant unauthenticated users access in new clusters will need to explicitly opt-in to expose the discovery and/or permission-checking APIs:
  - `kubectl create clusterrolebinding anonymous-discovery --clusterrole=system:discovery --group=system:unauthenticated`
  - `kubectl create clusterrolebinding anonymous-access-review --clusterrole=system:basic-user --group=system:unauthenticated`
  - The deprecated –storage-versions flag has been removed. The storage versions will always be the default value built-in the kube-apiserver binary. (#67678, @caesarxuchao)
  - The deprecated `--repair-malformed-updates` flag has been removed (#73663, @danielqsj)
  - The `/swaggerapi/*` schema docs, deprecated since 1.7, have been removed in favor of the /openapi/v2 schema docs. (#72924, @liggitt)
  - The /swagger.json and /swagger-2.0.0.pb-v1 schema documents, deprecated since v1.10, have been removed in favor of `/openapi/v2` (#73148, @liggitt)
  - `kube-apiserver` now only aggregates openapi schemas from `/openapi/v2` endpoints of aggregated API servers. The fallback to aggregate from `/swagger.json` has been removed. Ensure aggregated API servers provide schema information via `/openapi/v2` (available since v1.10). (#73441, @roycaihw)
  - The OpenAPI definitions with the prefix “io.k8s.kubernetes.pkg” (deprecated since 1.9) have been removed. (#74596, @sttts)

  - The `ValidateProxyRedirects` feature was promoted to Beta and enabled by default. This feature restricts redirect-following from the apiserver to same-host redirects. If nodes are configured to respond to CRI streaming requests on a different host interface than what the apiserver makes requests on (only the case if not using the built-in dockershim & setting the kubelet flag `--redirect-container-streaming=true`), then these requests will be broken. In that case, the feature can be temporarily disabled until the node configuration is corrected. We suggest setting `--redirect-container-streaming=false` on the kubelet to avoid issues.(#72552, @tallclair)

- kubectl
  - The deprecated `--show-all` flag to `kubectl get` has been removed (#69255, @Pingan2017)

- kubelet
  - The deprecated `--experimental-fail-swap-on` flag has been removed (#69552, @Pingan2017)
  - Health check (liveness & readiness) probes using an HTTPGetAction will no longer follow redirects to different hostnames from the original probe request. Instead, these non-local redirects will be treated as a Success (the documented behavior). In this case an event with reason "ProbeWarning" will be generated, indicating that the redirect was ignored. If you were previously relying on the redirect to run health checks against different endpoints, you will need to perform the healthcheck logic outside the Kubelet, for instance by proxying the external endpoint rather than redirecting to it. (#75416, @tallclair)

- client-go
  - The deprecated versionless API group accessors (like `clientset.Apps()`) have been removed. Use an explicit version instead (like `clientset.AppsV1()`) (#74422, @liggitt)
  - The disk-cached discovery client is moved from k8s.io/client-go/discovery to k8s.io/client-go/discovery/cached/disk. The memory-cached discovery client is moved from k8s.io/client-go/discovery/cached to k8s.io/client-go/discovery/cached/memory. (#72214, @caesarxuchao)

- kubeadm
  - `kubeadm alpha preflight` and `kubeadm alpha preflight node` are removed; you can now use `kubeadm join phase preflight` (#73718, @fabriziopandini)

- The deprecated taints `node.alpha.kubernetes.io/notReady` and

`node.alpha.kubernetes.io/unreachable` are no longer supported or adjusted. These uses should be replaced with `node.kubernetes.io/not-ready` and `node.kubernetes.io/unreachable` (#73001, @shivnagarajan)

- Any Prometheus queries that match `pod_name` and `container_name` labels (e.g. cadvisor or kubelet probe metrics) should be updated to use `pod` and `container` instead. `pod_name` and `container_name` labels will be present alongside `pod` and `container` labels for one transitional release and removed in the future. (#69099, @ehashman)

Deprecations

- kubectl
  - `kubectl convert` is deprecated and will be removed in v1.17.
  - The `--export` flag for the `kubectl get` command is deprecated and will be removed in v1.18. (#73787, @soltysh)
- kubelet
  - OS and Arch information is now recorded in `kubernetes.io/os` and `kubernetes.io/arch` labels on Node objects. The previous labels (`beta.kubernetes.io/os` and `beta.kubernetes.io/arch`) are still recorded, but are deprecated and targeted for removal in v1.18. (#73333, @yujuhong)
  - The `--containerized` flag is deprecated and will be removed in a future release (#74267, @dims)
- hyperkube
  - The `--make-symlinks` flag is deprecated and will be removed in a future release. (#74975, @dims)
- API
  - Ingress resources are now available via `networking.k8s.io/v1beta1`. Ingress resources in `extensions/v1beta1` are deprecated and will no longer be served in v1.18. Existing persisted data is available via the new API group/version (#74057, @liggitt)
  - NetworkPolicy resources will no longer be served from `extensions/v1beta1` in v1.16. Migrate use to the `networking.k8s.io/v1` API, available since v1.8. Existing persisted data can be retrieved via the `networking.k8s.io/v1` API.
  - PodSecurityPolicy resources will no longer be served from `extensions/v1beta1` in v1.16. Migrate to the `policy/v1beta1` API, available since v1.10. Existing persisted data can be retrieved via the `policy/v1beta1` API.
  - DaemonSet, Deployment, and ReplicaSet resources will no longer be served from `extensions/v1beta1`, `apps/v1beta1`, or `apps/v1beta2`

in v1.16. Migrate to the `apps/v1` API, available since v1.9. Existing persisted data can be retrieved via the `apps/v1` API.
  - PriorityClass resources have been promoted to `scheduling.k8s.io/v1` with no changes. The `scheduling.k8s.io/v1beta1` and `scheduling.k8s.io/v1alpha1` versions are now deprecated and will stop being served by default in v1.17. (#73555, #74465, @bsalamat)
  - The `export` query parameter for list API calls is deprecated and will be removed in v1.18 (#73783, @deads2k)
- The following features are now GA, and the associated feature gates are deprecated and will be removed in v1.15:
  - `CustomPodDNS`
  - `HugePages`
  - `MountPropagation`
  - `PersistentLocalVolumes`
- CoreDNS: The following directives or keywords are deprecated and will be removed in v1.15:
  - `upstream` option of `kubernetes` plugin, becoming default behavior in v1.15.
  - `proxy` plugin replaced by `forward` plugin

Removed and deprecated metrics

Removed metrics

- `reflector_items_per_list`
- `reflector_items_per_watch`
- `reflector_last_resource_version`
- `reflector_list_duration_seconds`
- `reflector_lists_total`
- `reflector_short_watches_total`
- `reflector_watch_duration_seconds`
- `reflector_watches_total`

Deprecated metrics

- `rest_client_request_latency_seconds` -> `rest_client_request_duration_seconds`
- `apiserver_proxy_tunnel_sync_latency_secs` -> `apiserver_proxy_tunnel_sync_duration_seconds`
- `scheduler_scheduling_latency_seconds` -> `scheduler_scheduling_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_secon
- 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_s
- 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
- docker_operations -> docker_operations_total
- docker_operations_latency_microseconds -> docker_operations_latency_seconds
- docker_operations_errors -> docker_operations_errors_total
- docker_operations_timeout -> docker_operations_timeout_total
- network_plugin_operations_latency_microseconds -> network_plugin_operations_latency_secc
- sync_proxy_rules_latency_microseconds -> sync_proxy_rules_latency_seconds
- apiserver_request_count -> apiserver_request_total
- apiserver_request_latencies -> apiserver_request_latency_seconds
- apiserver_request_latencies_summary -> apiserver_request_latency_seconds
- apiserver_dropped_requests -> apiserver_dropped_requests_total
- etcd_helper_cache_hit_count -> etcd_helper_cache_hit_total
- etcd_helper_cache_miss_count -> etcd_helper_cache_miss_total
- etcd_helper_cache_entry_count -> etcd_helper_cache_entry_total
- etcd_request_cache_get_latencies_summary -> etcd_request_cache_get_latency_seconds
- etcd_request_cache_add_latencies_summary -> etcd_request_cache_add_latency_seconds
- etcd_request_latencies_summary -> etcd_request_latency_seconds
- transformation_latencies_microseconds -> transformation_latencies_seconds
- data_key_generation_latencies_microseconds -> data_key_generation_latencies_seconds

Notable Features

- Increased the histogram resolution of the API server client certificate to accommodate short-lived (< 6h) client certificates. (#74806, @mxinden)
- Updated to use golang 1.12 (#74632, @cblecker)
- The `RunAsGroup` feature has been promoted to beta and enabled by default. `PodSpec` and `PodSecurityPolicy` objects can be used to control the primary GID of containers on supported container runtimes. (#73007, @krmayankk)
- Added the same information to an init container as a standard container in a pod when using `PodPresets`. (#71479, @soggiest)
- kube-conformance image will now run ginkgo with the `--dryRun` flag if the container is run with the environment variable E2E_DRYRUN set. (#74731, @johnSchnake)
- Introduced dynamic volume provisioning shim for CSI migration (#73653, @ddebroy)

- Applied resources from a directory containing kustomization.yaml (#74140, @Liujingfang1)
- kubeadm: Allowed to download certificate secrets uploaded by `init` or `upload-certs` phase, allowing to transfer certificate secrets (certificates and keys) from the cluster to other master machines when creating HA deployments. (#74168, @ereslibre)
- The `--quiet` option to `kubectl run` now suppresses resource deletion messages emitted when the `--rm` option is specified. (#73266, @awh)
- Added Custom Resource support to `kubectl autoscale` (#72678, @rmohr)
- Cinder volume limit can now be configured from node too (#74542, @gnufied)
- It is now possible to combine the `-f` and `-l` flags in `kubectl logs` (#67573, @m1kola)
- New conformance tests added for API Aggregation. (#63947, @jennybuckley)
- Moved fluentd-elasticsearch addon images to community controlled location (#73819, @coffeepac)
- Removed local etcd members from the etcd cluster when `kubeadm reset` (#74112, @pytimer)
- kubeadm will now not fail preflight checks when running on >= 5.0 Linux kernel (#74355, @brb)
- Scheduler cache snapshot algorithm has been optimized to improve scheduling throughput. (#74041, @bsalamat)
- It is now possible to upload certificates required to join a new control-plane to kubeadm-certs secret using the flag `--experimental-upload-certs` on `init` or upload-certs phase. (#73907, @yagonobre) @RobertKrawitz)
- `kubectl auth reconcile` now outputs details about what changes are being made (#71564, @liggitt)
- Added Kustomize as a subcommand in kubectl (#73033, @Liujingfang1)
- Added `kubelet_node_name` metrics. (#72910, @danielqsj)
- Updated AWS SDK to v1.16.26 for ECR PrivateLink support (#73435, @micahhausler)
- Expanded `kubectl wait` to work with more types of selectors. (#71746, @rctl) (#72832, @MrHohn)
- Added configuration for AWS endpoint fine control: (#72245, @ampsingram)
- The CoreDNS configuration now has the forward plugin for proxy in the default configuration instead of the proxy plugin. (#73267, @rajansandeep)
- Added alpha field storageVersionHash to the discovery document for each resource. Its value must be treated as opaque by clients. Only equality comparison on the value is valid. (#73191, @caesarxuchao)
- If you are running the cloud-controller-manager and you have the `pvlabel.kubernetes.io` alpha Initializer enabled, you must now enable PersistentVolume labeling using the `PersistentVolumeLabel` admission controller instead. You can do this by adding `PersistentVolumeLabel`

in the `--enable-admission-plugins` kube-apiserver flag. (#73102, @andrewsykim)
- kubectl supports copying files with wild card (#72641, @dixudx)
- kubeadm now attempts to detect an installed CRI by its usual domain socket, so that `--cri-socket` can be omitted from the command line if Docker is not used and there is a single CRI installed. (#69366, @rosti)
- `CSINodeInfo` and `CSIDriver` CRDs have been installed in the local cluster. (#72584, @xing-yang)
- Node OS/arch labels have been promoted to GA (#73048, @yujuhong)
- Added support for max attach limit for Cinder (#72980, @gnufied)
- Enabled mTLS encryption between etcd and kube-apiserver in GCE (#70144, @wenjiaswe)
- Added `ResourceVersion` as a precondition for delete in order to ensure a delete fails if an unobserved change happens to an object. (#74040, @ajatprabha)
- There is now support for collecting pod logs under `/var/log/pods/NAMESPACE_NAME_UID` to stackdriver with `k8s_pod` resource type. (#74502, @Random-Liu)
- Changed CRI pod log directory from `/var/log/pods/UID` to `/var/log/pods/NAMESPACE_NAME_UID`. (#74441, @Random-Liu)
- `RuntimeClass` has been promoted to beta, and is enabled by default. (#75003, @tallclair)
- New "dry_run" metric label (indicating the value of the dryRun query parameter) has been added into the metrics:
- apiserver_request_total
- apiserver_request_duration_seconds New "APPLY" value for the "verb" metric label which indicates a PATCH with "Content-Type: apply-patch+yaml". This value is experimental and will only be present if the ServerSideApply alpha feature is enabled. (#74997, @jennybuckley)
- GCE: bumped COS image version to `cos-beta-73-11647-64-0` (#75149, @yguo0905)
- Added alpha support for ephemeral CSI inline volumes that are embedded in pod specs. (#74086, @vladimirvivien)

API Changes

- [CRI] Added a new field called `runtime_handler` into `PodSandbox` and `PodSandboxStatus` to track the `RuntimeClass` information of a pod. (#73833, @haiyanmeng)

Detailed Bug Fixes And Changes

API Machinery

- client-go: `PortForwarder.GetPorts()` now contain correct local port if no local port was initially specified when setting up the port forwarder (#73676, @martin-helmich)
- Fixed an issue with missing `apiVersion/kind` in object data sent to admission webhooks (#74448, @liggitt)
- Prometheus metrics for `crd_autoregister`, `crd_finalizer` and `crd_naming_condition_controller` are exported. (#71767, @roycaihw)
- Fixed admission metrics in seconds. (#72343, @danielqsj)
- When a watch is closed by an HTTP2 load balancer and we are told to go away, skip printing the message to stderr by default.
- Spedup kubectl by >10 when calling out to kube-apiserver for discovery information. (#73345, @sttts)
- Fixed watch to not send the same set of events multiple times causing watcher to go back in time (#73845, @wojtek-t) (#73277, @smarterclayton)
- Fix kube-apiserver not to create default/kubernetes service endpoints before it reports readiness via the /healthz and therefore is ready to serve requests. Also early during startup old endpoints are remove which might be left over from a previously crashed kube-apiserver. (#74668, @sttts)
- Add a configuration field to shorten the timeout of validating/mutating admission webhook call. The timeout value must be between 1 and 30 seconds. Default to 30 seconds when unspecified. (#74562, @roycaihw)
- The apiserver, including both the kube-apiserver and apiservers built with the generic apiserver library, will now return 413 RequestEntityTooLarge error if a json patch contains more than 10,000 operations. (#74000, @caesarxuchao)
- Fixed an error processing watch events when running skewed apiservers (#73482, @liggitt)
- jsonpath expressions containing `[start:end:step]` slice are now evaluated correctly (#73149, @liggitt)
- `metadata.deletionTimestamp` is no longer moved into the future when issuing repeated DELETE requests against a resource containing a finalizer. (#73138, @liggitt)
- Fixed kube-apiserver not to create default/kubernetes service endpoints before it reports readiness via the /healthz and therefore is ready to serve requests. Also early during startup old endpoints are remove which might be left over from a previously crashed kube-apiserver. (#74668, @sttts)
- `watch.Until` now works for long durations. (#67350, @tnozicka)
- Added duration metric for CRD webhook converters. (#74376, @mbohlool)

- Fixed keymutex issues which may crash in some platforms. (#74348, @danielqsj)
- Considerably reduced the CPU load in kube-apiserver while aggregating OpenAPI specifications from aggregated API servers. (#71223, @sttts)
- Fixed graceful apiserver shutdown to not drop outgoing bytes before the process terminates. (#72970, @sttts)

Apps

- Added deleting pods created by `DaemonSet` assigned to not existing nodes. (#73401, @krzysztof-jastrzebski)
- Pod eviction now honors graceful deletion by default if no delete options are provided in the eviction request. (#72730, @liggitt)

Auth

- Added `kubectl auth can-i --list` option, which allows users to know what actions they can do in specific namespaces. (#64820, @WanLinghao)
- The `rules` field in RBAC `Role` and `ClusterRole` objects is now correctly reported as optional in the openapi schema. (#73250, @liggitt)
- `system:kube-controller-manager` and `system:kube-scheduler` users are now permitted to perform delegated authentication/authorization checks by default RBAC policy (#72491, @liggitt)
- Error messages returned in authentication webhook status responses are now correctly included in the apiserver log (#73595, @liggitt)
- Fixed use of webhook admission plugins with multi-version custom resources (#74154, @mbohlool)

AWS

- Prevented AWS Network Load Balancer security groups ingress rules to be deleted by ensuring target groups are tagged. (#73594, @masterzen)
- AWS ELB health checks will now use HTTPS/SSL protocol for HTTPS/SSL backends. (#70309, @2rs2ts)

Azure

- Fixed failure to detach Azure disk when there is server side error (#74398, @andyzhangx)
- Fixed subnet annotation checking for Azure internal loadbalancer (#74498, @feiskyer)
- Fixed mixed protocol issue for Azure load balancer (#74200, @andyzhangx)

- Fixed Azure accounts timeout issue when there is no out-bound IP (#74191, @andyzhangx)
- Fixed Azure Container Registry anonymous repo image pull error (#74715, @andyzhangx)
- Fixed parse devicePath issue on Azure Disk (#74499, @andyzhangx)

CLI

- Fixed `--help` flag parsing (#74682, @soltysh)
- Fixed a bug where `kubectl describe` cannot obtain the event messages for a static pod (#74156, @gaorong)
- Fixed panic when performing a `set env` operation on a `--local` resource (#65636, @juanvallejo)
- Missing directories listed in a user's PATH are no longer considered errors and are instead logged by the `kubectl plugin list` command when listing available plugins. (#73542, @juanvallejo)
- Now users can get object info like:

```
a. kubectl get pod test-pod -o custom-columns=CONTAINER:.spec.containers[0:3].name
b. kubectl get pod test-pod -o custom-columns=CONTAINER:.spec.containers[-2:].name
```

(#73063, @WanLinghao)

- The `kubectl api-resources` command will no longer fail to display any resources on a single failure (#73035, @juanvallejo)
- kubectl now loads config file once and uses persistent client config (#71117, @dixudx)
- Printed `SizeLimit` of `EmptyDir` in `kubectl describe pod` outputs. (#69279, @dtaniwaki)
- `kubectl delete --all-namespaces` is now a recognized flag. (#73716, @deads2k)

Cloud Provider

- Fixed a bug that caused PV allocation on non-English vSphere installations to fail (#73115, @alvaroaleman)

Cluster Lifecycle

- kubeadm: fixed nil pointer dereference caused by a bug in url parsing (#74454, @bart0sh)
- CoreDNS adds readinessProbe which prevents loadbalancing to unready pods, and also allows rolling updates to work as expected. (#74137, @rajansandeep)

- kubeadm no longer allows using v1alpha3 configs for anything else than converting them to `v1beta1`. (#74025, @rosti)
- kubeadm: now allows the usage of `--kubeconfig-dir` and `--config` flags on kubeadm init (#73998, @yagonobre)
- kubeadm: all master components are now exclusively relying on the `PriorityClassName` pod spec for annotating them as cluster critical components. Since `scheduler.alpha.kubernetes.io/critical-pod` annotation is no longer supported by Kubernetes 1.14 this annotation is no longer added to master components. (#73857, @ereslibre)
- kubeadm no longer dumps backtrace if it fails to remove the running containers on reset. (#73951, @rosti)
- kubeadm: fixed a bug in the underlying library for diff related to characters like '%' (#73941, @neolit123)
- Scale max-inflight now limits together with master VM sizes. (#73268, @wojtek-t)
- kubeadm reset: fixed a crash caused by the absence of a configuration file (#73636, @bart0sh)
- CoreDNS is now version 1.3.1 (#73610, @rajansandeep)
- kubeadm: When certificates are present in joining a new control plane now ensures that they match at least the required SANs (#73093, @ereslibre)
- kubeadm: added back `--cert-dir` option for `kubeadm init phase certs sa` (#73239, @mattkelly)
- kubeadm: now explicitly waits for `etcd` to have grown when joining a new control plane (#72984, @ereslibre)
- kubeadm: now pulls images when joining a new control plane instance (#72870, @MalloZup)
- Exited kube-proxy when configuration file changes (#59176, @dixudx)
- kube-addon-manager was updated to v9.0, and now uses kubectl v1.13.2 and prunes workload resources via the apps/v1 API (#72978, @liggitt)
- kubeadm: Now allows certain certs/keys to be missing on the secret when transferring secrets using `--experimental-upload-certs` feature (#75415, @ereslibre)

GCP

- Fixed liveness probe in fluentd-gcp cluster addon (#74522, @Pluies)
- Reduced GCE log rotation check from 1 hour to every 5 minutes. Rotation policy is unchanged (new day starts, log file size > 100MB). (#72062, @jpbetz)

Network

- Reduces the cache TTL for negative responses to 5s minimum. (#74093, @blakebarnett)

Node

- Fixed help message for `--container-runtime-endpoint`: only unix socket is support on Linux. (#74712, @feiskyer)
- Image garbage collection no longer fails for images with only one tag but more than one repository associated. (#70647, @corvus-ch)
- Re-issued Allocate grpc calls before starting a container that requests device-plugin resources if the cached state is missing. (#73824, @jiayingz)
- [CRI] Added a new field called `runtime_handler` into `PodSandbox` and `PodSandboxStatus` to track the `RuntimeClass` information of a pod. (#73833, @haiyanmeng)
- Kubelet now tries to stop containers in unknown state once before restart or remove. (#73802, @Random-Liu)
- When pleg channel is full, events are now discarded and count is recorded (#72709, @changyaowei)
- Fixed the unexpected `NotReady` status when Node's iops is full if the runtime is dockershim. (#74389, @answer1991)
- Fixed #73264 `cpuPeriod` was not reset, but used as set via flag, although it was disabled via alpha gate (#73342, @szuecs)
- Updated kubelet CLI summary documentation and generated webpage (#73256, @deitch)
- Set a low `oom_score_adj` for containers in pods with system-critical priorities (#73758, @sjenning)
- kubelet: Resolved hang/timeout issues when running large numbers of pods with unique `ConfigMap/Secret` references (#74755, @liggitt)
- Events reported for container creation, start, and stop now report the container name in the message and are more consistently formatted. (#73892, @smarterclayton)
- Removed stale `OutOfDisk` condition from kubelet side (#72507, @dixudx)
- Fixed the setting of `NodeAddresses` when using the vSphere Cloud-Provider and nodes that have multiple IP addresses. (#70805, @danwinship)
- Fixed dockershim panic issues when deleting docker images. (#75367, @feiskyer)
- Kubelet no longer watches `ConfigMaps` and `Secrets` for terminated pods, in worst scenario causing it to not be able to send other requests to kube-apiserver (#74809, @oxddr)
- A new `TaintNodesByCondition` admission plugin taints newly created Node objects as "not ready", to fix a race condition that could cause pods to be scheduled on new nodes before their taints were updated to accurately reflect their reported conditions. This admission plugin is enabled by default if the `TaintNodesByCondition` feature is enabled. (#73097, @bsalamat)
- kubelet now accepts `pid=<number>` in the `--system-reserved` and `--kube-reserved` options to ensure that the specified number of process

IDs will be reserved for the system as a whole and for Kubernetes system daemons respectively. Please reference `Kube Reserved` and `System Reserved` in `Reserve Compute Resources for System Daemons` in the Kubernetes documentation for general discussion of resource reservation. To utilize this functionality, you must set the feature gate `SupportNodePidsLimit=true` (#73651

Scheduling

- Improved fairness of the scheduling queue by placing pods which are attempted recently behind other pods with the same priority. (#73700, @denkensk)
- Improved scheduler robustness to ensure that unschedulable pods are reconsidered for scheduling when appropriate. (#73700, #72558, @denkensk, #73078, @Huang-Wei)

Storage

- Fixed scanning of failed iSCSI targets. (#74306, @jsafrane)
- StorageOS volume plugin updated to fix an issue where volume mount succeeds even if request to mount via StorageOS API fails. (#69782, @darkowlzz)
- Ensured directories on volumes are group-executable when using `fsGroup` (#73533, @mxey)
- Updated CSI version to 1.1 (#75391, @gnufied)
- Ensured that volumes get provisioned based on the zone information provided in `allowedTopologies`. (#72731, @skarthiksrinivas)
- Extended the `VolumeSubpathEnvExpansion` alpha feature to support environment variable expansion (#71351, @kevtaylor)
- Fixed a bug that prevented deletion of dynamically provisioned volumes in Quobyte backends. (#68925, @casusbelli)

Testing

- e2e storage tests now run faster and are easier to read (#72434, @pohly)
- `e2e.test` now rejects unknown `--provider` values instead of merely warning about them. An empty provider name is not accepted anymore and was replaced by `skeleton` (a provider with no special behavior). (#73402, @pohly)
- Updated to go1.11.5 (#73326, @ixdy)
- Updated to use go1.12.1 (#75413, @BenTheElder)
- e2e tests that require SSH may now be used against clusters that have nodes without external IP addresses by setting the environment variable

`KUBE_SSH_BASTION` to the `host:port` of a machine that is allowed to SSH to those nodes. The same private key that the test would use is used for the bastion host. The test connects to the bastion and then tunnels another SSH connection to the node. (#72286, @smarterclayton)
- `PidPressure` now evicts pods from lowest priority to highest priority (#72844, @dashpole)
- Split up the mondo `kubernetes-test` tarball into `kubernetes-test-portable` and `kubernetes-test-{OS}-{ARCH}` tarballs. (#74065, @ixdy)

VMware

- Applied zone labels to vSphere Volumes automatically. The zone labels are visible on the PV: (#72687, @subramanian-neelakantan)

Windows

Support for Windows nodes and Windows containers graduated to stable.

Support for Group Managed Service Accounts (GMSA) for Windows containers in Kubernetes is alpha. GMSA 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.

- Fixed smb remount and unmount issues on Windows (#73661, @andyzhangx, #75087, @andyzhangx)
- Added network stats for Windows nodes and containers (#74788, @feiskyer)
- The new test `[sig-network] DNS should now provide /etc/hosts entries for the cluster [LinuxOnly] [Conformance]` will validate the host entries set in the `/etc/hosts` file (pod's FQDN and hostname), which should be managed by Kubelet. (#72729, @bclau)
- Allowed the kubelet to pass Windows GMSA credentials down to Docker (#73726, @wk8)
- Added kube-proxy support for overlay networking and DSR in Windows and new flags for `network-name`, `source-vip`, and `enable-dsr`. (#70896, @ksubrmnn)
- windows: Ensured graceful termination when being run as windows service (#73292, @steffengy)
- vSphere cloud provider now correctly retrieves the VM's UUID when running on Windows (#71147, @benmoss)
- Kubelet: added `usageNanoCores` from CRI stats provider (#73659, @feiskyer)
- Introduced support for Windows nodes into the cluster bringup scripts for GCE. (#73442, @pjh)

- Added network stats for Windows nodes and pods. (#70121, @feiskyer)
- CoreDNS is only officially supported on Linux at this time. As such, when kubeadm is used to deploy this component into your kubernetes cluster, it will be restricted (using `nodeSelectors`) to run only on nodes with that operating system. This ensures that in clusters which include Windows nodes, the scheduler will not ever attempt to place CoreDNS pods on these machines, reducing setup latency and enhancing initial cluster stability. (#69940, @MarcPow)

External Dependencies

- Default etcd server and client have been updated to v3.3.10. (#71615, #70168)
- The list of validated docker versions has changed. 1.11.1 and 1.12.1 have been removed. The current list is 1.13.1, 17.03, 17.06, 17.09, 18.06, 18.09. (#72823, #72831)
- The default Go version was updated to 1.12.1. (#75422)
- CNI has been updated to v0.7.5 (#75455)
- CSI has been updated to v1.1.0. (#75391)
- The dashboard add-on has been updated to v1.10.1. (#72495)
- Cluster Autoscaler has been updated to v1.14.0 (#75480)
- kube-dns is unchanged at v1.14.13 since Kubernetes 1.12 (#68900)
- Influxdb is unchanged at v1.3.3 since Kubernetes 1.10 (#53319)
- Grafana is unchanged at v4.4.3 since Kubernetes 1.10 (#53319)
- Kibana has been upgraded to v6.6.1. (#71251)
- CAdvisor has been updated to v0.33.1 (#75140)
- fluentd-gcp-scaler is unchanged at v0.5.0 since Kubernetes 1.13 (#68837)
- Fluentd in fluentd-elasticsearch has been upgraded to v1.3.3 (#71180)
- fluentd-elasticsearch has been updated to v2.4.0 (#71180)
- The fluent-plugin-kubernetes_metadata_filter plugin in fluentd-elasticsearch has been updated to v2.1.6 (#71180)
- fluentd-gcp is unchanged at v3.2.0 since Kubernetes 1.13 (#70954)
- OIDC authentication is unchanged at coreos/go-oidc v2 since Kubernetes 1.10 (#58544)
- Calico is unchanged at v3.3.1 since Kubernetes 1.13 (#70932)

- crictl on GCE is unchanged at v1.12.0 since Kubernetes 1.13 (#69033)
- CoreDNS has been updated to v1.3.1 (#73610)
- event-exporter has been updated to v0.2.3 (#67691)
- Es-image has been updated to Elasticsearch 6.6.1 (#71252)
- metrics-server remains unchanged at v0.3.1 since Kubernetes 1.12 (#68746)
- GLBC remains unchanged at v1.2.3 since Kubernetes 1.12 (#66793)
- Ingress-gce remains unchanged at v1.2.3 since Kubernetes 1.12 (#66793)
- ip-masq-agen remains unchanged at v2.1.1 since Kubernetes 1.12 (#67916)
- v1.14.0-rc.1
- v1.14.0-beta.2
- v1.14.0-beta.1
- v1.14.0-alpha.3
- v1.14.0-alpha.2
- v1.14.0-alpha.1

v1.14.0-rc.1

Documentation

Downloads for v1.14.0-rc.1

filename	sha512 hash
kubernetes.tar.gz	`5cb5e8b14b301864063fd7531ab3b755fea054f540c55ecce70ac49fb59193488575eb42`
kubernetes-src.tar.gz	`395424c2bcdb5e242995b18e8d6e5c00002ce2cb5a3964c28da0a4a181fada73ffceacce`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	`ea304f3f8188db30cddd5423b25dc434b8f05315103f773619a65f83`
kubernetes-client-darwin-amd64.tar.gz	`17e106b63067429b9228a4879a7350c01ae98650ef2e6fcc23d00415`
kubernetes-client-linux-386.tar.gz	`406323ea4cbd524807e73b9a2f4eb0a813730b262402c224e5076080`
kubernetes-client-linux-amd64.tar.gz	`293747816e5da30c53ca29f27479fae880404edd5fca413af165e52c`

filename	sha512 hash
kubernetes-client-linux-arm.tar.gz	98b6749c367282048ecd0c5b70ae8b7dafad82c599e359cbbe782a53
kubernetes-client-linux-arm64.tar.gz	0a650c53946ebd9c38705df36efabbf1fbe3da1acdf418cc4ab88153
kubernetes-client-linux-ppc64le.tar.gz	807263f316f46c9334ec4dad84895cfe2b942ac4ed9aacf3ec8a6319
kubernetes-client-linux-s390x.tar.gz	5589562ce0da49f4987388e3e2b6fcf29e92859fae65fb57cdd61bba
kubernetes-client-windows-386.tar.gz	101245cd70221b443150be046e5b5a2c6c83334085bb17f603f59bb6
kubernetes-client-windows-amd64.tar.gz	b1c3802f60cc70ebc1258cb8fc4ffa1154ecee8fda473b033be4f9d1

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	8e09465aab0a1d1ec39afc98af17de9b5de99b763c0c5feb2dc824f2bb
kubernetes-server-linux-arm.tar.gz	9409c368e1f9f26e633b7df5d6c90435394931d48a21f4ad1548d172b1
kubernetes-server-linux-arm64.tar.gz	3893290dc11ca71746fee77a44a607ad9e02036bab56b7fc3be247b71b
kubernetes-server-linux-ppc64le.tar.gz	8ac1e70cb39aeb5b1fe92c3ebba428db2036be739f462cc2f876f17dc7
kubernetes-server-linux-s390x.tar.gz	80576cb6cc3a69c4fc0a0358dee5772ecb38437c534a3454c961342641

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	71cd5dd6075a2eea851231a5a855e58b3f479d83358defafd068dd1d0
kubernetes-node-linux-arm.tar.gz	48babf4a52013c2bc69049167579ba1bc70c769b782a2704c9dfcf44a
kubernetes-node-linux-arm64.tar.gz	cc19a52beaa7440ca7581e85d1e10137e93c2decdb7d7d7919e7fdcde
kubernetes-node-linux-ppc64le.tar.gz	241bf20ae366384efa0fb3798e07e1cdd3d4ea7ba91c146ef7761fb0b
kubernetes-node-linux-s390x.tar.gz	37ce3e021073c6c10cf240fb2c3f9a7ab35ece3c0b4a9fecbbdf790eb
kubernetes-node-windows-amd64.tar.gz	3248db12c274171f094ef41d6a59523aec35655ba35d151576ff2d208

Changelog since v1.14.0-beta.2

Action Required

- ACTION REQUIRED: Health check (liveness & readiness) probes using an HTTPGetAction will no longer follow redirects to different hostnames from the original probe request. Instead, these non-local redirects will be treated as a Success (the documented behavior). In this case an event with reason “ProbeWarning” will be generated, indicating that the redirect was ignored. If you were previously relying on the redirect to run health checks against different endpoints, you will need to perform the healthcheck logic outside the Kubelet, for instance by proxying the external endpoint rather than redirecting to it. (#75416, @tallclair)

Other notable changes

- Restores –username and –password flags to kubectl (#75451, @liggitt)
- fix race condition issue for smb mount on windows (#75371, @andyzhangx)
- UDP Service conntrack entries for ExternalIPs are now correctly cleared when endpoints are added (#75265, @JacobTanenbaum)
- kubeadm: the kubeadm init output now provides join control-plane example only when the preconditions for joining a control plane are satisfied (#75420, @fabriziopandini)
- Fix dockershim panic issues when deleting docker images. (#75367, @feiskyer)
- kubeadm: Allow certain certs/keys to be missing on the secret when transferring secrets using `--experimental-upload-certs` feature (#75415, @ereslibre)
- Update to use go1.12.1 (#75413, @BenTheElder)
- Update CSI version to 1.1 (#75391, @gnufied)
- Ensure ownership when deleting a load balancer security group (#74311, @hpedrorodrigues)
- kubelet: updated logic of verifying a static critical pod. (#75144, @Huang-Wei)
- Allow disable outbound SNAT when Azure standard load balancer is used together with outbound rules. (#75282, @feiskyer)
- Add ResourceVersion as a precondition for delete in order to ensure a delete fails if an unobserved change happens to an object. (#74040, @ajatprabha)
- Services of type=LoadBalancer which have no endpoints will now immediately ICMP reject connections, rather than time out. (#74394, @thockin)
- Ensure Azure load balancer cleaned up on 404 or 403 when deleting LoadBalancer services. (#75256, @feiskyer)

v1.14.0-beta.2

Documentation

Downloads for v1.14.0-beta.2

filename	sha512 hash
kubernetes.tar.gz	`c1d5f2615c3319fc167c577f40f385abe6652bf4fd3bdb04617b36029dc3000b190c18b4`
kubernetes-src.tar.gz	`0a8d8ed208bc0bf424060126c76fcd8dbbd53a9b9695647314a4097f7013f548b7685043`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	c919d030255c5d3879926d8aaa53939cd5aa37084799748452166ca6
kubernetes-client-darwin-amd64.tar.gz	ec14d4a1d720890065211544b099be17315265534cfd20435194dc84
kubernetes-client-linux-386.tar.gz	6cee12be5b855600ee80f15d1e0511099941b099bd5b252549abdc2a
kubernetes-client-linux-amd64.tar.gz	27d8cd48c1f3259055965b85a6b973ecb5b8a36894f94c232d773f89
kubernetes-client-linux-arm.tar.gz	7f98230569c61fdf2b141e519f042b2e27ff37aeda746dc30bb7ce22
kubernetes-client-linux-arm64.tar.gz	159da67010af38d87c5318b7ad594120afd6a9b780d11d6e607e7214
kubernetes-client-linux-ppc64le.tar.gz	d8ca7871d3d40947db69061284cb31c4d072d4da56fbb11a4485f685
kubernetes-client-linux-s390x.tar.gz	1c58db90b6e09b8d8f956a00263cb20271b8403f7fb6c5b20d76cca9
kubernetes-client-windows-386.tar.gz	00fb87dd4899208dd6607c22828f3985ebfd5e1f97cb24e3b2c69c24
kubernetes-client-windows-amd64.tar.gz	7afdf637d62dde480162ad1521360b2bc78e0d4d20f6e6201e2f19b5

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	2ac3c4910cd36f02a62304d78fe144b821edf445c522028e6b57d2dc3b
kubernetes-server-linux-arm.tar.gz	db06b5f1a83ca4ec82428ab771eae2858b188dc23780fb9b146494c06a
kubernetes-server-linux-arm64.tar.gz	b40e1745d1ecfcc95f3a750990244fa128381d6d74246798a62aecb8ce
kubernetes-server-linux-ppc64le.tar.gz	c84297fd2b18b6bdade5a135a3da929e286bbba5c8dd66778091bad4ee
kubernetes-server-linux-s390x.tar.gz	39c8f6e7f52bec155b11652b4e80c2c52acf8754dbdf80a9d5bab5370d

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	bf19ae7140836aea1b6f414532eff886e3b91e0746b9224ce46e60e0b
kubernetes-node-linux-arm.tar.gz	4b8194340a8675107da3969845173fb34ed2b0a38745ec0ac395ebf21
kubernetes-node-linux-arm64.tar.gz	8494dae5225f3b543afd575003fe0f30eb0f3cf9bc9dfbae72d6bda8f
kubernetes-node-linux-ppc64le.tar.gz	0bd41d95f0a76c1b057a8913a8b2cffbf6d48c47aef1d9beed0de205b
kubernetes-node-linux-s390x.tar.gz	169b16512df42a6cb5b000a3d6d6da5ae48a733c5d11b034eaec6b381
kubernetes-node-windows-amd64.tar.gz	ccfebde5176cb28529552889250bf706add04df3c3f9aee5b8bdd9ebc

Changelog since v1.14.0-beta.1

Action Required

- ACTION REQUIRED: The node.k8s.io API group and runtime-classes.node.k8s.io resource have been migrated to a built-in API. If you were using RuntimeClasses (an default-disabled alpha feature, as

of Kubernetes v1.12), then you must recreate all RuntimeClasses after upgrading, and the runtimeclasses.node.k8s.io CRD should be manually deleted. RuntimeClasses can no longer be created without a defined handler. (#74433, @tallclair)
    - Introduce a RuntimeClass v1beta1 API. This new beta API renames `runtimeHandler` to `handler`, makes it a required field, and cuts out the spec (handler is a top-level field).
- Transition CSINodeInfo and CSIDriver alpha CRDs to in-tree CSINode and CSIDriver core storage v1beta1 APIs. (#74283, @xing-yang)
    - ACTION REQUIRED: the alpha CRDs are no longer used and drivers will need to be updated to use the beta APIs.
    - The support for _ in the CSI driver name will be dropped as the CSI Spec does not allow that.

Other notable changes

- Support collecting pod logs under /var/log/pods/NAMESPACE_NAME_UID to stackdriver with `k8s_pod` resource type. (#74502, @Random-Liu)
- –make-symlinks for hyperkube was marked hidden for a while, This flag is now deprecated and will be removed in a future release. (#74975, @dims)
- fix smb unmount issue on Windows (#75087, @andyzhangx)
- Kubelet no longer watches configmaps and secrets for terminated pods, in worst scenario causing it to not be able to send other requests to kube-apiserver (#74809, @oxddr)
- - Fixes a bug concerning Quobyte volumes where user mappings only worked if the hosts Kubernetes plugin mount was provided via an external configuration using the *allow-usermapping-in-volumename* option. (#74520, @casusbelli)
- Change CRI pod log directory from `/var/log/pods/UID` to `/var/log/pods/NAMESPACE_NAME_UID`. (#74441, @Random-Liu)
    - It is recommended to drain the node before upgrade, or reboot the node after upgrade.
- Promote RuntimeClass to beta, and enable by default. (#75003, @tallclair)
- New "dry_run" metric label (indicating the value of the dryRun query parameter) into the metrics: (#74997, @jennybuckley) * apiserver_request_total * apiserver_request_duration_seconds
    - New "APPLY" value for the "verb" metric label which indicates a PATCH with "Content-Type: apply-patch+yaml". This value is experimental and will only be present if the ServerSideApply alpha feature is enabled.
- GCE: bump COS image version to cos-beta-73-11647-64-0 (#75149, @yguo0905)
- - Add duration metric for CRD webhook converters (#74376, @mbohlool)
- Alpha support for ephemeral CSI inline volumes that are embedded in

pod specs. (#74086, @vladimirvivien)
- Add support for node side CSI volume expansion (#74863, @gnufied)
- - Add mechanism for Admission Webhooks to specify which version of AdmissionReview they support (#74998, @mbohlool)
  - - Add mechanism for CRD Conversion Webhooks to specify which version of ConversionReview they support
- Add a new kubelet endpoint for serving first-class resource metrics (#73946, @dashpole)
- Deprecate AWS, Azure, GCE and Cinder specific volume limit predicates. (#74544, @gnufied)
- PodReadinessGate feature is now GA. The feature gate will not allow disabling it. (#74434, @freehan)
- If CSINodeInfo and CSIMigration feature flags are active in the cluster, Kubelet will post NotReady until CSINode is initialized with basic volume plugin mechanism information for well-known drivers (#74835, @davidz627)
- Add network stats for Windows nodes and containers (#74788, @feiskyer)
- kubeadm: when calling "reset" on a control-plane node, remove the API-Endpoint information for this node from the ClusterStatus in the kubeadm ConfigMap. (#75082, @neolit123)
- kube-apiserver now serves OpenAPI specs for registered CRDs with defined (#71192, @roycaihw)
  - validation schemata as an alpha feature, to be enabled via the "CustomResourcePublishOpenAPI" feature gate. Kubectl will validate client-side using those. Note that in
  - future, client-side validation in 1.14 kubectl against a 1.15 cluster will reject
  - unknown fields for CRDs with validation schema defined.
- Fix kubelet start failure issue on Azure Stack due to InstanceMetadata setting (#74936, @rjaini)
- add subcommand `kubectl create cronjob` (#71651, @Pingan2017)
- The CSIBlockVolume feature gate is now beta, and defaults to enabled. (#74909, @bswartz)
- Pre-existing log files are now opened with O_APPEND, instead of O_TRUNC. This helps prevent losing logs when components crash-loop, and also enables external log rotation utilities to truncate log files in-place without components extending log files to their pre-truncation sizes on subsequent writes. (#74837, @mtaufen)
- the test/e2e/e2e.test binary can test arbitrary storage drivers, see the `-storage.testdriver` parameter (#72836, @pohly)
- Fix panic in kubectl cp command (#75037, @soltysh)
- iscsi modules haven't even been loaded /sys/class/iscsi_host directory won't exist (#74787, @jianglingxia)
- the fluentd addon daemonset will now target all nodes. (#74424, @liggitt)
  * setting `ENABLE_METADATA_CONCEALMENT=true` in kube-up will now set a `cloud.google.com/metadata-proxy-ready=true` label on new nodes. In

v1.16, the metadata proxy add-on will switch to using that label as a node selector. * setting `KUBE_PROXY_DAEMONSET=true` in kube-up will now set a `node.kubernetes.io/kube-proxy-ds-ready=true` label on new nodes. In v1.16, the kube-proxy daemonset add-on will switch to using that label as a node selector. * In 1.16, the masq-agent daemonset add-on will switch to using `node.kubernetes.io/masq-agent-ds-ready` as a node selector.

- - Kubelet: replace `du` and `find` with a golang implementation (#74675, @dashpole)
  - - Kubelet: periodically update machine info to support hot-add/remove
- kubeadm: add certificate-key and skip-certificate-key-print flags to kubeadm init (#74671, @yagonobre)
- Admission webhooks rules can now limit scope to only match namespaced, or only cluster-scoped resources with a `scope: "Cluster" | "Namespaced" | "*"` field. (#74477, @liggitt)
- The CSIPersistentVolume and KubeletPluginWatcher feature gates cannot be disabled, and will be removed in Kubernetes v1.16 (#74830, @msau42)
- Kubelet won't evict a static pod with priority `system-node-critical` upon resource pressure. (#74222, @Huang-Wei)
- Fixes panic if a kubelet is run against an older kube-apiserver (#74529, @liggitt)
- The resource group name in Azure providerID is not converted to lower cases. (#74882, @feiskyer)
- Remove the out-of-tree PersistentVolumeLabel controller because it cannot run without Initializers (removed in v1.14). If you are using AWS EBS, GCE PD, Azure Disk, Cinder Disk or vSphere volumes and rely on zone labels, then enable the `PersistentVolumeLabel` admission controller in the `kube-apiserver` in the `--enable-admission-plugins` flag. (#74615, @andrewsykim)
- kubeadm: improved RequiredIPVSKernelModulesAvailable warning message (#74033, @bart0sh)
- Add `nullable` support to CustomResourceDefinition OpenAPI validation schemata. (#74804, @sttts)
- Fix kube-apiserver not to create default/kubernetes service endpoints before it reports readiness via the /healthz and therefore is ready to serve requests. Also early during startup old endpoints are remove which might be left over from a previously crashed kube-apiserver. (#74668, @sttts)
- kubeadm: fix a bug where standard kubeconfig paths were searched even if the user provided /etc/kubernetes/admin.conf explicitly for commands that accept –kubeconfig, like kubeadm token. (#71874, @neolit123)
  - kubeadm: use the default kubeconfig (/etc/kubernetes/admin.conf) for "kubeadm reset" and "kubeadm upgrade" commands.
- Increase api server client certificate expiration histogram resolution to accommodate short-lived (< 6h) client certificates. (#74806, @mxinden)
- Default RBAC policy no longer grants access to discovery and permission-checking APIs (used by `kubectl auth can-i`) to *unau-*

thenticated users. Upgraded clusters preserve prior behavior, but cluster administrators wishing to grant unauthenticated users access in new clusters will need to explicitly opt-in to expose the discovery and/or permission-checking APIs: (#73807, @dekkagaijin) * `kubectl create clusterrolebinding anonymous-discovery --clusterrole=system:discovery --group=system:unauthenticated` * `kubectl create clusterrolebinding anonymous-access-review --clusterrole=system:basic-user --group=system:unauthenticated`

- The PersistentLocalVolumes feature is GA. The feature gate cannot be disabled and will be removed in Kubernetes 1.17 (#74769, @msau42)
- kubelet: resolved hang/timeout issues when running large numbers of pods with unique configmap/secret references by reverting to 1.11 configmap/secret lookup behavior (#74755, @liggitt)
- Convert `latency/latencies` in metrics name to `duration`. (#74418, @danielqsj)
  - The following metrics are changed and mark previous metrics as deprecated:
    - `rest_client_request_latency_seconds` -> `rest_client_request_duration_seconds`
    - `apiserver_proxy_tunnel_sync_latency_secs` -> `apiserver_proxy_tunnel_sync_duration`
    - `scheduler_scheduling_latency_seconds` -> `scheduler_scheduling_duration_seconds`
- Fix help message for –container-runtime-endpoint: only unix socket is support on Linux. (#74712, @feiskyer)
- Update to use golang 1.12 (#74632, @cblecker)
- The `RunAsGroup` feature has been promoted to beta and enabled by default. PodSpec and PodSecurityPolicy objects can be used to control the primary GID of containers on supported container runtimes. (#73007, @krmayankk)
- fix Azure Container Registry anonymous repo image pull error (#74715, @andyzhangx)
- Adds the same information to an init container as a standard container in a pod when using PodPresets. (#71479, @soggiest)
- fix the flake in scheduling_queue_test.go (#74611, @denkensk)
- The kube-apiserver OpenAPI definitions with the prefix "io.k8s.kubernetes.pkg" (deprecated since 1.9) have been removed. (#74596, @sttts)
- kube-conformance image will now run ginkgo with the –dryRun flag if the container is run with the environment variable E2E_DRYRUN set. (#74731, @johnSchnake)
- The deprecated `MountPropagation` feature gate has been removed, and the feature is now unconditionally enabled. (#74720, @bertinatto)
- Introduce dynamic volume provisioning shim for CSI migration (#73653, @ddebroy)
- Fix –help flag parsing (#74682, @soltysh)
- This PR removes the following metrics: (#74636, @logicalhan)
  - reflector_items_per_list
  - reflector_items_per_watch
  - reflector_last_resource_version

  - reflector_list_duration_seconds
  - reflector_lists_total
  - reflector_short_watches_total
  - reflector_watch_duration_seconds
  - reflector_watches_total
  - While this is a backwards-incompatible change, it would have been impossible to setup reliable monitoring around these metrics since the labels were not stable.
- Add a configuration field to shorten the timeout of validating/mutating admission webhook call. The timeout value must be between 1 and 30 seconds. Default to 30 seconds when unspecified. (#74562, @roycaihw)
- client-go: PortForwarder.GetPorts() now contain correct local port if no local port was initially specified when setting up the port forwarder (#73676, @martin-helmich)
- # Apply resources from a directory containing kustomization.yaml (#74140, @Liujingfang1)
  - kubectl apply -k dir
  - # Delete resources from a directory containing kustomization.yaml.
  - kubectl delete -k dir
  - # List resources from a directory containing kustomization.yaml
  - kubectl get -k dir
- kubeadm: Allow to download certificate secrets uploaded by `init` or `upload-certs` phase, allowing to transfer certificate secrets (certificates and keys) from the cluster to other master machines when creating HA deployments. (#74168, @ereslibre)
- Fixes an issue with missing apiVersion/kind in object data sent to admission webhooks (#74448, @liggitt)
- client-go: the deprecated versionless API group accessors (like `clientset.Apps()` have been removed). Use an explicit version instead (like `clientset.AppsV1()`) (#74422, @liggitt)
- The `--quiet` option to `kubectl run` now suppresses resource deletion messages emitted when the `--rm` option is specified. (#73266, @awh)
- Add Custom Resource support to "kubectl autoscale" (#72678, @rmohr)
- Image garbage collection no longer fails for images with only one tag but more than one repository associated. (#70647, @corvus-ch)
- - Fix liveness probe in fluentd-gcp cluster addon (#74522, @Pluies)
- The new test `[sig-network] DNS should provide /etc/hosts entries for the cluster [LinuxOnly] [Conformance]` will validate the host entries set in the `/etc/hosts` file (pod's FQDN and hostname), which should be managed by Kubelet. (#72729, @bclau)
  - The test has the tag `[LinuxOnly]` because individual files cannot be mounted in Windows Containers, which means that it cannot pass using Windows nodes.

v1.14.0-beta.1

Documentation

Downloads for v1.14.0-beta.1

filename	sha512 hash
kubernetes.tar.gz	065014c751635f8c077fbcc105df578594baf8afd8b8339004909198e1bd68d0a7ce3644
kubernetes-src.tar.gz	244c19d9fad21ae154ee78fc94888dc60bcfcf3ec72bdc28a82e77c572cbc969d2abbf20

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	78a61a1e922daa39a9f7dd61b8bad87d202b537bda59f90ae8aae941
kubernetes-client-darwin-amd64.tar.gz	5a1d66fd90a5dc07b95b7617b5583595e0a4a664e7005f6281f846c8
kubernetes-client-linux-386.tar.gz	30991ce0776bae7551b98a811e3ccb5104b0859805c41a216db7d577
kubernetes-client-linux-amd64.tar.gz	5501a9c3a95e43f0c691b05043546f2c497d50a6ad88b88219842d61
kubernetes-client-linux-arm.tar.gz	4bf341a2f943719d006f4cacc26fdf4d021560d37d49c8d9c4620d29
kubernetes-client-linux-arm64.tar.gz	1c0c660618947b053404ab8ef40bdec0a06d54a1f9edc585a7259806
kubernetes-client-linux-ppc64le.tar.gz	9d2c3f4bfbad03b41859670f9cfda6596f51a89077fd4da2f74490f7
kubernetes-client-linux-s390x.tar.gz	9be5259caf39ff3c4d0f024d616bff50ac417d42a87c56b6877fbdf5
kubernetes-client-windows-386.tar.gz	5b1e75f532d9a4d656cd1c5ec48a19d01c4ca731c6c3d986ebbc48a9
kubernetes-client-windows-amd64.tar.gz	a501d3c0be55d5a73214a83d1f48b39a49d4a84f5f9988fa34cf66ef

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	aed44ec5bc6bdaf41c20824a9841ee541bfd23362966f9326eca2cafbd
kubernetes-server-linux-arm.tar.gz	e751cb675013183a70a8817dca0b5c456a1ccf075244b411317e813b51
kubernetes-server-linux-arm64.tar.gz	3aabb1a9bd07413d0740adfb638b0e5ca4cd4a58eda244c5ebc1ea0178
kubernetes-server-linux-ppc64le.tar.gz	3c57e2b47b3b9ddc1039684840886877947141b1e4d31f909793678544
kubernetes-server-linux-s390x.tar.gz	fe418cfafc63cbccc1898086296e52026df27ff498753089792175ce0d

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	83b63d7e9d18fe35564105fd70629af9ba8f20112933b3ace92a48887
kubernetes-node-linux-arm.tar.gz	e4ac000be25ced9b308ec698da9702bd10a0a7183bfea9648500ab9d0

filename	sha512 hash
kubernetes-node-linux-arm64.tar.gz	`35fa5ec8a7fc001fe33abd89f8a446ce0ea2a011db27dc8ff544c2b19`
kubernetes-node-linux-ppc64le.tar.gz	`a5b938e9cdc39fc292269af4c3961a17b9bcdaf85b3c58db680f2d1a4`
kubernetes-node-linux-s390x.tar.gz	`d8e2b5a945f881ddb0a25576a614d564ccef0ad4e93c84b30cdc5788`
kubernetes-node-windows-amd64.tar.gz	`426774466800ca11cae16821c7d92917b2ce138fcb3f6dca813ec0f06`

Changelog since v1.14.0-alpha.3

Action Required

- Added cadvisor metric labels `pod` and `container` where `pod_name` and `container_name` are present to match instrumentation guidelines. (#69099, @ehashman)
  - Action required: any Prometheus queries that match `pod_name` and `container_name` labels (e.g. cadvisor or kubelet probe metrics) should be updated to use `pod` and `container` instead. `pod_name` and `container_name` labels will be present alongside `pod` and `container` labels for one transitional release and removed in the future.

Other notable changes

- fix parse devicePath issue on Azure Disk (#74499, @andyzhangx)
- fix issue: fail to detach azure disk when there is server side error (#74398, @andyzhangx)
- Allow Cinder volume limit to be configured from node too (#74542, @gnufied)
- Fix subnet annotation checking for Azure internal loadbalancer (#74498, @feiskyer)
- Allow the kubelet to pass Windows GMSA credentials down to Docker (#73726, @wk8)
- PriorityClass in scheduling.k8s.io/v1beta1 and scheduling.k8s.io/v1alpha1 are deprecated by PriorityClass scheduling.k8s.io/v1 and will not be served starting in v1.17. (#74465, @bsalamat)
- kubeadm: fixed nil pointer dereference caused by a bug in url parsing (#74454, @bart0sh)
- Fix the unexpected NotReady status when Node's iops is full if the runtime is dockershim. (#74389, @answer1991)
- Split up the mondo `kubernetes-test` tarball into `kubernetes-test-portable` and `kubernetes-test-{OS}-{ARCH}` tarballs. (#74065, @ixdy)
- Move fluentd-elasticsearch addon images to community controlled location (#73819, @coffeepac)
- The PriorityClass API has been promoted to `scheduling.k8s.io/v1` with no changes. The `scheduling.k8s.io/v1beta1` version is now deprecated

and will stop being served by default in v1.17. (#73555, @bsalamat)
- fix get azure accounts timeout issue when there is no out-bound IP (#74191, @andyzhangx)
- fix mixed protocol issue for azure load balancer (#74200, @andyzhangx)
- Don't update the Pod object after each scheduling attempt by adding a timestamp to the scheduling queue. (#73700, @denkensk)
- kubeadm: remove local etcd members from the etcd cluster when kubeadm reset (#74112, @pytimer)
- Fix keymutex issues which may crash in some platforms. (#74348, @danielqsj)
- Fixed scanning of failed iSCSI targets. (#74306, @jsafrane)
- kubeadm: Do not fail preflight checks when running on >= 5.0 Linux kernel (#74355, @brb)
- Reduces the cache TTL for negative responses to 5s minimum. (#74093, @blakebarnett)
- The Ingress API is now available via `networking.k8s.io/v1beta1`. `extensions/v1beta1` Ingress objects are deprecated and will no longer be served in v1.18. (#74057, @liggitt)
- kubelet's –containerized flag will no longer be supported and will be removed in a future release (#74267, @dims)
- Optimize scheduler cache snapshot algorithm to improve scheduling throughput. (#74041, @bsalamat)
- Extends the VolumeSubpathEnvExpansion alpha feature to support environment variable expansion (#71351, @kevtaylor)
  - Implements subPathExpr field for expanding environment variables into a subPath
  - The fields subPathExpr and subPath are mutually exclusive
  - Note: This is a breaking change from the previous version of this alpha feature
- Added kube-proxy support for overlay networking and DSR in Windows and new flags for network-name, source-vip, and enable-dsr. (#70896, @ksubrmnn)
- StorageOS volume plugin updated to fix an issue where volume mount succeeds even if request to mount via StorageOS API fails. (#69782, @darkowlzz)
- kubeadm: Allow to upload certificates required to join a new control-plane to kubeadm-certs secret using the flag `--experimental-upload-certs` on `init` or upload-certs phase. (#73907, @yagonobre)
- export query parameter is deprecated and will be removed in a future release (#73783, @deads2k)
- e2e storage tests run faster and are easier to read (#72434, @pohly)
- kubectl: fix a bug where "describe" cannot obtain the event messages for a static pod (#74156, @gaorong)
- windows: Ensure graceful termination when being run as windows service (#73292, @steffengy)
- CoreDNS adds readinessProbe which prevents loadbalancing to unready

pods, and also allows rolling updates to work as expected. (#74137, @rajansandeep)
- Fixes use of webhook admission plugins with multi-version custom resources (#74154, @mbohlool)
- kubeadm no longer allows using v1alpha3 configs for anything else than converting them to v1beta1. (#74025, @rosti)
- Change kubelet metrics to conform metrics guidelines. (#72470, @danielqsj)
  - The following metrics are deprecated, and will be removed in a future release:
    * `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`
  - Please convert to the following metrics:
    * `kubelet_pod_worker_duration_seconds`
    * `kubelet_pod_start_duration_seconds`
    * `kubelet_cgroup_manager_duration_seconds`
    * `kubelet_pod_worker_start_duration_seconds`
    * `kubelet_pleg_relist_duration_seconds`
    * `kubelet_pleg_relist_interval_seconds`
    * `kubelet_eviction_stats_age_seconds`
    * `kubelet_runtime_operations_total`
    * `kubelet_runtime_operations_duration_seconds`
    * `kubelet_runtime_operations_errors_total`
    * `kubelet_device_plugin_registration_total`
    * `kubelet_device_plugin_alloc_duration_seconds`
- This change ensures that volumes get provisioned based on the zone information provided in allowedTopologies. (#72731, @skarthiksrinivas)
  - Storage class spec:
  - kind: StorageClass
  - apiVersion: storage.k8s.io/v1
  - metadata:
  - name: fastpolicy1
  - provisioner: kubernetes.io/vsphere-volume
  - parameters:
  - diskformat: zeroedthick
  - storagePolicyName: vSAN Default Storage Policy

  - allowedTopologies:
  - - matchLabelExpressions:
  - - key: failure-domain.beta.kubernetes.io/zone
  - values:
  - - zone1
  - PV creation Logs:
  - I0109 11:17:52.321372 1 vsphere.go:1147] Starting to create a vSphere volume with volumeOptions: &{CapacityKB:1048576 Tags:map[kubernetes.io/created-for/pvc/namespace:default kubernetes.io/created-for/pvc/name:pvcsc-1-policy kubernetes.io/created-for/pv/name:pvc-34650c12-1400-11e9-aef4-005056804cc9] Name:kubernetes-dynamic-pvc-34650c12-1400-11e9-aef4-005056804cc9 DiskFormat:zeroedthick Datastore: VSANStorageProfileData: StoragePolicyName:vSAN Default Storage Policy StoragePolicyID: SCSIControllerType: Zone:[zone1]}
  - ...
  - I0109 11:17:59.430113 1 vsphere.go:1334] The canonical volume path for the newly created vSphere volume is "[vsanDatastore] 98db185c-6683-d8c7-bc55-0200435ec5da/kubernetes-dynamic-pvc-34650c12-1400-11e9-aef4-005056804cc9.vmdk"
  - Ran regression tests (no zone) and they passed.
- vSphere cloud provider correctly retrieves the VM's UUID when running on Windows (#71147, @benmoss)
- Re-issue Allocate grpc calls before starting a container that requests device-plugin resources if the cached state is missing. (#73824, @jiayingz)
- [CRI] Add a new field called `runtime_handler` into PodSandbox and PodSandboxStatus to track the RuntimeClass information of a pod. (#73833, @haiyanmeng)
- kubelet: OS and Arch information is now recorded in `kubernetes.io/os` and `kubernetes.io/arch` labels on Node objects. The previous labels (`beta.kubernetes.io/os` and `beta.kubernetes.io/arch`) are still recorded, but are deprecated and targeted for removal in 1.18. (#73333, @yujuhong)
- This change applies zone labels to vSphere Volumes automatically. The zone labels are visible on the PV: (#72687, @subramanian-neelakantan)
  - $ kubectl get pv –show-labels
  - NAME CAPACITY ACCESSMODES STATUS CLAIM REASON AGE LABELS
  - pv-abc 5Gi RWO Bound default/claim1 46s failure-domain.beta.kubernetes.io/region=VC1,failure-domain.beta.kubernetes.io/zone=cluster-1
- fix smb remount issue on Windows (#73661, @andyzhangx)
- Kubelet now tries to stop containers in unknown state once before restart or remove. (#73802, @Random-Liu)
- Deprecate –export flag from kubectl get command. (#73787, @soltysh)
- Breaking changes in client-go: (#72214, @caesarxuchao)
  - The disk-cached discovery client is moved from k8s.io/client-

go/discovery to k8s.io/client-go/discovery/cached/disk.
  - The memory-cached discovery client is moved from k8s.io/client-go/discovery/cached to k8s.io/client-go/discovery/cached/memory.
- kubelet now accepts `pid=<number>` in the `--system-reserved` and `--kube-reserved` options to ensure that the specified number of process IDs will be reserved for the system as a whole and for Kubernetes system daemons respectively. Please reference `Kube Reserved` and `System Reserved` in `Reserve Compute Resources for System Daemons` in the Kubernetes documentation for general discussion of resource reservation. To utilize this functionality, you must set the feature gate `SupportNodePidsLimit=true` (#73651, @RobertKrawitz)
- The apiserver, including both the kube-apiserver and apiservers built with the generic apiserver library, will now return 413 RequestEntityTooLarge error if a json patch contains more than 10,000 operations. (#74000, @caesarxuchao)
- kubeadm: allow the usage of –kubeconfig-dir and –config flags on kubeadm init (#73998, @yagonobre)
- when pleg channel is full, discard events and record its count (#72709, @changyaowei)
- Is ->It in line 6 (#73898, @xiezongzhe)
- Events reported for container creation, start, and stop now report the container name in the message and are more consistently formatted. (#73892, @smarterclayton)
- `kubectl auth reconcile` now outputs details about what changes are being made (#71564, @liggitt)
- kubeadm: fix a bug in the underlying library for diff related to characters like '%' (#73941, @neolit123)
- kube-apiserver: a request body of a CREATE/UPDATE/PATCH/DELETE resource operation larger than 100 MB will return a 413 "request entity too large" error. (#73805, @caesarxuchao)
  - Custom apiservers built with the latest apiserver library will have the 100MB limit on the body of resource requests as well. The limit can be altered via ServerRunOptions.MaxRequestBodyBytes.
  - The body size limit does not apply to subresources like pods/proxy that proxy request content to another server.
- Kustomize is developed in its own repo https://github.com/kubernetes-sigs/kustomize (#73033, @Liujingfang1)
  - This PR added a new subcommand `kustomize` in kubectl.
  - kubectl kustomize has the same effect as kustomize build
  - To build API resources from somedir with a kustomization.yaml file
  - kubectl kustomize
  - This command can be piped to apply or delete
  - kubectl kustomize | kubectl apply -f -
  - kubectl kustomize | kubectl delete -f -
- kubeadm: all master components are now exclusively relying on the `PriorityClassName` pod spec for annotating them as cluster critical

components. Since `scheduler.alpha.kubernetes.io/critical-pod` annotation is no longer supported by Kubernetes 1.14 this annotation is no longer added to master components. (#73857, @ereslibre)
- Speedup kubectl by >10 when calling out to kube-apiserver for discovery information. (#73345, @sttts)
- kubeadm no longer dumps backtrace if it fails to remove the running containers on reset. (#73951, @rosti)

v1.14.0-alpha.3

Documentation

Downloads for v1.14.0-alpha.3

filename	sha512 hash
kubernetes.tar.gz	`5060dcf689dad4e19da5029eb8fc3060a4b2bad988fddff438d0703a45c02481bcfbc15f`
kubernetes-src.tar.gz	`754c948b5d25b01f211866d473257be5fb576b4b97703eb6fc08679d6525e1f53195a450`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	`5bd74dfc86bacf89d6b05d541e13bf390216039a42cc90fef2b24882`
kubernetes-client-darwin-amd64.tar.gz	`34e16661d66d337083583dfb478756ec8cc664d7cfc2dd1817bf1da0`
kubernetes-client-linux-386.tar.gz	`15f99e85bcc95f7b8e1b4c6ecc23de36e89a54108003db926e97ec2e`
kubernetes-client-linux-amd64.tar.gz	`2e61cf9b776150c4f1830d068ffee9701cb04979152ed6b62fc1bf53`
kubernetes-client-linux-arm.tar.gz	`67fb3805bb1b4a77f6603fbde9bd1d26e179de1a594c85618aa7b17b`
kubernetes-client-linux-arm64.tar.gz	`28930dc384b51051081a52874bc4d6dafa3c992dfa214b977ef711de`
kubernetes-client-linux-ppc64le.tar.gz	`f59eda797a57961d52fe67ba8b25a3a10267f9ce46029ed2140ef4b0`
kubernetes-client-linux-s390x.tar.gz	`c56bfb64e55cf95251157a8229a3e94310b2c46bb1c1250050893873`
kubernetes-client-windows-386.tar.gz	`e49a00fbe600892dc5eed0bc21bac64806da65280c818ca79b5e8adb`
kubernetes-client-windows-amd64.tar.gz	`797e20969ed4935adcbc80ccbcd72ec5aa697e70b0d071eceefc6dba`

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	`eddfc9afd7337475c3865443170d1425dcf4a87d981555871a69bcf132`
kubernetes-server-linux-arm.tar.gz	`dc85cd3a039cc0516beb19018c8378f3b7b88fa2edb8fa1476305e89eb`
kubernetes-server-linux-arm64.tar.gz	`d7c3a72abaa4c3e3243f8b4b3a8adb8be2758e0f883423ea62d2c61b20`

filename	sha512 hash
kubernetes-server-linux-ppc64le.tar.gz	b268a94eb056eea8bdf4d5739dec430f75a6a6b3c18e30df68d970c356
kubernetes-server-linux-s390x.tar.gz	f4cfd8d2faacdd1f0065f9e0f4f8d0db7bd8f438f812f70a07f4cb5272

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	7040ee3c032ec4fe14530c3e47ee53d731acb947b06e2d560cbcd0e7e
kubernetes-node-linux-arm.tar.gz	3d32e5243d1c65bce573cfb0f60d643ef3fc684a15551dbc8c3d5435e
kubernetes-node-linux-arm64.tar.gz	d3a17027fa1c057528422b35e32260f5b7c7246400df595f0ebda5d15
kubernetes-node-linux-ppc64le.tar.gz	89ed1f5093b49ab9d58d7a70089e881bf388f3316cb2607fa18e3bf07
kubernetes-node-linux-s390x.tar.gz	755a60824a9b8c4090a791d332e410692708ecece90e37388f58eb2c7
kubernetes-node-windows-amd64.tar.gz	c71d8055d89e535771f345e0f673da021915a7a82c75951855ba2574a

Changelog since v1.14.0-alpha.2

Action Required

- The –storage-versions flag of kube-apiserver is removed. The storage versions will always be the default value built-in the kube-apiserver binary. (#67678, @caesarxuchao)

Other notable changes

- fix #73264 cpuPeriod was not reset, but used as set via flag, although it was disabled via alpha gate (#73342, @szuecs)
- Update kubelet CLI summary documentation and generated Webpage (#73256, @deitch)
- Considerably reduced the CPU load in kube-apiserver while aggregating OpenAPI specifications from aggregated API servers. (#71223, @sttts)
- kubeadm: add a preflight check that throws a warning if the cgroup driver for Docker on Linux is not “systemd” as per the k8s.io CRI installation guide. (#73837, @neolit123)
- Kubelet: add usageNanoCores from CRI stats provider (#73659, @feiskyer)
- Fix watch to not send the same set of events multiple times causing watcher to go back in time (#73845, @wojtek-t)

- `system:kube-controller-manager` and `system:kube-scheduler` users are now permitted to perform delegated authentication/authorization checks by default RBAC policy (#72491, @liggitt)
- Prevent AWS Network Load Balancer security groups ingress rules to be deleted by ensuring target groups are tagged. (#73594, @masterzen)
- Set a low oom_score_adj for containers in pods with system-critical priorities (#73758, @sjenning)
- Ensure directories on volumes are group-executable when using fsGroup (#73533, @mxey)
- kube-apiserver now only aggregates openapi schemas from `/openapi/v2` endpoints of aggregated API servers. The fallback to aggregate from `/swagger.json` has been removed. Ensure aggregated API servers provide schema information via `/openapi/v2` (available since v1.10). (#73441, @roycaihw)
- Change docker metrics to conform metrics guidelines and using histogram for better aggregation. (#72323, @danielqsj)
  - The following metrics are deprecated, and will be removed in a future release:
    - `docker_operations`
    - `docker_operations_latency_microseconds`
    - `docker_operations_errors`
    - `docker_operations_timeout`
    - `network_plugin_operations_latency_microseconds`
  - Please convert to the following metrics:
    - `docker_operations_total`
    - `docker_operations_latency_seconds`
    - `docker_operations_errors_total`
    - `docker_operations_timeout_total`
    - `network_plugin_operations_latency_seconds`
- `kubectl delete --all-namespaces` is a recognized flag. (#73716, @deads2k)
- MAC Address filter has been fixed in vSphere Cloud Provider, it no longer ignores `00:1c:14` and `00:05:69` prefixes (#73721, @frapposelli)
- Add kubelet_node_name metrics. (#72910, @danielqsj)
- The HugePages feature gate has graduated to GA, and can no longer be disabled. The feature gate will be removed in v1.16 (#72785, @derekwaynecarr)
- Fix a bug that aggregated openapi spec may override swagger securityDefinitions and swagger info in kube-apiserver (#73484, @roycaihw)

- Fixes a bug that prevented deletion of dynamically provisioned volumes in Quobyte backends. (#68925, @casusbelli)
- error messages returned in authentication webhook status responses are now correctly included in the apiserver log (#73595, @liggitt)
- kubeadm: `kubeadm alpha preflight` and `kubeadm alpha preflight node` are removed; you can now use `kubeadm join phase preflight` (#73718, @fabriziopandini)
- kube-apiserver: the deprecated `repair-malformed-updates` has been removed (#73663, @danielqsj)
- e2e.test now rejects unknown –provider values instead of merely warning about them. An empty provider name is not accepted anymore and was replaced by "skeleton" (= a provider with no special behavior). (#73402, @pohly)
- Updated AWS SDK to v1.16.26 for ECR PrivateLink support (#73435, @micahhausler)
- Expand kubectl wait to work with more types of selectors. (#71746, @rctl)
- The CustomPodDNS feature gate has graduated to GA, and can no longer be disabled. The feature gate will be removed in v1.16 (#72832, @MrHohn)
- The `rules` field in RBAC Role and ClusterRole objects is now correctly reported as optional in the openapi schema. (#73250, @liggitt)
- AWS ELB health checks will now use HTTPS/SSL protocol for HTTPS/SSL backends. (#70309, @2rs2ts)
- kubeadm reset: fixed crash caused by absence of a configuration file (#73636, @bart0sh)
- CoreDNS is now version 1.3.1 (#73610, @rajansandeep)
  - - A new `k8s_external` plugin that allows external zones to point to Kubernetes in-cluster services.
  - - CoreDNS now checks if a zone transfer is allowed. Also allow a TTL of 0 to avoid caching in the cache plugin.
  - - TTL is also applied to negative responses (NXDOMAIN, etc).
- Missing directories listed in a user's PATH are no longer considered errors and are instead logged by the "kubectl plugin list" command when listing available plugins. (#73542, @juanvallejo)
- remove kubelet flag '–experimental-fail-swap-on' (deprecated in v1.8) (#69552, @Pingan2017)
- Introduced support for Windows nodes into the cluster bringup scripts for GCE. (#73442, @pjh)

- Now users could get object info like: (#73063, @WanLinghao)
  - a. kubectl get pod test-pod -o custom-columns=CONTAINER:.spec.containers[0:3].name
  - b. kubectl get pod test-pod -o custom-columns=CONTAINER:.spec.containers[-2:].name
- scheduler: use incremental scheduling cycle in PriorityQueue to put all in-flight unschedulable pods back to active queue if we received move request (#73309, @cofyc)
- fixes an error processing watch events when running skewed apiservers (#73482, @liggitt)
- Prometheus metrics for crd_autoregister, crd_finalizer and crd_naming_condition_controller are exported. (#71767, @roycaihw)
- Adds deleting pods created by DaemonSet assigned to not existing nodes. (#73401, @krzysztof-jastrzebski)
- Graduate Pod Priority and Preemption to GA. (#73498, @bsalamat)
- Adds configuration for AWS endpoint fine control: (#72245, @ampsingram)
  - OverrideEndpoints bool Set to true to allow custom endpoints
  - ServiceDelimiter string Delimiter to use to separate overridden services (multiple services) Defaults to "&"
  - ServicenameDelimiter string Delimiter to use to separate servicename from its configuration parameters Defaults "|"
  - OverrideSeparator string Delimiter to use to separate region of occurrence, url and signing region for each override Defaults to ","
  - ServiceOverrides string example: s3|region1, https://s3.foo.bar, some signing_region & ec2|region2, https://ec2.foo.bar, signing_region
- The CoreDNS configuration now has the forward plugin for proxy in the default configuration instead of the proxy plugin. (#73267, @rajansandeep)
- Fixed a bug that caused PV allocation on non-English vSphere installations to fail (#73115, @alvaroaleman)

v1.14.0-alpha.2

Documentation

Downloads for v1.14.0-alpha.2

filename	sha512 hash

filename	sha512 hash
kubernetes.tar.gz	1330e4421b61f6b1e6e4dee276d4742754bd3dd4493508d67ebb4445065277c619c4da8b
kubernetes-src.tar.gz	352c043bebf13a616441c920f3eec80d3f02f111d8488c31aa903e1483bce6d1fbe74722

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	ee5aba4efce323167e6d897a2ff6962a240e466333bcae9390be2c85
kubernetes-client-darwin-amd64.tar.gz	4b5c0b340322956a8d096c595124a765ac318d0eb460d6320218f247
kubernetes-client-linux-386.tar.gz	7a5bfe68dd58c8478746a410872b615daf8abb9a78754140fb4d014a
kubernetes-client-linux-amd64.tar.gz	c3139f58070241f2da815f701af3c0bd0ea4fdec1fe54bb859bd1123
kubernetes-client-linux-arm.tar.gz	9d96d2e1e11aa61e2c3a5f4f27c18866feae9833b6ee70b15f5cdb5f
kubernetes-client-linux-arm64.tar.gz	7b4dd825cf9f217c18b28976a3faa94f0bd4868e541e5be7d57cd770
kubernetes-client-linux-ppc64le.tar.gz	490638e250c24b6bad8b67358fd7890f7a2f6456ae8ffe537c28bb5b
kubernetes-client-linux-s390x.tar.gz	9dd8c3361eda15dd1594066c55b79cb9a34578c225b2b48647cd5b34
kubernetes-client-windows-386.tar.gz	d624b8aead053201765b713d337528be82a71328ee3dd569f556868c
kubernetes-client-windows-amd64.tar.gz	a1cf8c67984dd4eb4610fa05d27fe9e9e4123159f933e3986e9db835

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	b93982b56371994c540cd11e6bc2180827934061716499 2c10f30d8e6a
kubernetes-server-linux-arm.tar.gz	bfd76c6b26e5927166d776f6110b97ee36c1d63ad39e2d18899f3e428e
kubernetes-server-linux-arm64.tar.gz	6721dec0df9466cd6c056160c73d598296cebb0af9259eb21b693abb87
kubernetes-server-linux-ppc64le.tar.gz	f8cd307db8141d989ae1218dd2b438bc9cee017d533b1451d2345f9689
kubernetes-server-linux-s390x.tar.gz	de7514bbd87a1b363e1bc7787f37d5ea10faac4afe7c5163c23c4df167

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	8c37fd2fe6232d2c148e23df021b8b5347136263399932bcdff0c7a01
kubernetes-node-linux-arm.tar.gz	389e4e77ab9e62968a25b8f4e146a2c3fbb3db2e60e051922edf6395c
kubernetes-node-linux-arm64.tar.gz	7efc32dfeefcef7f860913c25431bd891a435e92cb8d5a95f8deca1a8
kubernetes-node-linux-ppc64le.tar.gz	da30c03bca4b81d810a7df006db02333dea87e336d6cdca9c93392e01
kubernetes-node-linux-s390x.tar.gz	cce43b7f0350b9e5a77ea703225adb9714ef022d176db5b99a0327937
kubernetes-node-windows-amd64.tar.gz	d3accf522d80cbfb3d03e9eaa60a09767ba11e88a8a5b44a629192a7c

Changelog since v1.14.0-alpha.1

Action Required

- Promote ValidateProxyRedirects to Beta, and enable by default. This feature restricts redirect following from the apiserver to same-host redirects. (#72552, @tallclair)
  - ACTION REQUIRED: If nodes are configured to respond to CRI streaming requests on a different host interface than what the apiserver makes requests on (only the case if not using the built-in dockershim & setting the kubelet flag `--redirect-container-streaming=true`), then these requests will be broken. In that case, the feature can be temporarily disabled until the node configuration is corrected. We suggest setting `--redirect-container-streaming=false` on the kubelet to avoid issues.

Other notable changes

- Added alpha field storageVersionHash to the discovery document for each resource. Its value must be treated as opaque by clients. Only equality comparison on the value is valid. (#73191, @caesarxuchao)
- Fix admission metrics in seconds. (#72343, @danielqsj)
  - Add metrics `*_admission_latencies_milliseconds` and `*_admission_latencies_milliseconds_summary` for backward compatible, but will be removed in a future release.
- Pod eviction now honors graceful deletion by default if no delete options are provided in the eviction request (#72730, @liggitt)
- Update to go1.11.5 (#73326, @ixdy)
- Change proxy metrics to conform metrics guidelines. (#72334, @danielqsj)
  - The metrics `sync_proxy_rules_latency_microseconds` is deprecated, and will be removed in a future release, please convert to metrics`sync_proxy_rules_latency_seconds`.
- Add network stats for Windows nodes and pods. (#70121, @feiskyer)
- kubeadm: When certificates are present joining a new control plane make sure that they match at least the required SANs (#73093, @ereslibre)
- A new `TaintNodesByCondition` admission plugin taints newly created Node objects as "not ready", to fix a race condition that could cause pods to be scheduled on new nodes before their taints were updated to accurately reflect their reported conditions. This admission plugin is enabled by default if the `TaintNodesByCondition` feature is enabled. (#73097, @bsalamat)
- kube-addon-manager was updated to v9.0, and now uses kubectl v1.13.2 and prunes workload resources via the apps/v1 API (#72978, @liggitt)

- When a watch is closed by an HTTP2 load balancer and we are told to go away, skip printing the message to stderr by default. (#73277, @smarterclayton)
- If you are running the cloud-controller-manager and you have the `pvlabel.kubernetes.io` alpha Initializer enabled, you must now enable PersistentVolume labeling using the `PersistentVolumeLabel` admission controller instead. You can do this by adding `PersistentVolumeLabel` in the `--enable-admission-plugins` kube-apiserver flag. (#73102, @andrewsykim)
- The alpha Initializers feature, `admissionregistration.k8s.io/v1alpha1` API version, `Initializers` admission plugin, and use of the `metadata.initializers` API field have been removed. Discontinue use of the alpha feature and delete any existing `InitializerConfiguration` API objects before upgrading. The `metadata.initializers` field will be removed in a future release. (#72972, @liggitt)
- Scale max-inflight limits together with master VM sizes. (#73268, @wojtek-t)
- kubectl supports copying files with wild card (#72641, @dixudx)
- kubeadm: add back `--cert-dir` option for `kubeadm init phase certs sa` (#73239, @mattkelly)
- Remove deprecated args '–show-all' (#69255, @Pingan2017)
- As per deprecation policy in https://kubernetes.io/docs/reference/using-api/deprecation-policy/ (#73001, @shivnagarajan)
  - the taints "node.alpha.kubernetes.io/notReady" and "node.alpha.kubernetes.io/unreachable". are no
  - longer supported or adjusted. These uses should be replaced with "node.kubernetes.io/not-ready"
  - and "node.kubernetes.io/unreachable" respectively instead.
- The /swagger.json and /swagger-2.0.0.pb-v1 schema documents, deprecated since v1.10, have been removed in favor of `/openapi/v2` (#73148, @liggitt)
- CoreDNS is only officially supported on Linux at this time. As such, when kubeadm is used to deploy this component into your kubernetes cluster, it will be restricted (using nodeSelectors) to run only on nodes with that operating system. This ensures that in clusters which include Windows nodes, the scheduler will not ever attempt to place CoreDNS pods on these machines, reducing setup latency and enhancing initial cluster stability. (#69940, @MarcPow)
- kubeadm now attempts to detect an installed CRI by its usual domain socket, so that –cri-socket can be omitted from the command line if Docker is not used and there is a single CRI installed. (#69366, @rosti)
- scheduler: makes pod less racing so as to be put back into activeQ properly (#73078, @Huang-Wei)
- jsonpath expressions containing `[start:end:step]` slice are now evaluated correctly (#73149, @liggitt)
- metadata.deletionTimestamp is no longer moved into the future when is-

suing repeated DELETE requests against a resource containing a finalizer. (#73138, @liggitt)
- The "kubectl api-resources" command will no longer fail to display any resources on a single failure (#73035, @juanvallejo)
- e2e tests that require SSH may be used against clusters that have nodes without external IP addresses by setting the environment variable `KUBE_SSH_BASTION` to the `host:port` of a machine that is allowed to SSH to those nodes. The same private key that the test would use is used for the bastion host. The test connects to the bastion and then tunnels another SSH connection to the node. (#72286, @smarterclayton)
- kubeadm: explicitly wait for `etcd` to have grown when joining a new control plane (#72984, @ereslibre)
- Install CSINodeInfo and CSIDriver CRDs in the local cluster. (#72584, @xing-yang)
- kubectl loads config file once and uses persistent client config (#71117, @dixudx)
- remove stale OutOfDisk condition from kubelet side (#72507, @dixudx)
- Node OS/arch labels are promoted to GA (#73048, @yujuhong)
- Fix graceful apiserver shutdown to not drop outgoing bytes before the process terminates. (#72970, @sttts)
- Change apiserver metrics to conform metrics guidelines. (#72336, @danielqsj)
  - The following metrics are deprecated, and will be removed in a future release:
    * `apiserver_request_count`
    * `apiserver_request_latencies`
    * `apiserver_request_latencies_summary`
    * `apiserver_dropped_requests`
    * `etcd_helper_cache_hit_count`
    * `etcd_helper_cache_miss_count`
    * `etcd_helper_cache_entry_count`
    * `etcd_request_cache_get_latencies_summary`
    * `etcd_request_cache_add_latencies_summary`
    * `etcd_request_latencies_summary`
    * `transformation_latencies_microseconds`
    * `data_key_generation_latencies_microseconds`
  - Please convert to the following metrics:
    * `apiserver_request_total`
    * `apiserver_request_latency_seconds`
    * `apiserver_dropped_requests_total`
    * `etcd_helper_cache_hit_total`
    * `etcd_helper_cache_miss_total`
    * `etcd_helper_cache_entry_total`
    * `etcd_request_cache_get_latency_seconds`
    * `etcd_request_cache_add_latency_seconds`
    * `etcd_request_latency_seconds`

        * `transformation_latencies_seconds`
        * `data_key_generation_latencies_seconds`
- acquire lock before operating unschedulablepodsmap (#73022, @denkensk)
- Print `SizeLimit` of `EmptyDir` in `kubectl describe pod` outputs. (#69279, @dtaniwaki)
- add goroutine to move unschedulable pods to activeq if they are not retried for more than 1 minute (#72558, @denkensk)
- PidPressure evicts pods from lowest priority to highest priority (#72844, @dashpole)
- Reduce GCE log rotation check from 1 hour to every 5 minutes. Rotation policy is unchanged (new day starts, log file size > 100MB). (#72062, @jpbetz)
- Add support for max attach limit for Cinder (#72980, @gnufied)
- Fixes the setting of NodeAddresses when using the vSphere CloudProvider and nodes that have multiple IP addresses. (#70805, @danwinship)
- kubeadm: pull images when joining a new control plane instance (#72870, @MalloZup)
- Enable mTLS encription between etcd and kube-apiserver in GCE (#70144, @wenjiaswe)
- The `/swaggerapi/*` schema docs, deprecated since 1.7, have been removed in favor of the /openapi/v2 schema docs. (#72924, @liggitt)
- Allow users to use Docker 18.09 with kubeadm (#72823, @dims)

v1.14.0-alpha.1

Documentation

Downloads for v1.14.0-alpha.1

filename	sha512 hash
kubernetes.tar.gz	`fac80e5674e547d00987516fb2eca6ea9947529307566be6a12932e3c9e430e8ad094afa`
kubernetes-src.tar.gz	`d1b5b2c15cb0daa076606f4ccf887724b0166dee0320f2a61d16ab4689931ab0cf5dac4c`

Client Binaries

filename	sha512 hash
kubernetes-client-darwin-386.tar.gz	`307c426e4abaf81648af393ddd641c225d87b02d8662d1309fe3528f`
kubernetes-client-darwin-amd64.tar.gz	`8daa85f3e8feaea0d55f20f850038dd113f0f08b62eef944b08a9109`
kubernetes-client-linux-386.tar.gz	`28d73c299cb9859fdfeb3e4869a7a9c77f5679309c2613bd2c72d92d`

filename	sha512 hash
kubernetes-client-linux-amd64.tar.gz	eb923e13026f80b743a57100d4f94995f322ab6f107c34ffd9aa74b5
kubernetes-client-linux-arm.tar.gz	279b0d0c560900021abea4bbfc25aeca7389f0b37d80022dc3335147
kubernetes-client-linux-arm64.tar.gz	d69d28361b9c9e16f3e6804ccda92d55ee743e63aba7fded04edf1f7
kubernetes-client-linux-ppc64le.tar.gz	ca6ebb87df98bf179c94f54a4e8ae2ef2ea534b1bc5014331f937aa9
kubernetes-client-linux-s390x.tar.gz	13fa2058ceba66d8da5ba5982aa302cdd1c61d15253183ab97739229
kubernetes-client-windows-386.tar.gz	42ba4bba477e2958aab674a0fbf888bd5401fa5fbc39466b6cad0fc9
kubernetes-client-windows-amd64.tar.gz	d5f339fe4d37c61babc97208446d1859423b7679f34040f72e9138b7

Server Binaries

filename	sha512 hash
kubernetes-server-linux-amd64.tar.gz	bcbcbd3ac4419e54e894d1e595f883e61fcf9db0353a30d794a9e5030c
kubernetes-server-linux-arm.tar.gz	fda4ea9168555f724659601b06737dea6ec95574569df4ef7e4ab6c2cc
kubernetes-server-linux-arm64.tar.gz	c142857711ec698844cd61188e70b5ab185ba2c8828cf5563a2f429584
kubernetes-server-linux-ppc64le.tar.gz	524a40c5717b24c5a3b2491c4c61cf3038ba5ae7f343797a1b56a5906d
kubernetes-server-linux-s390x.tar.gz	ef943fe326b05ece57f2e409ab1cc5fe863f5effa591abae17181c84a5

Node Binaries

filename	sha512 hash
kubernetes-node-linux-amd64.tar.gz	396f7588e9131dd1b99d101c8bb94fb7e67ab067327ee58dab5a6e248
kubernetes-node-linux-arm.tar.gz	b75c1550438da0b66582d6de90436ee3c44e41e67f74947d93ee9a07e
kubernetes-node-linux-arm64.tar.gz	b6c46f9250b5565fa178ecc99ffedc6724b0bfffb73acc7d3da2c678a
kubernetes-node-linux-ppc64le.tar.gz	8d505c61a59bc9fc53d6f219d6434ddd962ba383654c46e16d413cee0
kubernetes-node-linux-s390x.tar.gz	83b6cf0fb348faa93fa40ec2a947b202b3a5a2081c3896ae39618f947
kubernetes-node-windows-amd64.tar.gz	43471680533685c534023787cd40431b67041bab43e93dea457283ee0

Changelog since v1.13.0

Action Required

- action required (#68753, @johnSchnake)
  - If you are running E2E tests which require SSH keys and you utilize environment variables to override their location, you may need to modify the environment variable set. On all providers the environment variable override can now be either an absolute path to the key or a relative path (relative to ~/.ssh). Specifically the changes are:
  - - Created new GCE_SSH_KEY allowing specification of SSH keys for gce, gke, and kubemark.

  - - AWS_SSH_KEY, previously assumed to be an absolute path can now be either relative or absolute
  - - LOCAL_SSH_KEY (for local and vsphere providers) was previously assumed to be a filename relative to ~/.ssh but can now also be an absolute path
  - - KUBE_SSH_KEY (for skeleton provider) was previously assumed to be a filename relative to ~/.ssh but can now also be an absolute path

Other notable changes

- Connections from Pods to Services with 0 endpoints will now ICMP reject immediately, rather than blackhole and timeout. (#72534, @thockin)
- Improve efficiency of preemption logic in clusters with many pending pods. (#72895, @bsalamat)
- Change scheduler metrics to conform metrics guidelines. (#72332, @danielqsj)
  - The following metrics are deprecated, and will be removed in a future release:
    * `e2e_scheduling_latency_microseconds`
    * `scheduling_algorithm_latency_microseconds`
    * `scheduling_algorithm_predicate_evaluation`
    * `scheduling_algorithm_priority_evaluation`
    * `scheduling_algorithm_preemption_evaluation`
    * `binding_latency_microseconds`
  - Please convert to the following metrics:
    * `e2e_scheduling_latency_seconds`
    * `scheduling_algorithm_latency_seconds`
    * `scheduling_algorithm_predicate_evaluation_seconds`
    * `scheduling_algorithm_priority_evaluation_seconds`
    * `scheduling_algorithm_preemption_evaluation_seconds`
    * `binding_latency_seconds`
- Fix SelectorSpreadPriority scheduler to match all selectors when distributing pods. (#72801, @Ramyak)
- Add bootstrap service account & cluster roles for node-lifecycle-controller, cloud-node-lifecycle-controller, and cloud-node-controller. (#72764, @andrewsykim)
- Fixes spurious 0-length API responses. (#72856, @liggitt)
- Updates Fluentd to 1.3.2 & added filter_parser (#71180, @monotek)
- The leaderelection package allows the lease holder to release its lease when the calling context is cancelled. This allows (#71490, @smarterclayton)
  - faster handoff when a leader-elected process is gracefully terminated.
- Make volume binder resilient to races between main schedule loop and async binding operation (#72045, @cofyc)
- Bump minimum docker API version to 1.26 (1.13.1) (#72831, @yujuhong)

- If the `TokenRequestProjection` feature gate is disabled, projected serviceAccountToken volume sources are now dropped at object creation time, or at object update time if the existing object did not have a projected serviceAccountToken volume source. Previously, these would result in validation errors. (#72714, @mourya007)
- Add `metrics-port` to kube-proxy cmd flags. (#72682, @whypro)
- kubectl: fixed an issue with "too old resource version" errors continuously appearing when calling `kubectl delete` (#72825, @liggitt)
- [Breaking change, client-go]: The WaitFor function returns, probably an ErrWaitTimeout, when the done channel is closed, even if the `WaitFunc` doesn't handle the done channel. (#72364, @kdada)
- removes newline from json output for windows nodes #72657 (#72659, @jsturtevant)
- The DenyEscalatingExec and DenyExecOnPrivileged admission plugins are deprecated and will be removed in v1.18. Use of `PodSecurityPolicy` or a custom admission plugin to limit creation of pods is recommended instead. (#72737, @liggitt)
- Fix `describe statefulset` not printing number of desired replicas correctly (#72781, @tghartland)
- Fix kube-proxy PodSecurityPolicy binding on GCE & GKE. This was only an issue when running kube-proxy as a DaemonSet, with PodSecurityPolicy enabled. (#72761, @tallclair)
- Drops `status.Conditions` of new `PersistentVolume` objects if it was not set on the old object during `PrepareForUpdate`. (#72739, @rajathagasthya)
- kubelet: fixes cadvisor internal error when "–container-runtime-endpoint" is set to "unix:///var/run/crio/crio.sock". (#72340, @makocchi-git)
- The `spec.SecurityContext.Sysctls` field is now dropped during creation of `Pod` objects unless the `Sysctls` feature gate is enabled. (#72752, @rajathagasthya)
  - The `spec.AllowedUnsafeSysctls` and `spec.ForbiddenSysctls` fields are now dropped during creation of `PodSecurityPolicy` objects unless the `Sysctls` feature gate is enabled.
- kubeadm: fixed storing of front-proxy certificate in secrets required by kube-controller-manager selfhosting pivoting (#72727, @bart0sh)
- Administrator is able to configure max pids for a pod on a node. (#72076, @derekwaynecarr)
- Move users of `factory.NewConfigFactory` to `scheduler.New`. (#71875, @wgliang)
- The `spec.SecurityContext.ShareProcessNamespace` field is now dropped during creation of `Pod` objects unless the `PodShareProcessNamespace` feature gate is enabled. (#72698, @rajathagasthya)
- kube-apiserver: When configuring integration with external KMS Providers, users can supply timeout value (i.e. how long should kube-apiserver wait before giving up on a call to KMS). (@immutableT) (#72540, @immutableT)

- The `spec.readinessGates` field is now dropped during creation of `Pod` objects unless the `PodReadinessGates` feature gate is enabled. (#72695, @rajathagasthya)
- The `spec.dataSource` field is now dropped during creation of PersistentVolumeClaim objects unless the `VolumeSnapshotDataSource` feature gate is enabled. (#72666, @rajathagasthya)
- Stop kubelet logging a warning to override hostname if there's no change detected. (#71560, @KashifSaadat)
- client-go: fake clients now properly return NotFound errors when attempting to patch non-existent objects (#70886, @bouk)
- kubectl: fixes a bug determining the correct namespace while running in a pod when the `--context` flag is explicitly specified, and the referenced context specifies the namespace `default` (#72529, @liggitt)
- Fix scheduling starvation of pods in cluster with large number of unschedulable pods. (#72619, @everpeace)
- If the AppArmor feature gate is disabled, AppArmor-specific annotations in pod and pod templates are dropped when the object is created, and during update of objects that do not already contain AppArmor annotations, rather than triggering a validation error. (#72655, @liggitt)
- client-go: shortens refresh period for token files to 1 minute to ensure auto-rotated projected service account tokens are read frequently enough. (#72437, @liggitt)
- Multiple tests which previously failed due to lack of external IP addresses defined on the nodes should now be passable. (#68792, @johnSchnake)
- kubeadm: fixed incorrect controller manager pod mutations during self-hosting pivoting (#72518, @bart0sh)
- Increase Azure default maximumLoadBalancerRuleCount to 250. (#72621, @feiskyer)
- RuntimeClass is now printed with extra `RUNTIME-HANDLER` column. (#72446, @Huang-Wei)
- Updates the kubernetes dashboard add-on to v1.10.1. Skipping dashboard login is no longer enabled by default. (#72495, @liggitt)
- [GCP] Remove confusing error log entry form fluentd scalers. (#72243, @cezarygerard)
- change azure disk host cache to ReadOnly by default (#72229, @andyzhangx)
- Nodes deleted in the cloud provider with Ready condition `Unknown` should also be deleted on the API server. (#72559, @andrewsykim)
- `kubectl apply --prune` now uses the apps/v1 API to prune workload resources (#72352, @liggitt)
- Fixes a bug in HPA controller so HPAs are always updated every resyncPeriod (15 seconds). (#72373, @krzysztof-jastrzebski)
- IPVS: "ExternalTrafficPolicy: Local" now works with LoadBalancer services using loadBalancerIP (#72432, @lbernail)
- Fixes issue with cleaning up stale NFS subpath mounts (#71804, @msau42)

- Modify the scheduling result struct and improve logging for successful binding. (#71926, @wgliang)
- Run one etcd storage compaction per default interval of 5min. Do not run one for each resource and each CRD. This fixes the compaction log spam and reduces load on etcd. (#68557, @sttts)
- kube-apiserver: `--runtime-config` can once again be used to enable/disable serving specific resources in the `extensions/v1beta1` API group. Note that specific resource enablement/disablement is only allowed for the `extensions/v1beta1` API group for legacy reasons. Attempts to enable/disable individual resources in other API groups will print a warning, and will return an error in future releases. (#72249, @liggitt)
- kubeadm: fixed storing of etcd certificates in secrets required by kube-apiserver selfhosting pivoting (#72478, @bart0sh)
- kubeadm: remove the deprecated "–address" flag for controller-manager and scheduler. (#71973, @MalloZup)
- kube-apiserver: improves performance of requests made with service account token authentication (#71816, @liggitt)
- Use prometheus conventions for workqueue metrics. (#71300, @danielqsj)
  - It is now deprecated to use the following metrics:
    - `{WorkQueueName}_depth`
    - `{WorkQueueName}_adds`
    - `{WorkQueueName}_queue_latency`
    - `{WorkQueueName}_work_duration`
    - `{WorkQueueName}_unfinished_work_seconds`
    - `{WorkQueueName}_longest_running_processor_microseconds`
    - `{WorkQueueName}_retries`
  - Please convert to the following metrics:
    - `workqueue_depth`
    - `workqueue_adds_total`
    - `workqueue_queue_latency_seconds`
    - `workqueue_work_duration_seconds`
    - `workqueue_unfinished_work_seconds`
    - `workqueue_longest_running_processor_seconds`
    - `workqueue_retries_total`
- Fix inability to use k8s with dockerd having default IPC mode set to private. (#70826, @kolyshkin)
- Fix a race condition in the scheduler preemption logic that could cause nominatedNodeName of a pod not to be considered in one or more scheduling cycles. (#72259, @bsalamat)
- Fix registration for scheduling framework plugins with the default plugin set (#72396, @y-taka-23)
- The GA VolumeScheduling feature gate can no longer be disabled and will be removed in a future release (#72382, @liggitt)
- Fix race condition introduced by graceful termination which can lead to a deadlock in kube-proxy (#72361, @lbernail)

- Fixes issue where subpath volume content was deleted during orphaned pod cleanup for Local volumes that are directories (and not mount points) on the root filesystem. (#72291, @msau42)
- Fixes `kubectl create secret docker-registry` compatibility (#72344, @liggitt)
- Add-on manifests now use the apps/v1 API for DaemonSets, Deployments, and ReplicaSets (#72203, @liggitt)
- "kubectl wait" command now supports the "–all" flag to select all resources in the namespace of the specified resource types. (#70599, @caesarxuchao)
- `deployments/rollback` is now passed through validation/admission controllers (#72271, @jhrv)
- The `Lease` API type in the `coordination.k8s.io` API group is promoted to `v1` (#72239, @wojtek-t)
- Move compatibility_test.go to pkg/scheduler/api (#72014, @huynq0911)
- New Azure cloud provider option 'cloudProviderBackoffMode' has been added to reduce Azure API retries. Candidate values are: (#70866, @feiskyer) * default (or empty string): keep same with before. * v2: only backoff retry with Azure SDK with fixed exponent 2.
- Set percentage of nodes scored in each cycle dynamically based on the cluster size. (#72140, @wgliang)
- Fix AAD support for Azure sovereign cloud in kubectl (#72143, @karataliu)
- Make kube-proxy service abstraction optional. (#71355, @bradhoekstra)
  - Add the 'service.kubernetes.io/service-proxy-name' label to a Service to disable the kube-proxy service proxy implementation.
- kubectl: `-A` can now be used as a shortcut for `--all-namespaces` (#72006, @soltysh)
- discovery.CachedDiscoveryInterface implementation returned by NewMemCacheClient has changed semantics of Invalidate method – the cache refresh is now deferred to the first cache lookup. (#70994, @mborsz)
- Fix device mountable volume names in DSW to prevent races in device mountable plugin, e.g. local. (#71509, @cofyc)
- Enable customize in kubectl: kubectl will be able to recognize directories with kustomization.YAML (#70875, @Liujingfang1)
- Stably sort controllerrevisions. This can prevent pods of statefulsets from continually rolling. (#66882, @ryanmcnamara)
- Update to use go1.11.4. (#72084, @ixdy)
- fixes an issue deleting pods containing subpath volume mounts with the VolumeSubpath feature disabled (#70490, @liggitt)
- Clean up old eclass code (#71399, @resouer)
- Fix a race condition in which kubeadm only waits for the kubelets kubeconfig file when it has performed the TLS bootstrap, but wasn't waiting for certificates to be present in the filesystem (#72030, @ereslibre)
- In addition to restricting GCE metadata requests to known APIs,

the metadata-proxy now restricts query strings to known parameters. (#71094, @dekkagaijin)
- kubeadm: fix a possible panic when joining a new control plane node in HA scenarios (#72123, @anitgandhi)
- fix race condition when attach azure disk in vmss (#71992, @andyzhangx)
- Update to use go1.11.3 with fix for CVE-2018-16875 (#72035, @seemethere)
- kubeadm: fix a bug when syncing etcd endpoints (#71945, @pytimer)
- fix kubelet log flushing issue in azure disk (#71990, @andyzhangx)
- Disable proxy to loopback and linklocal (#71980, @micahhausler)
- Fix overlapping filenames in diff if multiple resources have the same name. (#71923, @apelisse)
- fix issue: vm sku restriction policy does not work in azure disk attach/detach (#71941, @andyzhangx)
- kubeadm: Create /var/lib/etcd with correct permissions (0700) by default. (#71885, @dims)
- Scheduler only activates unschedulable pods if node's scheduling related properties change. (#71551, @mlmhl)
- kube-proxy in IPVS mode will stop initiating connections to terminating pods for services with sessionAffinity set. (#71834, @lbernail)
- kubeadm: improve hostport parsing error messages (#71258, @bart0sh)
- Support graceful termination with IPVS when deleting a service (#71895, @lbernail)
- Include CRD for BGPConfigurations, needed for calico 2.x to 3.x upgrade. (#71868, @satyasm)
- apply: fix detection of non-dry-run enabled servers (#71854, @apelisse)
- Clear UDP conntrack entry on endpoint changes when using nodeport (#71573, @JacobTanenbaum)
- Add successful and failed history limits to cronjob describe (#71844, @soltysh)
- kube-controller-manager: fixed issue display help for the deprecated insecure –port flag (#71601, @liggitt)
- kubectl: fixes regression in –sort-by behavior (#71805, @liggitt)
- Fixes pod deletion when cleaning old cronjobs (#71801, @soltysh)
- kubeadm: use kubeconfig flag instead of kubeconfig-dir on init phase bootstrap-token (#71803, @yagonobre)
- kube-scheduler: restores ability to run without authentication configuration lookup permissions (#71755, @liggitt)
- Add aggregator_unavailable*apiservice*{count,gauge} metrics in the kube-aggregator. (#71380, @sttts)
- Fixes apiserver nil pointer panics when requesting v2beta1 autoscaling object metrics (#71744, @yue9944882)
- Only use the first IP address got from instance metadata. This is because Azure CNI would set up a list of IP addresses in instance metadata, while only the first one is the Node's IP. (#71736, @feiskyer)
- client-go: restores behavior of populating the BearerToken field in

rest.Config objects constructed from kubeconfig files containing tokenFile config, or from in-cluster configuration. An additional BearerTokenFile field is now populated to enable constructed clients to periodically refresh tokens. (#71713, @liggitt)

- kubeadm: remove deprecated kubeadm config print-defaults command (#71467, @rosti)
- hack/local-up-cluster.sh now enables kubelet authentication/authorization by default (they can be disabled with KUBELET_AUTHENTICATION_WEBHOOK=false and KUBELET_AUTHORIZATION_WEBHOOK=false (#71690, @liggitt)
- Fixes an issue where Azure VMSS instances not existing in Azure were not being deleted by the Cloud Controller Manager. (#71597, @marcsensenich)
- kubeadm reset correcty unmounts mount points inside /var/lib/kubelet (#71663, @bart0sh)
- Upgrade default etcd server to 3.3.10 (#71615, @jpbetz)
- When creating a service with annotation: service.beta.kubernetes.io/load-balancer-source-ranges containing multiple source ranges and service.beta.kubernetes.io/azure-shared-securityrule: "false", the NSG rules will be collapsed. (#71484, @ritazh)
- disable node's proxy use of http probe (#68663, @WanLinghao)
- Bumps version of kubernetes-cni to 0.6.0 (#71629, @mauilion)
- On GCI, NPD starts to monitor kubelet, docker, containerd crashlooping, read-only filesystem and corrupt docker overlay2 issues. (#71522, @wangzhen127)
- When a kubelet is using –bootstrap-kubeconfig and certificate rotation, it no longer waits for bootstrap to succeed before launching static pods. (#71174, @smarterclayton)
- Add an plugin interfaces for "reserve" and "prebind" extension points of the scheduling framework. (#70227, @bsalamat)
- Fix scheduling starvation of pods in cluster with large number of unschedulable pods. (#71488, @bsalamat)
- Reduce CSI log and event spam. (#71581, @saad-ali)
- Add conntrack as a dependency of kubelet and kubeadm when building rpms and debs. Both require conntrack to handle cleanup of connections. (#71540, @mauilion)
- UDP connections now support graceful termination in IPVS mode (#71515, @lbernail)
- Log etcd client errors. The verbosity is set with the usual `-v` flag. (#71318, @sttts)
- The `DefaultFeatureGate` package variable now only exposes read-only feature gate methods. Methods for mutating feature gates have moved into a `MutableFeatureGate` interface and are accessible via the `DefaultMutableFeatureGate` package variable. Only top-level commands and options setup should access `DefaultMutableFeatureGate`. (#71302, @liggitt)

- `node.kubernetes.io/pid-pressure` toleration is added for DaemonSet pods, and `node.kubernetes.io/out-of-disk` isn't added any more even if it's a critical pod. (#67036, @Huang-Wei)
- Update k8s.io/utils to allow for asynchronous process control (#71047, @hoegaarden)
- Fixes possible panic during volume detach, if corresponding volume plugin became non-attachable (#71471, @mshaverdo)
- Fix cloud-controller-manager crash when using AWS provider and PersistentVolume initializing controller (#70432, @mvladev)
- Fixes an issue where Portworx volumes cannot be mounted if 9001 port is already in use on the host and users remap 9001 to another port. (#70392, @harsh-px)
- Fix `SubPath` printing of `VolumeMounts`. (#70127, @dtaniwaki)
- Fixes incorrect paths (missing first letter) when copying files from pods to (#69885, @clickyotomy)
  - local in 'kubectl cp'.
- Fix AWS NLB security group updates where valid security group ports were incorrectly removed (#68422, @kellycampbell)
  - when updating a service or when node changes occur.

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, 2019 at 8:19 AM PST by Fixed minor issues in Windows section (#13451) (Page History)

Edit This Page

Set up a High Availability etcd cluster with kubeadm

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

- Before you begin
- Setting up the cluster
- What's next

Before you begin

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

Setting up the cluster

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

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

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

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

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

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

2. Create configuration files for kubeadm.

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

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

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

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

for i in "${!ETCDHOSTS[@]}"; do
HOST=${ETCDHOSTS[$i]}
NAME=${NAMES[$i]}
cat << EOF > /tmp/${HOST}/kubeadmcfg.yaml
apiVersion: "kubeadm.k8s.io/v1beta1"
kind: ClusterConfiguration
etcd:
    local:
        serverCertSANs:
        - "${HOST}"
        peerCertSANs:
        - "${HOST}"
        extraArgs:
            initial-cluster: ${NAMES[0]}=https://${ETCDHOSTS[0]}:2380,${NAMES[1]}=https
            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}
   kubeadmcfg.yaml
---
/etc/kubernetes/pki
   apiserver-etcd-client.crt
   apiserver-etcd-client.key
   etcd
        ca.crt
        ca.key
        healthcheck-client.crt
        healthcheck-client.key
        peer.crt
        peer.key
        server.crt
        server.key
```

   On `$HOST1`:

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

   On `$HOST2`

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

```
peer.crt
peer.key
server.crt
server.key
```

7. Create the static pod manifests

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

   ```
   root@HOST0 $ kubeadm init phase etcd local --config=/tmp/${HOST0}/kubeadmcfg.yaml
   root@HOST1 $ kubeadm init phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
   root@HOST2 $ kubeadm init phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
   ```

8. Optional: Check the cluster health

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

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

What's next

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

Feedback

Was this page helpful?

Yes

No

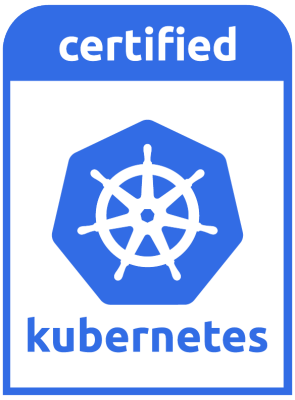
Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page

Page last modified on February 11, 2019 at 1:35 AM PST by docs: fix typo (#12568) (Page History)

Edit This Page

Installing kubeadm



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

Before you begin

- One or more machines running one of:
  - Ubuntu 16.04+
  - Debian 9
  - CentOS 7
  - RHEL 7
  - Fedora 25/26 (best-effort)

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

Verify the MAC address and product_uuid are unique for every node

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

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

Check network adapters

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

Check required ports

Master node(s)

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

Worker node(s)

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

** Default port range for NodePort Services.

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

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

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

Installing runtime

Since v1.6.0, Kubernetes has enabled the use of CRI, Container Runtime Interface, by default.

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

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

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

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

If the container runtime of choice is Docker, it is used through the built-in `dockershim` CRI implementation inside of the `kubelet`.

Other CRI-based runtimes include:

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

Refer to the CRI installation instructions for more information.

Installing kubeadm, kubelet and kubectl

You will install these packages on all of your machines:

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

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

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

For more information on version skews, see:

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

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

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

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
```

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

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

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

systemctl enable --now kubelet
```

Note:

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

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

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

Install CNI plugins (required for most pod network):

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

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

```
CRICTL_VERSION="v1.11.1"
mkdir -p /opt/bin
```

```
curl -L "https://github.com/kubernetes-incubator/cri-tools/releases/download/${CRICTL_VERSIO
```

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

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

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

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

Enable and start `kubelet`:

```
systemctl enable --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 Master Node

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

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

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

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

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

Restarting the kubelet is required:

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

Troubleshooting

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

What's next

- Using kubeadm to Create a Cluster

Feedback

Was this page helpful?

Yes

No

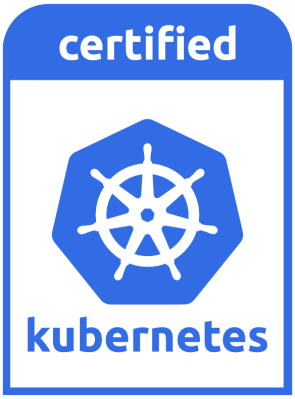
Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page

Page last modified on April 01, 2019 at 12:56 PM PST by Update install-kubeadm.md (#13514) (Page History)

Edit This Page

Creating a single master cluster with kubeadm



kubeadm helps you bootstrap a minimum viable Kubernetes cluster that conforms to best practices. With kubeadm, your cluster should pass Kubernetes Conformance tests. Kubeadm also supports other cluster lifecycle functions, such as upgrades, downgrade, and managing bootstrap tokens.

Because you can install kubeadm on various types of machine (e.g. laptop, server, Raspberry Pi, etc.), it's well suited for integration with provisioning systems such as Terraform or Ansible.

kubeadm's simplicity means it can serve a wide range of use cases:

- New users can start with kubeadm to try Kubernetes out for the first time.
- Users familiar with Kubernetes can spin up clusters with kubeadm and test their applications.
- Larger projects can include kubeadm as a building block in a more complex system that can also include other installer tools.

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

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

kubeadm Maturity

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

kubeadm's overall feature state is **GA**. Some sub-features, like the configuration file API are still under active development. The implementation of creating the cluster may change slightly as the tool evolves, but the overall implementation should be pretty stable. Any commands under `kubeadm alpha` are by definition, supported on an alpha level.

Support timeframes

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

Kubernetes version	Release month	End-of-life-month
v1.6.x	March 2017	December 2017
v1.7.x	June 2017	March 2018
v1.8.x	September 2017	June 2018
v1.9.x	December 2017	September 2018
v1.10.x	March 2018	December 2018
v1.11.x	June 2018	March 2019
v1.12.x	September 2018	June 2019
v1.13.x	December 2018	September 2019

- Before you begin
- Objectives
- Instructions
- Tear down
- Maintaining a cluster
- Explore other add-ons
- What's next
- Feedback
- Version skew policy
- kubeadm works on multiple platforms
- Limitations
- Troubleshooting

Before you begin

- One or more machines running a deb/rpm-compatible OS, for example Ubuntu or CentOS
- 2 GB or more of RAM per machine. Any less leaves little room for your apps.
- 2 CPUs or more on the master
- Full network connectivity among all machines in the cluster. A public or private network is fine.

Objectives

- Install a single master Kubernetes cluster or high availability cluster

- Install a Pod network on the cluster so that your Pods can talk to each other

Instructions

Installing kubeadm on your hosts

See "Installing kubeadm".

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

Initializing your master

The master is the machine where the control plane components run, including etcd (the cluster database) and the API server (which the kubectl CLI communicates with).

1. Choose a pod network add-on, and verify whether it requires any arguments to be passed to kubeadm initialization. Depending on which third-party provider you choose, you might need to set the `--pod-network-cidr` to a provider-specific value. See Installing a pod network add-on.
2. (Optional) Since version 1.14, kubeadm will try to detect the container runtime on Linux by using a list of well known domain socket paths. To use different container runtime or if there are more than one installed on the provisioned node, specify the `--cri-socket` argument to `kubeadm init`. See Installing runtime.
3. (Optional) Unless otherwise specified, kubeadm uses the network interface associated with the default gateway to advertise the master's IP. To use a different network interface, specify the `--apiserver-advertise-address=<ip-address>` argument to `kubeadm init`. To deploy an IPv6 Kubernetes cluster using IPv6 addressing, you must specify an IPv6 address, for example `--apiserver-advertise-address=fd00::101`
4. (Optional) Run `kubeadm config images pull` prior to `kubeadm init` to verify connectivity to gcr.io registries.

Now run:

```
kubeadm init <args>
```

More information

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

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

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

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

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

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

```
[init] Using Kubernetes version: vX.Y.Z
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connect
[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/kubead
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [kubeadm-master localhost] and IPs
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [kubeadm-master localhost] and IPs [1
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubeadm-master kubernetes kubernetes
[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 fro
[apiclient] All control plane components are healthy after 31.501735 seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-sys
[kubelet] Creating a ConfigMap "kubelet-config-X.Y" in namespace kube-system with the config
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API
[mark-control-plane] Marking the node kubeadm-master as control-plane by adding the label "n
[mark-control-plane] Marking the node kubeadm-master as control-plane by adding the taints
[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
[bootstraptoken] configured RBAC rules to allow the csrapprover controller automatically app
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client cer
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes master has initialized successfully!

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

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

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

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

  kubeadm join <master-ip>:<master-port> --token <token> --discovery-token-ca-cert-hash sha2
```

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

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

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

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

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

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

Installing a pod network add-on

> **Caution:** This section contains important information about installation and deployment order. Read it carefully before proceeding.

You must install a pod network add-on so that your pods can communicate with each other.

The network must be deployed before any applications. Also, CoreDNS will not start up before a network is installed. kubeadm only supports Container Network Interface (CNI) based networks (and does not support kubenet).

Several projects provide Kubernetes pod networks using CNI, some of which also support Network Policy. See the add-ons page for a complete list of available network add-ons. - IPv6 support was added in CNI v0.6.0. - CNI bridge and local-ipam are the only supported IPv6 network plugins in Kubernetes version 1.9.

Note that kubeadm sets up a more secure cluster by default and enforces use of RBAC. Make sure that your network manifest supports RBAC.

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

You can install a pod network add-on with the following command:

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

You can install only one pod network per cluster.

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

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

For more information about using Calico, see Quickstart for Calico on Kubernetes, Installing Calico for policy and networking, and other related resources.

For Calico to work correctly, you need to pass `--pod-network-cidr=192.168.0.0/16` to `kubeadm init` or update the `calico.yml` file to match your Pod network. Note that Calico works on `amd64`, `arm64`, and `ppc64le` only.

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

Canal uses Calico for policy and Flannel for networking. Refer to the Calico documentation for the official getting started guide.

For Canal to work correctly, `--pod-network-cidr=10.244.0.0/16` has to be passed to `kubeadm init`. Note that Canal works on `amd64` only.

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

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

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

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

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

To deploy Cilium you just need to run:

```
kubectl create -f https://raw.githubusercontent.com/cilium/cilium/v1.4/examples/kubernetes/1
```

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

```
$ kubectl get pods -n kube-system --selector=k8s-app=cilium
NAME           READY   STATUS    RESTARTS   AGE
cilium-drxkl   1/1     Running   0          18m
```

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

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

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

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

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/a70459be0084506e4ec919aa1c
```

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

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

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

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

For information on setting up Kubernetes cluster with Kube-router using kubeadm, please see official setup guide.

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

The official Romana set-up guide is here.

Romana works on `amd64` only.

```
kubectl apply -f https://raw.githubusercontent.com/romana/romana/master/containerize/specs/r
```

Set `/proc/sys/net/bridge/bridge-nf-call-iptables` to `1` by running `sysctl net.bridge.bridge-nf-call-iptables=1` to pass bridged IPv4 traffic to iptables' chains. This is a requirement for some CNI plugins to work, for more information please see here.

The official Weave Net set-up guide is here.

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

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 |
```

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

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

Kindly refer to this quickstart: TungstenFabric

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

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

If your network is not working or CoreDNS is not in the Running state, check out our troubleshooting docs.

Control plane node isolation

By default, your cluster will not schedule pods on the master for security reasons. If you want to be able to schedule pods on the master, e.g. for a single-machine Kubernetes cluster for development, run:

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

With output looking something like:

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

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

Joining your nodes

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

- SSH to the machine
- Become root (e.g. `sudo su -`)
- Run the command that was output by `kubeadm init`. For example:

```
kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256
```

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

```
kubeadm token list
```

The output is similar to this:

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

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

```
kubeadm token create
```

The output is similar to this:

```
5didvk.d09sbcov8ph2amjw
```

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

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

The output is similar to this:

```
8cb2de97839780a412b93877f8507ad6c94f73add17d5d7058e91741c9d5ec78
```

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

The output should look something like:

```
[preflight] Running pre-flight checks

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

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

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

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

(Optional) Controlling your cluster from machines other than the master

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

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

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

(Optional) Proxying API Server to localhost

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

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

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

Tear down

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

Talking to the master with the appropriate credentials, run:

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

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

```
kubeadm reset
```

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

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

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

```
ipvsadm -C
```

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

More options and information about the `kubeadm reset command`.

Maintaining a cluster

Instructions for maintaining kubeadm clusters (e.g. upgrades,downgrades, etc.) can be found here.

Explore other add-ons

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

What's next

- Verify that your cluster is running properly with Sonobuoy
- Learn about kubeadm's advanced usage in the kubeadm reference documentation
- Learn more about Kubernetes concepts and `kubectl`.
- Configure log rotation. You can use **logrotate** for that. When using Docker, you can specify log rotation options for Docker daemon, for example `--log-driver=json-file --log-opt=max-size=10m`

`--log-opt=max-file=5`. See Configure and troubleshoot the Docker daemon for more details.

Feedback

- For bugs, visit kubeadm Github issue tracker
- For support, visit kubeadm Slack Channel: #kubeadm
- General SIG Cluster Lifecycle Development Slack Channel: #sig-cluster-lifecycle
- SIG Cluster Lifecycle SIG information
- SIG Cluster Lifecycle Mailing List: kubernetes-sig-cluster-lifecycle

Version skew policy

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

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

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

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

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

kubeadm works on multiple platforms

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

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

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

Limitations

Please note: kubeadm is a work in progress and these limitations will be addressed in due course.

1. The cluster created here has a single master, with a single etcd database running on it. This means that if the master fails, your cluster may lose data and may need to be recreated from scratch. Adding HA support (multiple etcd servers, multiple API servers, etc) to kubeadm is still a work-in-progress.

Workaround: regularly back up etcd. The etcd data directory configured by kubeadm is at `/var/lib/etcd` on the master.

Troubleshooting

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

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 April 15, 2019 at 6:12 AM PST by Flannel will not work unless UDP ports 8285 and 8472 are allowed for all hosts (#13789) (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.

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

- APIServer flags
- ControllerManager flags
- Scheduler flags

APIServer flags

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

Example usage:

```
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
kubernetesVersion: v1.13.0
metadata:
  name: 1.13-sample
apiServer:
  extraArgs:
    advertise-address: 192.168.0.103
    anonymous-auth: false
    enable-admission-plugins: AlwaysPullImages,DefaultStorageClass
```

```
    audit-log-path: /home/johndoe/audit.log
```

ControllerManager flags

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

Example usage:

```
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
kubernetesVersion: v1.13.0
metadata:
  name: 1.13-sample
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/v1beta1
kind: ClusterConfiguration
kubernetesVersion: v1.13.0
metadata:
  name: 1.13-sample
scheduler:
  extraArgs:
    address: 0.0.0.0
    config: /home/johndoe/schedconfig.yaml
    kubeconfig: /home/johndoe/kubeconfig.yaml
```

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on January 08, 2019 at 9:30 PM PST by adding feature state snip (#12134) (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.

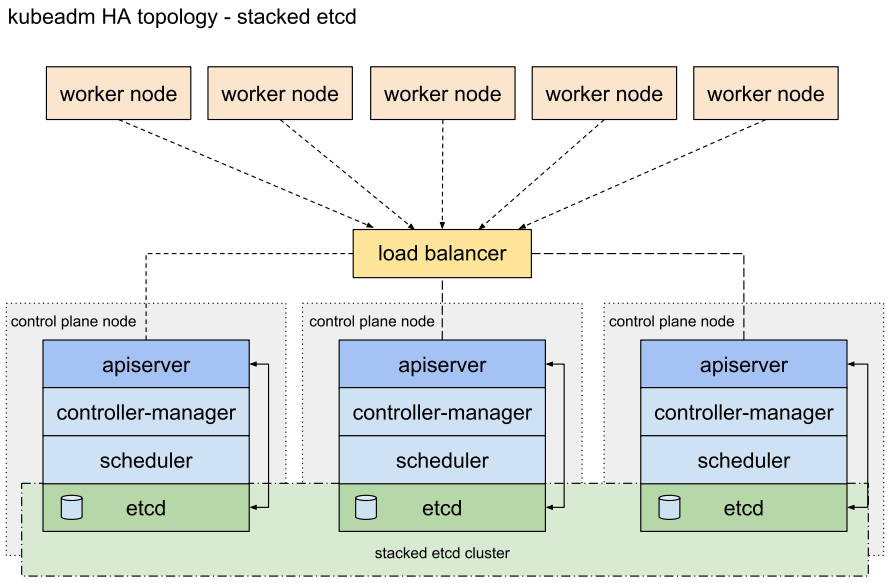
Each control plane node creates a local etcd member and this etcd member communicate only with the `kube-apiserver` of this node. The same applies to the local `kube-controller-manager` and `kube-scheduler` instances.

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 --experimental-control-plane`.



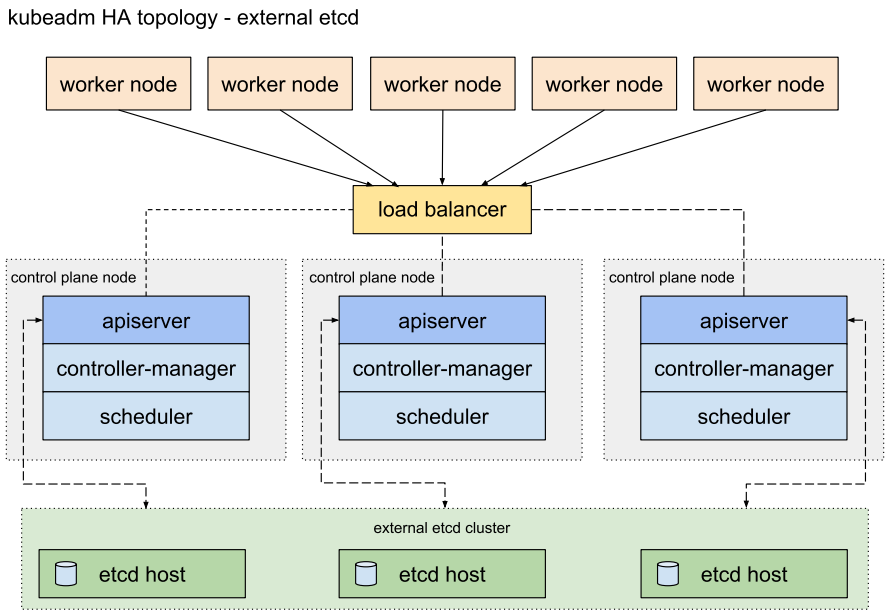
External etcd topology

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

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

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

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



What's next

- Set up a highly available cluster with kubeadm

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on December 03, 2018 at 7:21 PM PST by Official 1.13 Release Docs (#11401) (Page History)

Edit This Page

Creating Highly Available Clusters with kubeadm

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

- With stacked 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.

You should also be aware that setting up HA clusters with kubeadm is still experimental and will be further simplified in future versions. You might encounter issues with upgrading your clusters, for example. We encourage you to try either approach, and provide us with feedback in the kubeadm issue tracker.

See also The upgrade documentation.

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

Before you begin

For both methods you need this infrastructure:

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

For the external etcd cluster only, you also need:

- Three additional machines for etcd members

First steps for both methods

Create load balancer for kube-apiserver

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

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

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

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

   ```
   nc -v LOAD_BALANCER_IP PORT
   ```

   - A connection refused error is expected because the apiserver is not yet running. A timeout, however, means the load balancer cannot

communicate with the control plane node. If a timeout occurs, reconfigure the load balancer to communicate with the control plane node.

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

Stacked control plane and etcd nodes

Steps for the first control plane node

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

   ```
   apiVersion: kubeadm.k8s.io/v1beta1
   kind: ClusterConfiguration
   kubernetesVersion: stable
   controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
   ```

   - `kubernetesVersion` should be set to the Kubernetes version to use. This example uses `stable`.
   - `controlPlaneEndpoint` should match the address or DNS and port of the load balancer.
   - It's recommended that the versions of kubeadm, kubelet, kubectl and Kubernetes match.

   Note: Some CNI network plugins like Calico require a CIDR such as `192.168.0.0/16` and some like Weave do not. See the CNI network documentation. To add a pod CIDR set the `podSubnet: 192.168.0.0/16` field under the `networking` object of `ClusterConfiguration`.

1. Initialize the control plane:

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

   - The `--experimental-upload-certs` flag is used to upload the certificates that should be shared across all the control-plane instances to the cluster. If instead, you prefer to copy certs across control-plane nodes manually or using automation tools, please remove this flag and refer to Manual certificate distribution section bellow.

   After the command completes you should see something like so:

   ```
   ...
   You can now join any number of control-plane node by running the following command on e
     kubeadm join 192.168.0.200:6443 --token 9vr73a.a8uxyaju799qwdjv --discovery-token-ca-

   Please note that the certificate-key gives access to cluster sensitive data, keep it se
   As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use
   ```

```
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-
```

- Copy this output to a text file. You will need it later to join control plane and worker nodes to the cluster.
- When `--experimental-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 --experimental-upload-certs
```

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

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

1. Apply the CNI plugin of your choice:

   Follow these instructions to install the CNI provider. Make sure the configuration corresponds to the Pod CIDR specified in the kubeadm configuration file if applicable.

   In this example we are using Weave Net:

   ```
   kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | bas
   ```

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

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

Steps for the rest of the control plane nodes

Caution: 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-
   ```

   - The `--experimental-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/v1beta1
   kind: ClusterConfiguration
   kubernetesVersion: stable
   controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT"
   etcd:
       external:
           endpoints:
           - https://ETCD_0_IP:2379
           - https://ETCD_1_IP:2379
           - https://ETCD_2_IP:2379
           caFile: /etc/kubernetes/pki/etcd/ca.crt
           certFile: /etc/kubernetes/pki/apiserver-etcd-client.crt
           keyFile: /etc/kubernetes/pki/apiserver-etcd-client.key
   ```

Note: The difference between stacked etcd and external etcd here is that we are using the `external` field for `etcd` in the kubeadm config. In the case of the stacked etcd topology this is managed automatically.

- Replace the following variables in the config template with the appropriate values for yo

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

The following steps are exactly the same as described for stacked etcd setup:

1. Run `sudo kubeadm init --config kubeadm-config.yaml --experimental-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 | bas
   ```

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-ce
```

Manual certificate distribution

If you choose to not use `kubeadm init` with the `--experimental-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}"@$host:
       scp /etc/kubernetes/pki/front-proxy-ca.key "${USER}"@$host:
       scp /etc/kubernetes/pki/etcd/ca.crt "${USER}"@$host:etcd-ca.crt
       scp /etc/kubernetes/pki/etcd/ca.key "${USER}"@$host:etcd-ca.key
   done
   ```

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

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

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on April 10, 2019 at 2:08 AM PST by fix typo in ha page (#13735) (Page History)

Edit This Page

Set up a High Availability etcd cluster with kubeadm

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

- Before you begin
- Setting up the cluster
- What's next

Before you begin

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

Setting up the cluster

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

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

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

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

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

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

2. Create configuration files for kubeadm.

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

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

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

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

for i in "${!ETCDHOSTS[@]}"; do
HOST=${ETCDHOSTS[$i]}
NAME=${NAMES[$i]}
cat << EOF > /tmp/${HOST}/kubeadmcfg.yaml
apiVersion: "kubeadm.k8s.io/v1beta1"
kind: ClusterConfiguration
etcd:
    local:
        serverCertSANs:
        - "${HOST}"
        peerCertSANs:
        - "${HOST}"
        extraArgs:
            initial-cluster: ${NAMES[0]}=https://${ETCDHOSTS[0]}:2380,${NAMES[1]}=https
            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}
   kubeadmcfg.yaml
---
/etc/kubernetes/pki
   apiserver-etcd-client.crt
   apiserver-etcd-client.key
   etcd
        ca.crt
        ca.key
        healthcheck-client.crt
        healthcheck-client.key
        peer.crt
        peer.key
        server.crt
        server.key
```

   On `$HOST1`:

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

   On `$HOST2`

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

```
peer.crt
peer.key
server.crt
server.key
```

7. Create the static pod manifests

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

   ```
   root@HOST0 $ kubeadm init phase etcd local --config=/tmp/${HOST0}/kubeadmcfg.yaml
   root@HOST1 $ kubeadm init phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
   root@HOST2 $ kubeadm init phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
   ```

8. Optional: Check the cluster health

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

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

What's next

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on February 11, 2019 at 1:35 AM PST by docs: fix typo (#12568) (Page History)

Edit This Page

Configuring each kubelet in your cluster using kubeadm

FEATURE STATE: `Kubernetes 1.11` stable

This feature is *stable*, meaning:

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

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

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

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

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

Kubelet configuration patterns

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

Propagating cluster-level configuration to each kubelet

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

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

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

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

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

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

Providing instance-specific configuration details

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

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

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

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

Configure kubelets using kubeadm

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

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

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

Workflow when using `kubeadm init`

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

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

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

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

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

`systemctl daemon-reload && systemctl restart kubelet`

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

Workflow when using `kubeadm join`

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

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

`systemctl daemon-reload && systemctl restart kubelet`

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

The kubelet drop-in file for systemd

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

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

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

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

Kubernetes binaries and package contents

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

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on April 09, 2019 at 3:07 AM PST by kubelet-integration: Update kubeadm config command (#13636) (Page History)

Edit This Page

Troubleshooting kubeadm

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

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

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

- `ebtables` or some similar executable not found during installation
- kubeadm blocks waiting for control plane during installation
- kubeadm blocks when removing managed containers
- Pods in `RunContainerError`, `CrashLoopBackOff` or `Error` state
- `coredns` (or `kube-dns`) is stuck in the `Pending` state
- `HostPort` services do not work
- Pods are not accessible via their Service IP
- TLS certificate errors
- Default NIC When using flannel as the pod network in Vagrant
- Non-public IP used for containers
- `coredns` pods have `CrashLoopBackOff` or `Error` state
- etcd pods restart continually
- Not possible to pass a comma separated list of values to arguments inside a `--component-extra-args` flag
- kube-proxy scheduled before node is initialized by cloud-controller-manager
- The NodeRegistration.Taints field is omitted when marshalling kubeadm configuration

`ebtables` or some similar executable not found during installation

If you see the following warnings while running `kubeadm init`

```
[preflight] WARNING: ebtables not found in system path
[preflight] WARNING: ethtool not found in system path
```

Then you may be missing `ebtables`, `ethtool` or a similar executable on your node. You can install them with the following commands:

- For Ubuntu/Debian users, run `apt install ebtables ethtool`.
- For CentOS/Fedora users, run `yum install ebtables ethtool`.

kubeadm blocks waiting for control plane during installation

If you notice that `kubeadm init` hangs after printing out the following line:

```
[apiclient] Created API client, waiting for the control plane to become ready
```

This may be caused by a number of problems. The most common are:

- network connection problems. Check that your machine has full network connectivity before continuing.
- the default cgroup driver configuration for the kubelet differs from that used by Docker. Check the system log file (e.g. `/var/log/message`) or examine the output from `journalctl -u kubelet`. If you see something like the following:

```
error: failed to run Kubelet: failed to create kubelet:
misconfiguration: kubelet cgroup driver: "systemd" is different from docker cgroup driver:
```

There are two common ways to fix the cgroup driver problem:

1. Install Docker again following instructions here.
2. Change the kubelet config to match the Docker cgroup driver manually, you can refer to Configure cgroup driver used by kubelet on Master Node for detailed instructions.

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

kubeadm blocks when removing managed containers

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

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

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

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

Inspecting the logs for docker may also be useful:

```
journalctl -ul docker
```

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

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

- If there are pods in one of these states *right after* `kubeadm init`, please open an issue in the kubeadm repo. `coredns` (or `kube-dns`) should be in the `Pending` state until you have deployed the network solution.
- If you see Pods in the `RunContainerError`, `CrashLoopBackOff` or `Error` state after deploying the network solution and nothing happens to `coredns` (or `kube-dns`), it's very likely that the Pod Network solution 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 bec
```

- Verify that the `$HOME/.kube/config` file contains a valid certificate, and regenerate a certificate if necessary. The certificates in a kubeconfig file are base64 encoded. The `base64 -d` command can be used to decode the certificate and `openssl x509 -text -noout` can be used for viewing the certificate information.
- Unset the `KUBECONFIG` environment variable using:

```
unset KUBECONFIG
```

Or set it to the default `KUBECONFIG` location:

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

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

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

Default NIC When using flannel as the pod network in Vagrant

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

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

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

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

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

Non-public IP used for containers

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

```
Error from server: Get https://10.19.0.41:10250/containerLogs/default/mysql-ddc65b868-glc5m/
```

- This may be due to Kubernetes using an IP that can not communicate with other IPs on the seemingly same subnet, possibly by policy of the machine provider.
- Digital Ocean assigns a public IP to `eth0` as well as a private one to be used internally as anchor for their floating IP feature, yet `kubelet` will pick the latter as the node's `InternalIP` instead of the public one.

Use `ip addr show` to check for this scenario instead of `ifconfig` because `ifconfig` will not display the offending alias IP address. Alternatively an API endpoint specific to Digital Ocean allows to query for the anchor IP from the droplet:

```
curl http://169.254.169.254/metadata/v1/interfaces/public/0/anchor_ipv4/address
```

The workaround is to tell `kubelet` which IP to use using `--node-ip`. When using Digital Ocean, it can be the public one (assigned to `eth0`) or the private one (assigned to `eth1`) should you want to use the optional private network. The `KubeletExtraArgs` section of the kubeadm `NodeRegistrationOptions` structure can be used for this.

Then restart `kubelet`:

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

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

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

that you can try one of the following options:

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

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

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

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

etcd pods restart continually

If you encounter the following error:

```
rpc error: code = 2 desc = oci runtime error: exec failed: container_linux.go:247: starting
```

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.git
  ```

- 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.r
  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=Namesp` 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": { "tolerati
```

The tracking issue for this problem is here.

The NodeRegistration.Taints field is omitted when marshalling kubeadm configuration

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

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

There are at least two workarounds:

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

2. Remove the taint after kubeadm init exits:

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on April 09, 2019 at 10:14 PM PST by Document workaround for kubeadm Taints field issue (#13343) (Page History)

Edit This Page

Running Kubernetes on Multiple Clouds with Stackpoint.io

StackPointCloud is the universal control plane for Kubernetes Anywhere. StackPointCloud allows you to deploy and manage a Kubernetes cluster to the cloud provider of your choice in 3 steps using a web-based interface.

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

AWS

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

1. Choose a Provider

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

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

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

2. Configure Your Provider

   a. Add your Access Key ID and a Secret Access Key from AWS. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on AWS, consult the Kubernetes documentation.

GCE

To create a Kubernetes cluster on GCE, you will need the Service Account JSON Data from Google.

1. Choose a Provider

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

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

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

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on GCE, consult the Kubernetes documentation.

Google Kubernetes Engine

To create a Kubernetes cluster on Google Kubernetes Engine, you will need the Service Account JSON Data from Google.

1. Choose a Provider

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

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

   c. Click to select Google Kubernetes Engine.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on Google Kubernetes Engine, consult the official documentation.

DigitalOcean

To create a Kubernetes cluster on DigitalOcean, you will need a DigitalOcean API Token.

1. Choose a Provider

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

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

c. Click to select DigitalOcean.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on DigitalOcean, consult the official documentation.

Microsoft Azure

To create a Kubernetes cluster on Microsoft Azure, you will need an Azure Subscription ID, Username/Email, and Password.

1. Choose a Provider

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

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

   c. Click to select Microsoft Azure.

2. Configure Your Provider

   a. Add your Azure Subscription ID, Username/Email, and Password. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on Azure, consult the Kubernetes documentation.

Packet

To create a Kubernetes cluster on Packet, you will need a Packet API Key.

1. Choose a Provider

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

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

   c. Click to select Packet.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on Packet, consult the official documentation.

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on July 18, 2018 at 4:26 PM PST by Change formatting of Stackpoint turnkey doc (#9493) (Page History)

Edit This Page

Running Kubernetes on AWS EC2

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

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

Before you begin

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

Supported Production Grade Tools

- conjure-up is an open-source installer for Kubernetes that creates Kubernetes clusters with native AWS integrations on Ubuntu.
- Kubernetes Operations - Production Grade K8s Installation, Upgrades, and Management. Supports running Debian, Ubuntu, CentOS, and RHEL in AWS.
- CoreOS Tectonic includes the open-source Tectonic Installer that creates Kubernetes clusters with Container Linux nodes on AWS.
- CoreOS originated and the Kubernetes Incubator maintains a CLI tool, kube-aws, that creates and manages Kubernetes clusters with Container Linux nodes, using AWS tools: EC2, CloudFormation and Autoscaling.

Getting started with your cluster

Command line administration tool: kubectl

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

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

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

# Linux
```

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

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

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

Examples

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

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

For more complete applications, please look in the examples directory

Scaling the cluster

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

Tearing down the cluster

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

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

Support Level

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

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

Further reading

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on August 21, 2018 at 11:37 PM PST by apply template (#9961) (Page History)

Edit This Page

Running Kubernetes on Alibaba Cloud

- – Alibaba Cloud Container Service
  – Custom Deployments

Alibaba Cloud Container Service

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

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

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

Custom Deployments

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

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on January 24, 2019 at 2:03 AM PST by update alibaba-cloud.md (#11940) (Page History)

Edit This Page

Running Kubernetes on Azure

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

Azure Kubernetes Service (AKS)

The Azure Kubernetes Service offers simple deployments for Kubernetes clusters.

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

Microsoft Azure Kubernetes Service

Custom Deployments: 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.

Create an Issue Edit This Page

Page last modified on February 28, 2019 at 5:04 AM PST by docs: update acs-engine -> aks-engine (#12804) (Page History)

Edit This Page

Running Kubernetes on CenturyLink Cloud

{: toc}

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

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

Find Help

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

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

Clusters of VMs or Physical Servers, your choice.

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

Requirements

The requirements to run this script are:

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

Script Installation

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

1) Clone this repository and cd into it.

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

2) Install all requirements, including

- Ansible
- CenturyLink Cloud SDK
- Ansible Modules

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

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

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

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

Script Installation Example: Ubuntu 14 Walkthrough

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

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

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

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

Cluster Creation

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

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

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

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

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

Cluster Creation: Script Options

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

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

     -h (--help)                   display this help and exit
     -c= (--clc_cluster_name=)     set the name of the cluster, as used in CLC group names
     -t= (--minion_type=)          standard -> VM (default), bareMetal -> physical]
     -d= (--datacenter=)           VA1 (default)
     -m= (--minion_count=)         number of kubernetes minion nodes
     -mem= (--vm_memory=)          number of GB ram for each minion
     -cpu= (--vm_cpu=)             number of virtual cps for each minion node
     -phyid= (--server_conf_id=)   physical server configuration id, one of
                                      physical_server_20_core_conf_id
                                      physical_server_12_core_conf_id
                                      physical_server_4_core_conf_id (default)
     -etcd_separate_cluster=yes    create a separate cluster of three etcd nodes,
                                   otherwise run etcd on the master node
```

Cluster Expansion

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

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

Cluster Expansion: Script Options

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

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

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

Cluster Deletion

There are two ways to delete an existing cluster:

1) Use our python script:

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

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

Examples

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

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

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

```
bash kube-up.sh --clc_cluster_name=k8s_2 --minion_type=standard --minion_count=6 --datacente
```

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

```
bash kube-up.sh --clc_cluster_name=k8s_3 --minion_type=standard --minion_count=10 --datacent
```

Cluster Features and Architecture

We configure the Kubernetes cluster with the following features:

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

We use the following to create the Kubernetes cluster:

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

Optional add-ons

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

aggregation documentation. Note: We don't install this by default as the footprint isn't trivial.

Cluster management

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

The script requires that the environment variable `CLC_CLUSTER_NAME` be defined

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

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

Accessing the cluster programmatically

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

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

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

But please note, this *does not* work out of the box with the `curl` binary distributed with macOS.

Accessing the cluster with a browser

We install the kubernetes dashboard. When you create a cluster, the script should output URLs for these interfaces like this:

kubernetes-dashboard is running at `https://${MASTER_IP}:6443/api/v1/namespaces/kube-system/servic`
```

Note on Authentication to the UIs: The cluster is set up to use basic authentication for the user *admin*. Hitting the url at

```
from the apiserver, and then presenting the admin password written to file at:

```> _${CLC_CLUSTER_HOME}/kube/admin_password.txt_```


Configuration files

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

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

```kubectl``` usage examples

There are a great many features of _kubectl_. Here are a few examples

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

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

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

```
kubectl proxy -p 8001
```

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

What Kubernetes features do not work on CenturyLink Cloud

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

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

Ansible Files

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

Further reading

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

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 14, 2019 at 3:26 AM PST by Improved openshift, docker, ansible references (#12201) (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
- Starting a cluster
- Installing the Kubernetes command line tools on your workstation
- Getting started with your cluster
- Tearing down the cluster
- Customizing
- Troubleshooting
- Support Level
- Further reading

Before you begin

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

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

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

Prerequisites

1. You need a Google Cloud Platform account with billing enabled. Visit the Google Developers Console for more details.
2. Install `gcloud` as necessary. `gcloud` can be installed as a part of the Google Cloud SDK.
3. Enable the Compute Engine Instance Group Manager API in the Google Cloud developers console.
4. Make sure that gcloud is set to use the Google Cloud Platform project you want. You can check the current project using `gcloud config list project` and change it via `gcloud config set project <project-id>`.
5. Make sure you have credentials for GCloud by running `gcloud auth login`.
6. (Optional) In order to make API calls against GCE, you must also run `gcloud auth application-default login`.
7. Make sure you can start up a GCE VM from the command line. At least make sure you can do the Create an instance part of the GCE Quickstart.

8. Make sure you can SSH into the VM without interactive prompts. See the Log in to the instance part of the GCE Quickstart.

Starting a cluster

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

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

or

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

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

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

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

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

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

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

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

The next few steps will show you:

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

Installing the Kubernetes command line tools on your workstation

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

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

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

```
gcloud components install kubectl
```

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

Getting started with your cluster

Inspect your cluster

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

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

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

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

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

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

command.

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

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

```
kube-system   fluentd-cloud-logging-kubernetes-minion-c1n9   1/1       Running   0
kube-system   fluentd-cloud-logging-kubernetes-minion-c4og   1/1       Running   0
kube-system   fluentd-cloud-logging-kubernetes-minion-ngua   1/1       Running   0
kube-system   kube-ui-v1-curt1                              1/1       Running   0
kube-system   monitoring-heapster-v5-ex4u3                  1/1       Running   1
kube-system   monitoring-influx-grafana-v1-piled            2/2       Running   0
```

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

Run some examples

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

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

Tearing down the cluster

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

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

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

Customizing

The script above relies on Google Storage to stage the Kubernetes release. It then will start (by default) a single master VM along with 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

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

Further reading

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

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 December 17, 2018 at 12:02 PM PST by GCE: cluster/kube-up.sh creates 3 nodes by default (#11694) (Page History)

Edit This Page

Running Kubernetes on Multiple Clouds with IBM Cloud Private

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

IBM® 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 on Red Hat OpenShift

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

Integration capabilities:

- Supports Linux® 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.

Create an Issue Edit This Page

Page last modified on February 24, 2019 at 8:28 AM PST by Revising bluemix.net links to cloud.ibm.com (#12792) (Page History)

Edit This Page

Running Kubernetes on Multiple Clouds with Stackpoint.io

StackPointCloud is the universal control plane for Kubernetes Anywhere. StackPointCloud allows you to deploy and manage a Kubernetes cluster to the cloud provider of your choice in 3 steps using a web-based interface.

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

AWS

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

1. Choose a Provider

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

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

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

2. Configure Your Provider

   a. Add your Access Key ID and a Secret Access Key from AWS. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on AWS, consult the Kubernetes documentation.

GCE

To create a Kubernetes cluster on GCE, you will need the Service Account JSON Data from Google.

1. Choose a Provider

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

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

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

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on GCE, consult the Kubernetes documentation.

Google Kubernetes Engine

To create a Kubernetes cluster on Google Kubernetes Engine, you will need the Service Account JSON Data from Google.

1. Choose a Provider

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

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

   c. Click to select Google Kubernetes Engine.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on Google Kubernetes Engine, consult the official documentation.

DigitalOcean

To create a Kubernetes cluster on DigitalOcean, you will need a DigitalOcean API Token.

1. Choose a Provider

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

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

   c. Click to select DigitalOcean.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on DigitalOcean, consult the official documentation.

Microsoft Azure

To create a Kubernetes cluster on Microsoft Azure, you will need an Azure Subscription ID, Username/Email, and Password.

1. Choose a Provider

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

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

   c. Click to select Microsoft Azure.

2. Configure Your Provider

   a. Add your Azure Subscription ID, Username/Email, and Password. Select your default StackPointCloud SSH keypair, or click **ADD SSH KEY** to add a new keypair.

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on Azure, consult the Kubernetes documentation.

Packet

To create a Kubernetes cluster on Packet, you will need a Packet API Key.

1. Choose a Provider

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

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

   c. Click to select Packet.

2. Configure Your Provider

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

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

3. Configure Your Cluster

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

4. Run the Cluster

   You can monitor the status of your cluster and suspend or delete it from your stackpoint.io dashboard.

   For information on using and managing a Kubernetes cluster on Packet, consult the official documentation.

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on July 18, 2018 at 4:26 PM PST by Change formatting of Stackpoint turnkey doc (#9493) (Page History)

Edit This Page

Installing Kubernetes on AWS with kops

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

kops is an opinionated provisioning system:

- Fully automated installation
- Uses DNS to identify clusters
- Self-healing: everything runs in Auto-Scaling Groups
- Multiple OS support (Debian, Ubuntu 16.04 supported, CentOS & RHEL, Amazon Linux and CoreOS) - see the images.md
- High-Availability support - see the high_availability.md
- Can directly provision, or generate terraform manifests - see the terraform.md

If your opinions differ from these you may prefer to build your own cluster using kubeadm as a building block. kops builds on the kubeadm work.

- Creating a cluster
- Cleanup
- Feedback
- What's next

Creating a cluster

(1/5) Install kops

Requirements

You must have kubectl installed in order for kops to work.

Installation

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

On macOS:

```
curl -OL https://github.com/kubernetes/kops/releases/download/1.10.0/kops-darwin-amd64
chmod +x kops-darwin-amd64
mv kops-darwin-amd64 /usr/local/bin/kops
# you can also install using Homebrew
brew update && brew install kops
```

On Linux:

```
wget https://github.com/kubernetes/kops/releases/download/1.10.0/kops-linux-amd64
chmod +x kops-linux-amd64
mv kops-linux-amd64 /usr/local/bin/kops
```

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

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

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

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

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

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

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

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

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

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

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

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

Multiple clusters can use the same S3 bucket, and you can share an S3 bucket between your colleagues that administer the same clusters - this is much easier than passing around kubecfg files. But anyone with access to the S3 bucket will

have administrative access to all your clusters, so you don't want to share it beyond the operations team.

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

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

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

(4/5) Build your cluster configuration

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

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

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

It prints commands you can use to explore further:

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

If this is your first time using kops, do spend a few minutes to try those out! An instance group is a set of instances, which will be registered as kubernetes nodes. On AWS this is implemented via auto-scaling-groups. You can have several instance groups, for example if you wanted nodes that are a mix of spot and on-demand instances, or GPU and non-GPU instances.

(5/5) Create the cluster in AWS

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

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

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

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

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

Explore other add-ons

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

Cleanup

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

Feedback

- Slack Channel: #kops-users
- GitHub Issues

What's next

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on August 21, 2018 at 11:44 PM PST by apply template (#9964) (Page History)

Edit This Page

Installing Kubernetes On-premises/Cloud Providers with Kubespray

This quickstart helps to install a Kubernetes cluster hosted on GCE, Azure, OpenStack, AWS, vSphere, Oracle Cloud Infrastructure (Experimental) or Baremetal with Kubespray.

Kubespray is a composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks. Kubespray provides:

- a highly available cluster
- composable attributes
- support for most popular Linux distributions
  - Container Linux by CoreOS
  - Debian Jessie, Stretch, Wheezy
  - Ubuntu 16.04, 18.04
  - CentOS/RHEL 7
  - Fedora/CentOS Atomic
  - openSUSE Leap 42.3/Tumbleweed
- continuous integration tests

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

- Creating a cluster
- Cluster operations
- Cleanup
- Feedback
- What's next

Creating a cluster

(1/5) Meet the underlay requirements

Provision servers with the following requirements:

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

Kubespray provides the following utilities to help provision your environment:

- Terraform scripts for the following cloud providers:
  - AWS
  - OpenStack

(2/5) Compose an inventory file

After you provision your servers, create an inventory file for Ansible. You can do this manually or via a dynamic inventory script. For more information, see "Building your own inventory".

(3/5) Plan your cluster deployment

Kubespray provides the ability to customize many aspects of the deployment:

- Choice deployment mode: kubeadm or non-kubeadm
- CNI (networking) plugins
- DNS configuration
- Choice of control plane: native/binary or containerized with docker or rkt
- Component versions
- Calico route reflectors
- Component runtime options
  - docker
  - rkt

  - cri-o
- Certificate generation methods (**Vault being discontinued**)

Kubespray customizations can be made to a variable file. If you are just getting started with Kubespray, consider using the Kubespray defaults to deploy your cluster and explore Kubernetes.

(4/5) Deploy a Cluster

Next, deploy your cluster:

Cluster deployment using ansible-playbook.

```
ansible-playbook -i your/inventory/hosts.ini cluster.yml -b -v \
  --private-key=~/.ssh/private_key
```

Large deployments (100+ nodes) may require specific adjustments for best results.

(5/5) Verify the deployment

Kubespray provides a way to verify inter-pod connectivity and DNS resolve with Netchecker. Netchecker ensures the netchecker-agents pods can resolve DNS requests and ping each over within the default namespace. Those pods mimic similar behavior of the rest of the workloads and serve as cluster health indicators.

Cluster operations

Kubespray provides additional playbooks to manage your cluster: *scale* and *upgrade*.

Scale your cluster

You can add worker nodes from your cluster by running the scale playbook. For more information, see "Adding nodes". You can remove worker nodes from your cluster by running the remove-node playbook. For more information, see "Remove nodes".

Upgrade your cluster

You can upgrade your cluster by running the upgrade-cluster playbook. For more information, see "Upgrades".

Cleanup

You can reset your nodes and wipe out all components installed with Kubespray via the reset playbook.

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

Feedback

- Slack Channel: #kubespray
- GitHub Issues

What's next

Check out planned work on Kubespray's roadmap.

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 16, 2018 at 5:05 PM PST by Update kubespray.md (#10992) (Page History)

Edit This Page

Installing Kubernetes on AWS with kops

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

kops is an opinionated provisioning system:

- Fully automated installation
- Uses DNS to identify clusters
- Self-healing: everything runs in Auto-Scaling Groups
- Multiple OS support (Debian, Ubuntu 16.04 supported, CentOS & RHEL, Amazon Linux and CoreOS) - see the images.md
- High-Availability support - see the high_availability.md
- Can directly provision, or generate terraform manifests - see the terraform.md

If your opinions differ from these you may prefer to build your own cluster using kubeadm as a building block. kops builds on the kubeadm work.

- Creating a cluster
- Cleanup
- Feedback
- What's next

Creating a cluster

(1/5) Install kops

Requirements

You must have kubectl installed in order for kops to work.

Installation

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

On macOS:

```
curl -OL https://github.com/kubernetes/kops/releases/download/1.10.0/kops-darwin-amd64
chmod +x kops-darwin-amd64
mv kops-darwin-amd64 /usr/local/bin/kops
# you can also install using Homebrew
brew update && brew install kops
```

On Linux:

```
wget https://github.com/kubernetes/kops/releases/download/1.10.0/kops-linux-amd64
chmod +x kops-linux-amd64
mv kops-linux-amd64 /usr/local/bin/kops
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(4/5) Build your cluster configuration

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

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

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

It prints commands you can use to explore further:

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

If this is your first time using kops, do spend a few minutes to try those out! An instance group is a set of instances, which will be registered as kubernetes nodes. On AWS this is implemented via auto-scaling-groups. You can have several instance groups, for example if you wanted nodes that are a mix of spot and on-demand instances, or GPU and non-GPU instances.

(5/5) Create the cluster in AWS

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

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

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

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

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

Explore other add-ons

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

Cleanup

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

Feedback

- Slack Channel: #kops-users
- GitHub Issues

What's next

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on August 21, 2018 at 11:44 PM PST by apply template (#9964) (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.

Create an Issue Edit This Page

Page last modified on July 19, 2018 at 7:25 AM PST by apply templates/concept (#9539) (Page History)

Edit This Page

Cloudstack

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

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

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

- Prerequisites
- Support Level

Prerequisites

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

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

```
yum install libselinux-python
```

cs is a python module for the CloudStack API.

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

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

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

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

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

Clone the playbook

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

Create a Kubernetes cluster

You simply need to run the playbook.

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

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

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

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

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

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

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

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

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

And you can list the machines in your cluster:

```
fleetctl list-machines
```

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

Support Level

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

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on July 19, 2018 at 6:04 AM PST by apply templates/concept and fix code snippets (#9540) (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.

Create an Issue Edit This Page

Page last modified on July 19, 2018 at 7:25 AM PST by apply templates/concept (#9539) (Page History)

Edit This Page

oVirt

oVirt is a virtual datacenter manager that delivers powerful management of multiple virtual machines on multiple hosts. Using KVM and libvirt, oVirt can be installed on Fedora, CentOS, or Red Hat Enterprise Linux hosts to set up and manage your virtual data center.

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

oVirt Cloud Provider Deployment

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

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

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

Using the oVirt Cloud Provider

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

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

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

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

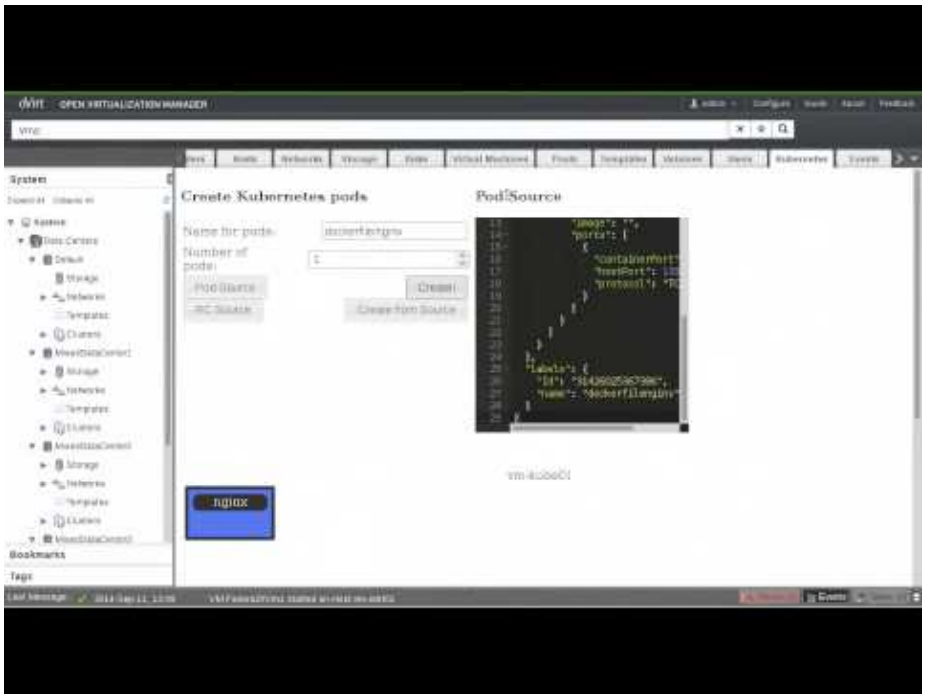
In the above example all the VMs tagged with the `kubernetes` label will be reported as nodes to Kubernetes.

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

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

oVirt Cloud Provider Screencast

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



Support Level

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

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on August 08, 2018 at 4:22 PM PST by Fix netlify "mixed content detected" warning (#9683) (Page History)

Edit This Page

Intro to Windows support in Kubernetes

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

- Windows containers in Kubernetes
- Supported Functionality and Limitations
- Getting Help and Troubleshooting
- Reporting Issues and Feature Requests
- What's next

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/W*
Kubernetes v1.14	Not Supported	Not Supported	Supported for Windows

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

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

Note: Windows containers with process isolation have strict compatibility rules, where the host OS version must match the container

base image OS version. Once we support Windows containers with Hyper-V isolation in Kubernetes, the limitation and compatibility rules will change.

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

- Pods

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

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

- Controllers

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

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

- Services

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

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

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

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

Container Runtime

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

Storage

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

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

Networking

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

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

The following service spec types are supported:

- NodePort
- ClusterIP
- LoadBalancer
- ExternalName

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

Network Driver	Description
L2bridge	Containers are attached to an ex
L2Tunnel	This is a special case of l2bridge
Overlay (Overlay networking for Windows in Kubernetes is in *alpha* stage)	Containers are given a vNIC cor
Transparent (special use case for ovn-kubernetes)	Requires an external vSwitch. C
NAT (*not used in Kubernetes*)	Containers are given a vNIC cor

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

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

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

The following IPAM options are supported on Windows:

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

Limitations

Control Plane

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

Compute

Resource management and process isolation

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

Operating System Restrictions

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

Feature Restrictions

- TerminationGracePeriod: not implemented
- Single file mapping: to be implemented with CRI-ContainerD
- Termination message: to be implemented with CRI-ContainerD
- Privileged Containers: not currently supported in Windows containers
- HugePages: not currently supported in Windows containers

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

Memory Reservations and Handling

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

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

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

The behavior of the flags behave differently as described below:

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

Storage

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

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

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

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

Networking

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

The Windows host networking networking service and virtual switch implement namespacing and can create virtual NICs as needed for a pod or container. However, many configurations such as DNS, routes, and metrics are stored in the Windows registry database rather than /etc/… 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
- `kubectl port-forward`
- 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.

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 '.' as a FQDN and skips PQDN resolution

- On Linux, you have a DNS suffix list, which is used when trying to resolve PQDNs. On Windows, we only have 1 DNS suffix, which is the DNS suffix associated with that pod's namespace (mydns.svc.cluster.local for example). Windows can resolve FQDNs and services or names resolvable with just that suffix. For example, a pod spawned in the default namespace, will have the DNS suffix **default.svc.cluster.local**. On a Windows pod, you can resolve both **kubernetes.default.svc.cluster.local** and **kubernetes**, but not the in-betweens, like **kubernetes.default** or **kubernetes.default.svc**.

Security

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

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

RunAsUseris 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.mem - 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.

   (a) 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 'minio
   ```

```
# 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
```

(b) 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> -Clust

# NetworkMode      = The network mode l2bridge (flannel host-gw, also the default v
# ManagementIP     = The IP address assigned to the Windows node. You can use ipcon
# 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
```

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\config --iface=<ManagementIP
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/co
# For more info search for "pause" in the "Guide for adding Windows Nodes in Kubern
nssm install kubelet C:\k\kubelet.exe
nssm set kubelet AppParameters --hostname-override=<hostname> --v=6 --pod-infra-con
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-overric
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="W
nssm set kube-proxy DependOnService kubelet
nssm start kube-proxy
```

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

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

For additional details, see official nssm usage docs.

3. My Windows Pods do not have network connectivity

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

4. My Windows Pods cannot ping external resources

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

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

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

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

```
]
```

5. My Windows node cannot access NodePort service

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

6. vNICs and HNS endpoints of containers are being deleted

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

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

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

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

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

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

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

   ```
   PS C:> [Environment]::SetEnvironmentVariable("NODE_NAME", "<Windows_Worker_Hostname>")
   PS C:> C:\flannel\flanneld.exe --kubeconfig-file=c:\k\config --iface=<Windows_Worker_No
   ```

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 — a necessary prerequisite for Kubelet to work — 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:1.0.0`. For more information search for "pause" in the Guide for adding Windows Nodes in Kubernetes.

13. DNS resolution is not properly working

    Check the DNS limitations for Windows in this section.

Further investigation

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

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

Reporting Issues and Feature Requests

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

and add additional logs. SIG-Windows Slack is also a great avenue to get some initial support and troubleshooting ideas prior to creating a ticket.

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

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

What's next

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

CRI-ContainerD

ContainerD is another OCI-compliant runtime that recently graduated as a CNCF project. It's currently tested on Linux, but 1.3 will bring support for Windows and Hyper-V. [reference]

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

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

Hyper-V isolation

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iis
spec:
  selector:
    matchLabels:
      app: iis
  replicas: 3
  template:
    metadata:
      labels:
        app: iis
      annotations:
        experimental.windows.kubernetes.io/isolation-type: hyperv
    spec:
      containers:
      - name: iis
        image: microsoft/iis
        ports:
        - containerPort: 80
```

Deployment with kubeadm and cluster API

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

A few other key features

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on April 09, 2019 at 10:30 PM PST by Update intro-windows-in-kubernetes.md (#13739) (Page History)

Edit This Page

Intro to Windows support in Kubernetes

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

- Windows containers in Kubernetes
- Supported Functionality and Limitations
- Getting Help and Troubleshooting
- Reporting Issues and Feature Requests
- What's next

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/W*
Kubernetes v1.14	Not Supported	Not Supported	Supported for Windows

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

> **Note:** The Windows Server Host Operating System is subject to

the Windows Server licensing. The Windows Container images are subject to the Supplemental License Terms for Windows containers.

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

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

- Pods

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

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

- Controllers

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

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

- Services

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

  - Service Environment variables

  - NodePort
  - ClusterIP
  - LoadBalancer
  - ExternalName
  - Headless services

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

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

Container Runtime

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

Storage

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

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

Networking

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

- Virtual networks (including creation of vSwitches)

- Endpoints / vNICs
- Namespaces
- Policies (Packet encapsulations, Load-balancing rules, ACLs, NAT'ing rules, etc.)

The following service spec types are supported:

- NodePort
- ClusterIP
- LoadBalancer
- ExternalName

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

Network Driver	Description
L2bridge	Containers are attached to an ex
L2Tunnel	This is a special case of l2bridge
Overlay (Overlay networking for Windows in Kubernetes is in *alpha* stage)	Containers are given a vNIC cor
Transparent (special use case for ovn-kubernetes)	Requires an external vSwitch. C
NAT (*not used in Kubernetes*)	Containers are given a vNIC cor

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

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

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

- Pod -> External (IP)
- Pod -> External (DNS)
- Node -> Pod
- Pod -> Node

The following IPAM options are supported on Windows:

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

Limitations

Control Plane

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

Compute

Resource management and process isolation

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

Operating System Restrictions

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

Feature Restrictions

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

Memory Reservations and Handling

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

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

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

The behavior of the flags behave differently as described below:

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

Storage

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

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

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

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

Networking

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

The Windows host networking networking service and virtual switch implement namespacing and can create virtual NICs as needed for a pod or container. However, many configurations such as DNS, routes, and metrics are stored in the Windows registry database rather than /etc/… 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
- `kubectl port-forward`
- 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.

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 '.' as a FQDN and skips PQDN resolution
- On Linux, you have a DNS suffix list, which is used when trying to resolve PQDNs. On Windows, we only have 1 DNS suffix, which is the DNS suffix associated with that pod's namespace (mydns.svc.cluster.local for example). Windows can resolve FQDNs and services or names resolvable with just that suffix. For example, a pod spawned in the default namespace, will have the DNS suffix **default.svc.cluster.local**. On a Windows pod, you can resolve both **kubernetes.default.svc.cluster.local** and **kubernetes**, but not the in-betweens, like **kubernetes.default** or **kubernetes.default.svc**.

Security

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

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

RunAsUseris 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.mem - 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.

   (a) 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 'minio

# 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
```

(b) 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> -Clust

# NetworkMode      = The network mode l2bridge (flannel host-gw, also the default v
# ManagementIP     = The IP address assigned to the Windows node. You can use ipcon
# 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
```

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\config --iface=<ManagementIP
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/co
# For more info search for "pause" in the "Guide for adding Windows Nodes in Kubern
nssm install kubelet C:\k\kubelet.exe
nssm set kubelet AppParameters --hostname-override=<hostname> --v=6 --pod-infra-con
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-overric
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="W
nssm set kube-proxy DependOnService kubelet
nssm start kube-proxy
```

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

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

For additional details, see official nssm usage docs.

3. My Windows Pods do not have network connectivity

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

4. My Windows Pods cannot ping external resources

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

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

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

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

5. My Windows node cannot access NodePort service

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

6. vNICs and HNS endpoints of containers are being deleted

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

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

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

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

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

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

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

   ```
   PS C:> [Environment]::SetEnvironmentVariable("NODE_NAME", "<Windows_Worker_Hostname>")
   PS C:> C:\flannel\flanneld.exe --kubeconfig-file=c:\k\config --iface=<Windows_Worker_No
   ```

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 — a necessary prerequisite for Kubelet to work — 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:1.0.0`. For more information search for "pause" in the Guide for adding Windows Nodes in Kubernetes.

13. DNS resolution is not properly working

    Check the DNS limitations for Windows in this section.

Further investigation

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

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

Reporting Issues and Feature Requests

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

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

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

What's next

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

CRI-ContainerD

ContainerD is another OCI-compliant runtime that recently graduated as a CNCF project. It's currently tested on Linux, but 1.3 will bring support for Windows and Hyper-V. [reference]

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

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

Hyper-V isolation

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iis
spec:
  selector:
    matchLabels:
      app: iis
  replicas: 3
  template:
    metadata:
      labels:
        app: iis
      annotations:
        experimental.windows.kubernetes.io/isolation-type: hyperv
    spec:
      containers:
      - name: iis
        image: microsoft/iis
        ports:
        - containerPort: 80
```

Deployment with kubeadm and cluster API

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

A few other key features

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on April 09, 2019 at 10:30 PM PST by Update intro-windows-in-kubernetes.md (#13739) (Page History)

Edit This Page

Guide for adding Windows Nodes in Kubernetes

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

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

- Before you begin
- Getting Started: Adding a Windows Node to Your Cluster

Before you begin

- Obtain a Windows Server license in order to configure the Windows node that hosts Windows containers. You can use your organization's licenses for the cluster, or acquire one from Microsoft, a reseller, or via the major cloud providers such as GCP, AWS, and Azure by provisioning a virtual machine running Windows Server through their marketplaces. A time-limited trial is also available.
- Build a Linux-based Kubernetes cluster in which you have access to the control plane (some examples include Getting Started from Scratch, kubeadm, AKS Engine, GCE, AWS).

Getting Started: Adding a Windows Node to Your Cluster

Plan IP Addressing

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

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

Subnet / address range	Description
Service Subnet	A non-routable, purely virtual subnet that is used by pods to uniformly access
Cluster Subnet	This is a global subnet that is used by all pods in the cluster. Each node is ass
Kubernetes DNS Service IP	IP address of `kube-dns` service that is used for DNS resolution & cluster servic

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

Components that run on Windows

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

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

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

Networking Configuration

Once you have a Linux-based Kubernetes master node you are ready to choose a networking solution. This guide illustrates using Flannel in VXLAN mode for simplicity.

Configuring Flannel in VXLAN mode on the Linux controller

1. Prepare Kubernetes master for Flannel

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

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

2. Download & configure Flannel

   Download the most recent Flannel manifest:

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

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

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

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

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

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

   (a) The cluster subnet (e.g. "10.244.0.0/16") is set as per your IP plan.
   - VNI 4096 is set in the backend
   - Port 4789 is set in the backend

   (b) In the `cni-conf.json` section of your `kube-flannel.yml`, change the network name to `vxlan0`.

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

   ```
   cni-conf.json: |
       {
         "name": "vxlan0",
         "plugins": [
           {
             "type": "flannel",
             "delegate": {
   ```

```
            "hairpinMode": true,
            "isDefaultGateway": true
          }
        },
        {
          "type": "portmap",
          "capabilities": {
            "portMappings": true
          }
        }
      ]
    }
```

4. Apply the Flannel yaml and Validate

   Let's apply the Flannel configuration:

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

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

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

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

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

   alt_text

   Verify that the Flannel DaemonSet has the NodeSelector applied.

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

   alt_text

Join Windows Worker

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

Preparing a Windows Node

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

1. Install Docker (requires a system reboot)

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

```
Enable-WindowsOptionalFeature -FeatureName Containers
Restart-Computer -Force
Install-Module -Name DockerMsftProvider -Repository PSGallery -Force
Install-Package -Name Docker -ProviderName DockerMsftProvider
```

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

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

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

alt_text

```
Start-Service docker
```

> **Note:** The "pause" (infrastructure) image is hosted on Microsoft Container Registry (MCR). You can access it using "docker pull mcr.microsoft.com/k8s/core/pause:1.0.0". The DOCKERFILE is available at https://github.com/Microsoft/SDN/blob/master/Kubernetes/windows/Dockerfile.

2. Prepare a Windows directory for Kubernetes

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

   ```
   mkdir c:\k
   ```

3. Copy Kubernetes certificate

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

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

4. Download Kubernetes binaries

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

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

Join the Windows node to the Flannel cluster

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

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

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

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

```
.\start.ps1 -ManagementIP <Windows Node IP> -NetworkMode overlay  -ClusterCIDR <Cluster CIDR
```

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

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

```
kubectl get nodes
```

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

Public Cloud Providers

Azure

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

GCP

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

Deployment with kubeadm and cluster API

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

Next Steps

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

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on April 04, 2019 at 7:55 PM PST by Fixed broken links to /docs/setup/scratch (#13614) (Page History)

Edit This Page

Guide for scheduling Windows containers in Kubernetes

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

Objectives

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

Before you begin

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

Getting Started: Deploying a Windows container

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

```
apiVersion: v1
kind: Service
metadata:
  name: win-webserver
  labels:
    app: win-webserver
```

```
spec:
  ports:
  # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    app: win-webserver
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: win-webserver
  name: win-webserver
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: win-webserver
      name: win-webserver
    spec:
      containers:
      - name: windowswebserver
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        command:
        - powershell.exe
        - -command
        - "<#code used from https://gist.github.com/wagnerandrade/5424431#> ; $$listener
      nodeSelector:
        beta.kubernetes.io/os: windows
```

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

1. Check that all nodes are healthy:

   ```
   kubectl get nodes
   ```

2. Deploy the service and watch for pod updates:

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

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

3. Check that the deployment succeeded. To verify:

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

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

Managing Workload Identity with Group Managed Service Accounts

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

Taints and Tolerations

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

Ensuring OS-specific workloads land on the appropriate container host

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

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

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

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

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

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

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

Feedback

Was this page helpful?

Yes

No

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

if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page

Page last modified on March 25, 2019 at 5:06 PM PST by Official 1.14 Release Docs (#13174) (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
- Supported version skew
- Supported component upgrade order

Supported versions

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

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

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

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

Supported version skew

kube-apiserver

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

Example:

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

kubelet

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

Example:

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

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

Example:

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

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

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

Example:

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

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

Example:

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

kubectl

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

Example:

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

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

Example:

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

Supported component upgrade order

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

kube-apiserver

Pre-requisites:

- In a single-instance cluster, the existing `kube-apiserver` instance is **1.n**
- In an HA cluster, all `kube-apiserver` instances are at **1.n** or **1.(n+1)** (this ensures maximum skew of 1 minor version between the oldest and newest `kube-apiserver` instance)
- The `kube-controller-manager`, `kube-scheduler`, and `cloud-controller-manager` instances that communicate with this server are at version **1.n** (this

ensures they are not newer than the existing API server version, and are within 1 minor version of the new API server version)
- `kubelet` instances on all nodes are at version **1.n** or **1.(n-1)** (this ensures they are not newer than the existing API server version, and are within 2 minor versions of the new API server version)
- Registered admission webhooks are able to handle the data the new `kube-apiserver` instance will send them:
  - `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` objects are updated to include any new versions of REST resources added in **1.(n+1)**
  - The webhooks are able to handle any new versions of REST resources that will be sent to them, and any new fields added to existing versions in **1.(n+1)**

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

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

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

Pre-requisites:

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

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

kubelet

Pre-requisites:

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

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

> **Warning:** Running a cluster with `kubelet` instances that are persistently two minor versions behind `kube-apiserver` is not recommended:

- they must be upgraded within one minor version of `kube-apiserver` before the control plane can be upgraded
- it increases the likelihood of running `kubelet` versions older than the three maintained minor releases

Feedback

Was this page helpful?

Yes

No

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

Create an Issue Edit This Page

Page last modified on March 27, 2019 at 4:46 PM PST by Fix the spelling of 'available'. (#13483) (Page History)

Edit This Page

Building Large Clusters

Support

At v1.14, 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.

Create an Issue Edit This Page

Page last modified on August 20, 2018 at 11:31 AM PST by Refactor Setup and Imported sections (#9916) (Page History)

Edit This Page

Running in Multiple Zones

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

- Introduction
- Functionality
- Limitations
- Walkthrough

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-centra
```

AWS:

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

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

Nodes are labeled

View the nodes; you can see that they are labeled with zone information. They are all in `us-central1-a` (GCE) or `us-west-2a` (AWS) so far. The

labels are `failure-domain.beta.kubernetes.io/region` for the region, and `failure-domain.beta.kubernetes.io/zone` for the zone:

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

The output is similar to this:

```
NAME                     STATUS                     ROLES    AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>   6m    v1.13.0          beta.kub
kubernetes-minion-87j9   Ready                      <none>   6m    v1.13.0          beta.kub
kubernetes-minion-9vlv   Ready                      <none>   6m    v1.13.0          beta.kub
kubernetes-minion-a12q   Ready                      <none>   6m    v1.13.0          beta.kub
```

Add more nodes in a second zone

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

GCE:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-centra
```

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

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2
```

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

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

The output is similar to this:

```
NAME                     STATUS                     ROLES    AGE   VERSION          LABELS
kubernetes-master        Ready,SchedulingDisabled   <none>   16m   v1.13.0          beta.ku
kubernetes-minion-281d   Ready                      <none>   2m    v1.13.0          beta.ku
kubernetes-minion-87j9   Ready                      <none>   16m   v1.13.0          beta.ku
kubernetes-minion-9vlv   Ready                      <none>   16m   v1.13.0          beta.ku
kubernetes-minion-a12q   Ready                      <none>   17m   v1.13.0          beta.ku
kubernetes-minion-pp2f   Ready                      <none>   2m    v1.13.0          beta.ku
kubernetes-minion-wf8i   Ready                      <none>   2m    v1.13.0          beta.ku
```

Volume affinity

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

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

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

Now 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            STORAGEC
pv-gce-mj4gm   5Gi        RWO           Retain           Bound     default/claim1   manual
```

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

```
kubectl apply -f - <<EOF
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
```

```
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: claim1
EOF
```

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

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

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

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=
```

Pods are spread across zones

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

GCE:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=gce KUBE_GCE_ZONE=us-centra
```

AWS:

```
KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-west-2
```

Verify that you now have nodes in 3 zones:

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

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

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

```
NAME                     STATUS    ROLES    AGE    VERSION          LABELS
kubernetes-minion-9vlv   Ready     <none>   34m    v1.13.0          beta.kubernetes.io/insta
kubernetes-minion-281d   Ready     <none>   20m    v1.13.0          beta.kubernetes.io/insta
kubernetes-minion-olsh   Ready     <none>   3m     v1.13.0          beta.kubernetes.io/insta
```

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

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

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

AWS:

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

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 25, 2019 at 5:06 PM PST by Official 1.14 Release Docs (#13174) (Page History)

Edit This Page

CRI installation

FEATURE STATE: `Kubernetes v1.6` stable

This feature is *stable*, meaning:

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

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

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

  Caution:

  A flaw was found in the way runc handled system file descriptors when running containers. A malicious container could use this flaw

to overwrite contents of the runc binary and consequently run arbitrary commands on the container host system.

Please refer to this link for more information about this issue cve-2019-5736 : runc vulnerability

Applicability

Note: This document is written for users installing CRI onto Linux. For other operating systems, look for documentation specific to your platform.

You should execute all the commands in this guide as `root`. For example, prefix commands with `sudo`, or become `root` and run the commands as that user.

Cgroup drivers

When systemd is chosen as the init system for a Linux distribution, the init process generates and consumes a root control group (`cgroup`) and acts as a cgroup manager. Systemd has a tight integration with cgroups and will allocate cgroups per process. It's possible to configure your container runtime and the kubelet to use `cgroupfs`. Using `cgroupfs` alongside systemd means that there will then be two different cgroup managers.

Control groups are used to constrain resources that are allocated to processes. A single cgroup manager will simplify the view of what resources are being allocated and will by default have a more consistent view of the available and in-use resources. When we have two managers we end up with two views of those resources. We have seen cases in the field where nodes that are configured to use `cgroupfs` for the kubelet and Docker, and `systemd` for the rest of the processes running on the node becomes unstable under resource pressure.

Changing the settings such that your container runtime and kubelet use `systemd` as the cgroup driver stabilized the system. Please note the `native.cgroupdriver=systemd` option in the Docker setup below.

Caution: Changing the cgroup driver of a Node that has joined a cluster is highly unrecommended. If the kubelet has created Pods using the semantics of one cgroup driver, changing the container runtime to another cgroup driver can cause errors when trying to re-create the PodSandbox for such existing Pods. Restarting the kubelet may not solve such errors. The recommendation is to drain the Node from its workloads, remove it from the cluster and re-join it.

Docker

On each of your machines, install Docker. Version 18.06.2 is recommended, but 1.11, 1.12, 1.13, 17.03 and 18.09 are known to work as well. Keep track of the latest verified Docker version in the Kubernetes release notes.

Use the following commands to install Docker on your system:

- Ubuntu 16.04
- CentOS/RHEL 7.4+

```
# Install Docker CE
## Set up the repository:
### Install packages to allow apt to use a repository over HTTPS
apt-get update && apt-get install apt-transport-https ca-certificates curl software-properti

### Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

### Add Docker apt repository.
add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"

## Install Docker CE.
apt-get update && apt-get install docker-ce=18.06.2~ce~3-0~ubuntu

# Setup daemon.
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF

mkdir -p /etc/systemd/system/docker.service.d

# Restart docker.
systemctl daemon-reload
systemctl restart docker
```

```
# Install Docker CE
## Set up the repository
### Install required packages.
yum install yum-utils device-mapper-persistent-data lvm2

### Add Docker repository.
yum-config-manager \
  --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo

## Install Docker CE.
yum update && yum install docker-ce-18.06.2.ce

## Create /etc/docker directory.
mkdir /etc/docker

# Setup daemon.
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
EOF

mkdir -p /etc/systemd/system/docker.service.d

# Restart Docker
systemctl daemon-reload
systemctl restart docker
```

Refer to the official Docker installation guides for more information.

CRI-O

This section contains the necessary steps to install `CRI-O` as CRI runtime.

Use the following commands to install CRI-O on your system:

Prerequisites

```
modprobe overlay
modprobe br_netfilter

# Setup required sysctl params, these persist across reboots.
cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sysctl --system
```

- Ubuntu 16.04
- CentOS/RHEL 7.4+

```
# Install prerequisites
apt-get update
apt-get install software-properties-common

add-apt-repository ppa:projectatomic/ppa
apt-get update

# Install CRI-O
apt-get install cri-o-1.11
```

```
# Install prerequisites
yum-config-manager --add-repo=https://cbs.centos.org/repos/paas7-crio-311-candidate/x86_64/c

# Install CRI-O
yum install --nogpgcheck cri-o
```

Start CRI-O

```
systemctl start crio
```

Refer to the CRI-O installation guide for more information.

Containerd

This section contains the necessary steps to use `containerd` as CRI runtime.

Use the following commands to install Containerd on your system:

Prerequisites

```
modprobe overlay
modprobe br_netfilter

# Setup required sysctl params, these persist across reboots.
cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sysctl --system
```

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-prope

### Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

### Add Docker apt repository.
add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"

## Install containerd
apt-get update && apt-get install -y containerd.io

# Configure containerd
mkdir -p /etc/containerd
containerd config default > /etc/containerd/config.toml

# Restart containerd
systemctl restart containerd
```

```
# Install containerd
## Set up the repository
### Install required packages
```

```
yum install yum-utils device-mapper-persistent-data lvm2

### Add docker repository
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

## Install containerd
yum update && yum install containerd.io

# Configure containerd
mkdir -p /etc/containerd
containerd config default > /etc/containerd/config.toml

# Restart containerd
systemctl restart containerd
```

systemd

To use the `systemd` cgroup driver, set `plugins.cri.systemd_cgroup = true` in `/etc/containerd/config.toml`. When using kubeadm, manually configure the cgroup driver for kubelet as well.

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.

Create an Issue Edit This Page

Page last modified on April 22, 2019 at 3:35 AM PST by cri: add notes about changing cgroup driver (#13899) (Page History)

Edit This Page

Installing Kubernetes with Digital Rebar Provision (DRP) via KRIB

Overview

This guide helps to install a Kubernetes cluster hosted on bare metal with Digital Rebar Provision using only its Content packages and *kubeadm*.

Digital Rebar Provision (DRP) is an integrated Golang DHCP, bare metal provisioning (PXE/iPXE) and workflow automation platform. While DRP can be used to invoke kubespray, it also offers a self-contained Kubernetes installation known as KRIB (Kubernetes Rebar Integrated Bootstrap).

> **Note:** KRIB is not a *stand-alone* installer: Digital Rebar templates drive a standard *kubeadm* configuration that manages the 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 and others

Creating a cluster

Review Digital Rebar documentation for details about installing the platform.

The Digital Rebar Provision Golang binary should be installed on a Linux-like system with 16 GB of RAM or larger (Packet.net Tiny and Rasberry Pi are also acceptable).

(1/5) Discover servers

Following the Digital Rebar installation, allow one or more servers to boot through the *Sledgehammer* discovery process to register with the API. This will automatically install the Digital Rebar runner and to allow for next steps.

(2/5) Install KRIB Content and Certificate Plugin

Upload the KRIB Content bundle (or build from source) and the Cert Plugin for your DRP platform (e.g.: amd64 Linux v2.4.0). Both are freely available via the RackN UX.

(3/5) Start your cluster deployment

> **Note:** KRIB documentation is dynamically generated from the source and will be more up to date than this guide.

Following the KRIB documentation, create a Profile for your cluster and assign your target servers into the cluster Profile. The Profile must set `krib\cluster-name` and `etcd\cluster-name` Params to be the name of the Profile. Cluster configuration choices can be made by adding additional Params to the Profile; however, safe defaults are provided for all Params.

Once all target servers are assigned to the cluster Profile, start a KRIB installation Workflow by assigning one of the included Workflows to all cluster servers. For example, selecting `krib-live-cluster` will perform an immutable deployment into the Sledgehammer discovery operating system. You may use one of the pre-created read-only Workflows or choose to build your own custom variation.

For basic installs, no further action is required. Advanced users may choose to assign the controllers, etcd servers or other configuration values in the relevant Params.

(4/5) Monitor your cluster deployment

Digital Rebar Provision provides detailed logging and live updates during the installation process. Workflow events are available via a websocket connection or monitoring the Jobs list.

During the installation, KRIB writes cluster configuration data back into the cluster Profile.

(5/5) Access your cluster

The cluster is available for access via *kubectl* once the `krib/cluster-admin-conf` Param has been set. This Param contains the `kubeconfig` information necessary to access the cluster.

For example, if you named the cluster Profile `krib` then the following commands would allow you to connect to the installed cluster from your local terminal.

::

```
drpcli profiles get krib params krib/cluster-admin-conf > admin.conf
export KUBECONFIG=admin.conf
kubectl get nodes
```

The installation continues after the `krib/cluster-admin-conf` is set to install the Kubernetes UI and Helm. You may interact with the cluster as soon as the `admin.conf` file is available.

Cluster operations

KRIB provides additional Workflows to manage your cluster. Please see the KRIB documentation for an updated list of advanced cluster operations.

Scale your cluster

You can add servers into your cluster by adding the cluster Profile to the server and running the appropriate Workflow.

Cleanup your cluster (for developers)

You can reset your cluster and wipe out all configuration and TLS certificates using the `krib-reset-cluster` Workflow on any of the servers in the cluster.

> **Caution:** When running the reset Workflow, be sure not to accidentally target your production cluster!

Feedback

- Slack Channel: #community
- GitHub Issues

Feedback

Was this page helpful?

Yes

No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page

Page last modified on February 12, 2019 at 1:45 PM PST by Update krib.md (#12580) (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

All certificates

If you don't wish to copy these private keys to your API servers, you can generate all certificates yourself.

Required certificates:

Default CN	Parent CA	O (in Subject)	kind	hosts (SAN)
kube-etcd	etcd-ca		server, client	`localhost`, `127.`
kube-etcd-peer	etcd-ca		server, client	`<hostname>`, `<Ho`
kube-etcd-healthcheck-client	etcd-ca		client	
kube-apiserver-etcd-client	etcd-ca	system:masters	client	
kube-apiserver	kubernetes-ca		server	`<hostname>`, `<Ho`
kube-apiserver-kubelet-client	kubernetes-ca	system:masters	client	
front-proxy-client	kubernetes-front-proxy-ca		client	

[1]: `kubernetes`, `kubernetes.default`, `kubernetes.default.svc`, `kubernetes.default.svc.cluster`, `kubernetes.default.svc.cluster.local`

where `kind` maps to one or more of the x509 key usage types:

kind	Key usage
server	digital signature, key encipherment, server auth
client	digital signature, key encipherment, client auth

Certificate paths

Certificates should be placed in a recommended path (as used by kubeadm).
Paths should be specified using the given argument regardless of location.

Default CN	recommend key path	recommended cert path	command	ke
etcd-ca		etcd/ca.crt	kube-apiserver	
etcd-client	apiserver-etcd-client.key	apiserver-etcd-client.crt	kube-apiserver	–e
kubernetes-ca		ca.crt	kube-apiserver	
kube-apiserver	apiserver.key	apiserver.crt	kube-apiserver	–t
apiserver-kubelet-client		apiserver-kubelet-client.crt	kube-apiserver	
front-proxy-ca		front-proxy-ca.crt	kube-apiserver	
front-proxy-client	front-proxy-client.key	front-proxy-client.crt	kube-apiserver	–p
etcd-ca		etcd/ca.crt	etcd	
kube-etcd	etcd/server.key	etcd/server.crt	etcd	–k
kube-etcd-peer	etcd/peer.key	etcd/peer.crt	etcd	–p
etcd-ca		etcd/ca.crt	etcdctl[2]	
kube-etcd-healthcheck-client	etcd/healthcheck-client.key	etcd/healthcheck-client.crt	etcdctl[2]	–k

[2]: For a liveness probe, if self-hosted

Configure certificates for user accounts

You must manually configure these administrator account and service accounts:

filename	credential name	Default CN	O (in Subject)
admin.conf	default-admin	kubernetes-admin	system:masters
kubelet.conf	default-auth	system:node:<nodeName> (see note)	system:nodes
controller-manager.conf	default-controller-manager	system:kube-controller-manager	
scheduler.conf	default-manager	system:kube-scheduler	

> **Note:** The value of `<nodeName>` for `kubelet.conf` **must** match precisely the value of the node name provided by the kubelet as it registers with the apiserver. For further details, read the Node Authorization.

1. For each config, generate an x509 cert/key pair with the given CN and O.
2. Run `kubectl` as follows for each config:

```
KUBECONFIG=<filename> kubectl config set-cluster default-cluster --server=https://<host ip>
KUBECONFIG=<filename> kubectl config set-credentials <credential-name> --client-key <path-to
KUBECONFIG=<filename> kubectl config set-context default-system --cluster default-cluster --
KUBECONFIG=<filename> kubectl config use-context default-system
```

These files are used as follows:

filename	command	comment
admin.conf	kubectl	Configures administrator user for the cluster
kubelet.conf	kubelet	One required for each node in the cluster.
controller-manager.conf	kube-controller-manager	Must be added to manifest in `manifests/kube-control`
scheduler.conf	kube-scheduler	Must be added to manifest in `manifests/kube-schedul`

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 20, 2019 at 6:14 AM PST by Remove mysterious "[1][etcdbug]" (#13266) (Page History)

Edit This Page

Running Kubernetes Locally via Minikube

Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

Minikube Features

- Minikube supports Kubernetes features such as:
  - DNS
  - NodePorts
  - ConfigMaps and Secrets
  - Dashboards
  - Container Runtime: Docker, rkt, CRI-O and containerd
  - Enabling CNI (Container Network Interface)
  - Ingress

Installation

See Installing Minikube.

Quickstart

Here's a brief demo of Minikube usage. If you want to change the VM driver add the appropriate `--vm-driver=xxx` flag to `minikube start`. Minikube supports the following drivers:

- virtualbox
- vmwarefusion
- kvm2 (driver installation)
- kvm (driver installation)
- hyperkit (driver installation)
- xhyve (driver installation) (deprecated)
- hyperv (driver installation) Note that the IP below is dynamic and can change. It can be retrieved with `minikube ip`.
- none (Runs the Kubernetes components on the host and not in a VM. Using this driver requires Docker (docker install) and a Linux environment)

```
minikube start
```

```
Starting local Kubernetes cluster...
Running pre-create checks...
Creating machine...
Starting local Kubernetes cluster...
```

```
kubectl run hello-minikube --image=k8s.gcr.io/echoserver:1.10 --port=8080
```

```
deployment.apps/hello-minikube created
```

```
kubectl expose deployment hello-minikube --type=NodePort
```

```
service/hello-minikube exposed
```

```
# We have now launched an echoserver pod but we have to wait until the pod is up before curl
# via the exposed service.
# To check whether the pod is up and running we can use the following:
kubectl get pod
```

```
NAME                              READY     STATUS              RESTARTS   AGE
hello-minikube-3383150820-vctvh   0/1       ContainerCreating   0          3s
```

```
# We can see that the pod is still being created from the ContainerCreating status
kubectl get pod
```

```
NAME                              READY     STATUS    RESTARTS   AGE
hello-minikube-3383150820-vctvh   1/1       Running   0          13s
```

```
# We can see that the pod is now Running and we will now be able to curl it:
curl $(minikube service hello-minikube --url)
```

```
Hostname: hello-minikube-7c77b68cff-8wdzq

Pod Information:
```

```
    -no pod information available-

Server values:
    server_version=nginx: 1.13.3 - lua: 10008

Request Information:
    client_address=172.17.0.1
    method=GET
    real path=/
    query=
    request_version=1.1
    request_scheme=http
    request_uri=http://192.168.99.100:8080/

Request Headers:
    accept=*/*
    host=192.168.99.100:30674
    user-agent=curl/7.47.0

Request Body:
    -no body in request-
```

```
kubectl delete services hello-minikube
```

```
service "hello-minikube" deleted
```

```
kubectl delete deployment hello-minikube
```

```
deployment.extensions "hello-minikube" deleted
```

```
minikube stop
```

```
Stopping local Kubernetes cluster...
Stopping "minikube"...
```

Alternative Container Runtimes

containerd

To use containerd as the container runtime, run:

```
minikube start \
    --network-plugin=cni \
    --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
```

CRI-O

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
```

rkt container engine

To use rkt as the container runtime run:

```
minikube start \
    --network-plugin=cni \
    --enable-default-cni \
    --container-runtime=rkt
```

This will use an alternative minikube ISO image containing both rkt, and Docker, and enable CNI networking.

Driver plugins

See DRIVERS for details on supported drivers and how to install plugins, if required.

Use local images by re-using the Docker daemon

When using a single VM of Kubernetes, it's really handy to reuse the Minikube's built-in Docker daemon; as this means you don't have to build a docker registry on your host machine and push the image into it - you can just build inside the same docker daemon as minikube which speeds up local experiments. Just make sure you tag your Docker image with something other than 'latest' and use that tag while you pull the image. Otherwise, if you do not specify version of your image, it will be assumed as `:latest`, with pull image policy of `Always` correspondingly, which may eventually result in `ErrImagePull` as you may not have any versions of your Docker image out there in the default docker registry (usually DockerHub) yet.

To be able to work with the docker daemon on your mac/linux host use the `docker-env command` in your shell:

```
eval $(minikube docker-env)
```

You should now be able to use docker on the command line on your host mac/linux machine talking to the docker daemon inside the minikube VM:

```
docker ps
```

On Centos 7, docker may report the following error:

```
Could not read CA certificate "/etc/docker/ca.pem": open /etc/docker/ca.pem: no such file or
```

The fix is to update /etc/sysconfig/docker to ensure that Minikube's environment changes are respected:

```
< DOCKER_CERT_PATH=/etc/docker
---
> if [ -z "${DOCKER_CERT_PATH}" ]; then
>   DOCKER_CERT_PATH=/etc/docker
> fi
```

Remember to turn off the imagePullPolicy:Always, otherwise Kubernetes won't use images you built locally.

Managing your Cluster

Starting a Cluster

The `minikube start` command can be used to start your cluster. This command creates and configures a Virtual Machine that runs a single-node Kubernetes cluster. This command also configures your kubectl installation to communicate with this cluster.

If you are behind a web proxy, you will need to pass this information to the `minikube start` command:

```
https_proxy=<my proxy> minikube start --docker-env http_proxy=<my proxy> --docker-env https_
```

Unfortunately just setting the environment variables will not work.

Minikube will also create a "minikube" context, and set it to default in kubectl. To switch back to this context later, run this command: `kubectl config use-context minikube`.

Specifying the Kubernetes version

You can specify the specific version of Kubernetes for Minikube to use by adding the `--kubernetes-version` string to the `minikube start` command. For example, to run version `v1.7.3`, you would run the following:

```
minikube start --kubernetes-version v1.7.3
```

Configuring Kubernetes

Minikube has a "configurator" feature that allows users to configure the Kubernetes components with arbitrary values. To use this feature, you can use the `--extra-config` flag on the `minikube start` command.

This flag is repeated, so you can pass it several times with several different values to set multiple options.

This flag takes a string of the form `component.key=value`, where `component` is one of the strings from the below list, `key` is a value on the configuration struct and `value` is the value to set.

Valid keys can be found by examining the documentation for the Kubernetes `componentconfigs` for each component. Here is the documentation for each supported configuration:

- kubelet
- apiserver
- proxy
- controller-manager
- etcd
- scheduler

Examples

To change the `MaxPods` setting to 5 on the Kubelet, pass this flag: `--extra-config=kubelet.MaxPods=5`.

This feature also supports nested structs. To change the `LeaderElection.LeaderElect` setting to `true` on the scheduler, pass this flag: `--extra-config=scheduler.LeaderElection.LeaderElect=t`

To set the `AuthorizationMode` on the `apiserver` to `RBAC`, you can use: `--extra-config=apiserver.authorization-mode=RBAC`.

Stopping a Cluster

The `minikube stop` command can be used to stop your cluster. This command shuts down the Minikube Virtual Machine, but preserves all cluster state and data. Starting the cluster again will restore it to its previous state.

Deleting a Cluster

The `minikube delete` command can be used to delete your cluster. This command shuts down and deletes the Minikube Virtual Machine. No data or state is preserved.

Interacting with Your Cluster

Kubectl

The `minikube start` command creates a kubectl context called "minikube". This context contains the configuration to communicate with your Minikube cluster.

Minikube sets this context to default automatically, but if you need to switch back to it in the future, run:

`kubectl config use-context minikube`,

Or pass the context on each command like this: `kubectl get pods --context=minikube`.

Dashboard

To access the Kubernetes Dashboard, run this command in a shell after starting Minikube to get the address:

```
minikube dashboard
```

Services

To access a service exposed via a node port, run this command in a shell after starting Minikube to get the address:

```
minikube service [-n NAMESPACE] [--url] NAME
```

Networking

The Minikube VM is exposed to the host system via a host-only IP address, that can be obtained with the `minikube ip` command. Any services of type `NodePort` can be accessed over that IP address, on the NodePort.

To determine the NodePort for your service, you can use a `kubectl` command like this:

```
kubectl get service $SERVICE --output='jsonpath="{.spec.ports[0].nodePort}"'
```

Persistent Volumes

Minikube supports PersistentVolumes of type `hostPath`. These PersistentVolumes are mapped to a directory inside the Minikube VM.

The Minikube VM boots into a tmpfs, so most directories will not be persisted across reboots (`minikube stop`). However, Minikube is configured to persist files stored under the following host directories:

- `/data`
- `/var/lib/minikube`
- `/var/lib/docker`

Here is an example PersistentVolume config to persist data in the `/data` directory:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 5Gi
  hostPath:
    path: /data/pv0001/
```

Mounted Host Folders

Some drivers will mount a host folder within the VM so that you can easily share files between the VM and host. These are not configurable at the moment and different for the driver and OS you are using.

> **Note:** Host folder sharing is not implemented in the KVM driver yet.

Driver	OS	HostFolder	VM
VirtualBox	Linux	/home	/hosthome
VirtualBox	macOS	/Users	/Users
VirtualBox	Windows	C://Users	/c/Users
VMware Fusion	macOS	/Users	/Users
Xhyve	macOS	/Users	/Users

Private Container Registries

To access a private container registry, follow the steps on this page.

We recommend you use `ImagePullSecrets`, but if you would like to configure access on the Minikube VM you can place the `.dockercfg` in the `/home/docker` directory or the `config.json` in the `/home/docker/.docker` directory.

Add-ons

In order to have Minikube properly start or restart custom addons, place the addons you wish to be launched with Minikube in the `~/.minikube/addons` directory. Addons in this folder will be moved to the Minikube VM and launched each time Minikube is started or restarted.

Using Minikube with an HTTP Proxy

Minikube creates a Virtual Machine that includes Kubernetes and a Docker daemon. When Kubernetes attempts to schedule containers using Docker, the Docker daemon may require external network access to pull containers.

If you are behind an HTTP proxy, you may need to supply Docker with the proxy settings. To do this, pass the required environment variables as flags during `minikube start`.

For example:

```
minikube start --docker-env http_proxy=http://$YOURPROXY:PORT \
               --docker-env https_proxy=https://$YOURPROXY:PORT
```

If your Virtual Machine address is 192.168.99.100, then chances are your proxy settings will prevent `kubectl` from directly reaching it. To by-pass proxy configuration for this IP address, you should modify your no_proxy settings. You can do so with:

```
export no_proxy=$no_proxy,$(minikube ip)
```

Known Issues

- Features that require a Cloud Provider will not work in Minikube. These include:
  - LoadBalancers
- Features that require multiple nodes. These include:
  - Advanced scheduling policies

Design

Minikube uses libmachine for provisioning VMs, and kubeadm to provision a Kubernetes cluster.

For more information about Minikube, see the proposal.

Additional Links

- **Goals and Non-Goals**: For the goals and non-goals of the Minikube project, please see our roadmap.
- **Development Guide**: See CONTRIBUTING.md for an overview of how to send pull requests.
- **Building Minikube**: For instructions on how to build/test Minikube from source, see the build guide.
- **Adding a New Dependency**: For instructions on how to add a new dependency to Minikube, see the adding dependencies guide.
- **Adding a New Addon**: For instructions on how to add a new addon for Minikube, see the adding an addon guide.
- **MicroK8s**: Linux users wishing to avoid running a virtual machine may consider MicroK8s as an alternative.

Community

Contributions, questions, and comments are all welcomed and encouraged! Minikube developers hang out on Slack in the #minikube channel (get an invitation here). We also have the kubernetes-dev Google Groups mailing list. If you are posting to the list please prefix your subject with “minikube: “.

Feedback

Was this page helpful?

Yes

No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Create an Issue Edit This Page

Page last modified on March 13, 2019 at 11:35 PM PST by Fix typos (#13155) (Page History)

Edit This Page

Validate Node Setup

Node Conformance Test

Node conformance test is a containerized test framework that provides a system verification and functionality test for a node. The test validates whether the node meets the minimum requirements for Kubernetes; a node that passes the test is qualified to join a Kubernetes cluster.

Limitations

In Kubernetes version 1.5, node conformance test has the following limitations:

- Node conformance test only supports Docker as the container runtime.

Node Prerequisite

To run node conformance test, a node must satisfy the same prerequisites as a standard Kubernetes node. At a minimum, the node should have the following daemons installed:

- Container Runtime (Docker)

- Kubelet

Running Node Conformance Test

To run the node conformance test, perform the following steps:

1. Point your Kubelet to localhost `--api-servers="http://localhost:8080"`, because the test framework starts a local master to test Kubelet. There are some other Kubelet flags you may care:
   - `--pod-cidr`: If you are using `kubenet`, you should specify an arbitrary CIDR to Kubelet, for example `--pod-cidr=10.180.0.0/24`.
   - `--cloud-provider`: If you are using `--cloud-provider=gce`, you should remove the flag to run the test.
2. Run the node conformance test with command:

```
# $CONFIG_DIR is the pod manifest path of your Kubelet.
# $LOG_DIR is the test output path.
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/result \
  k8s.gcr.io/node-test:0.2
```

Running Node Conformance Test for Other Architectures

Kubernetes also provides node conformance test docker images for other architectures:

Arch	Image
amd64	node-test-amd64
arm	node-test-arm
arm64	node-test-arm64

Running Selected Test

To run specific tests, overwrite the environment variable `FOCUS` with the regular expression of tests you want to run.

```
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs:ro -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/result \
  -e FOCUS=MirrorPod \ # Only run MirrorPod test
  k8s.gcr.io/node-test:0.2
```

To skip specific tests, overwrite the environment variable `SKIP` with the regular expression of tests you want to skip.

```
sudo docker run -it --rm --privileged --net=host \
  -v /:/rootfs:ro -v $CONFIG_DIR:$CONFIG_DIR -v $LOG_DIR:/var/result \
  -e SKIP=MirrorPod \ # Run all conformance tests but skip MirrorPod test
  k8s.gcr.io/node-test:0.2
```

Node conformance test is a containerized version of node e2e test. By default, it runs all conformance tests.

Theoretically, you can run any node e2e test if you configure the container and mount required volumes properly. But **it is strongly recommended to only run conformance test**, because it requires much more complex configuration to run non-conformance test.

Caveats

- The test leaves some docker images on the node, including the node conformance test image and images of containers used in the functionality test.
- The test leaves dead containers on the node. These containers are created during the functionality test.

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 19, 2018 at 11:37 AM PST by Remove or move topics under docs/admin. (#9140) (Page History)