kubeadm kubeconfig

`kubeadm kubeconfig` provides utilities for managing kubeconfig files.

For examples on how to use `kubeadm kubeconfig user` see Generating kubeconfig files for additional users.

kubeadm kubeconfig

- overview

Synopsis

Kubeconfig file utilities.

Options

-h, --help
  help for kubeconfig

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm kubeconfig user

This command can be used to output a kubeconfig file for an additional user.

- user

Synopsis

Output a kubeconfig file for an additional user.

```
kubeadm kubeconfig user [flags]
```

Examples

```
  # Output a kubeconfig file for an additional user named foo
  kubeadm kubeconfig user --client-name=foo

  # Output a kubeconfig file for an additional user named foo using a kubeadm config file bar
  kubeadm kubeconfig user --client-name=foo --config=bar
```

Options

--client-name string
  The name of user. It will be used as the CN if client certificates are created
--config string
  Path to a kubeadm configuration file.
-h, --help
  help for user
--org strings
  The organizations of the client certificate. It will be used as the O if client certificates are created
--token string
  The token that should be used as the authentication mechanism for this kubeconfig, instead of client certificates
--validity-period duration Default: 8760h0m0s
  The validity period of the client certificate. It is an offset from the current time.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

---

kubeadm certs

`kubeadm certs` provides utilities for managing certificates. For more details on how these commands can be used, see Certificate Management with kubeadm.

kubeadm certs

A collection of operations for operating Kubernetes certificates.

- overview

Synopsis

Commands related to handling Kubernetes certificates

```
kubeadm certs [flags]
```

Options

-h, --help
  help for certs

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm certs renew

You can renew all Kubernetes certificates using the `all` subcommand or renew them selectively. For more details see Manual certificate renewal.

- renew
- all
- admin.conf
- apiserver-etcd-client
- apiserver-kubelet-client
- apiserver
- controller-manager.conf
- etcd-healthcheck-client
- etcd-peer
- etcd-server
- front-proxy-client
- scheduler.conf
- super-admin.conf

Synopsis

Renew certificates for a Kubernetes cluster

```
kubeadm certs renew [flags]
```

Options

-h, --help
  help for renew

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Renew all available certificates

Synopsis

Renew all known certificates necessary to run the control plane. Renewals are run unconditionally, regardless of expiration date. Renewals can also be run individually for more control.

```
kubeadm certs renew all [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save the certificates
--config string
  Path to a kubeadm configuration file.

-h, --help
  help for all
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate embedded in the kubeconfig file for the admin to use and for kubeadm itself.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew admin.conf [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save the certificates
--config string
  Path to a kubeadm configuration file.
-h, --help
  help for admin.conf
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate the apiserver uses to access etcd.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew apiserver-etcd-client [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save the certificates
--config string
  Path to a kubeadm configuration file.
-h, --help
  help for apiserver-etcd-client
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate for the API server to connect to kubelet.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew apiserver-kubelet-client [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save the certificates
--config string
  Path to a kubeadm configuration file.
-h, --help
  help for apiserver-kubelet-client
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate for serving the Kubernetes API.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew apiserver [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save the certificates
--config string
  Path to a kubeadm configuration file.
-h, --help
  help for apiserver
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate embedded in the kubeconfig file for the controller manager to use.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew controller-manager.conf [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for controller-manager.conf
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate for liveness probes to healthcheck etcd.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew etcd-healthcheck-client [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for etcd-healthcheck-client
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate for etcd nodes to communicate with each other.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew etcd-peer [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for etcd-peer

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate for serving etcd.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew etcd-server [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for etcd-server
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate for the front proxy client.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew front-proxy-client [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for front-proxy-client
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate embedded in the kubeconfig file for the scheduler manager to use.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew scheduler.conf [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for scheduler.conf
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Renew the certificate embedded in the kubeconfig file for the super-admin.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm certs renew super-admin.conf [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for super-admin.conf
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm certs certificate-key

This command can be used to generate a new control-plane certificate key. The key can be passed as `--certificate-key` to `kubeadm init` and `kubeadm join` to enable the automatic copy of certificates when joining additional control-plane nodes.

- certificate-key

Generate certificate keys

Synopsis

This command will print out a secure randomly-generated certificate key that can be used with the "init" command.

You can also use "kubeadm init --upload-certs" without specifying a certificate key and it will generate and print one for you.

```
kubeadm certs certificate-key [flags]
```

Options

-h, --help
  help for certificate-key

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm certs check-expiration

This command checks expiration for the certificates in the local PKI managed by kubeadm. For more details see Check certificate expiration.

- check-expiration

Check certificates expiration for a Kubernetes cluster

Synopsis

Checks expiration for the certificates in the local PKI managed by kubeadm.

```
kubeadm certs check-expiration [flags]
```

Options

--allow-missing-template-keys Default: true
  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save the certificates
--config string
  Path to a kubeadm configuration file.
-h, --help
  help for check-expiration
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
-o, --output string Default: "text"
  Output format. One of: textljsonlyamllgo-templatelgo-template-fileltemplateltemplatefileljsonpathljsonpath-as-jsonljsonpath-file.
--show-managed-fields
  If true, keep the managedFields when printing objects in JSON or YAML format.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm certs generate-csr

This command can be used to generate keys and CSRs for all control-plane certificates and kubeconfig files. The user can then sign the CSRs with a CA of their choice. To read more information on how to use the command see Signing certificate signing requests (CSR) generated by kubeadm.

- generate-csr

Generate keys and certificate signing requests

Synopsis

Generates keys and certificate signing requests (CSRs) for all the certificates required to run the control plane. This command also generates partial kubeconfig files with private key data in the "users > user > client-key-data" field, and for each kubeconfig file an accompanying ".csr" file is created.

This command is designed for use in Kubeadm External CA Mode. It generates CSRs which you can then submit to your external certificate authority for signing.

The PEM encoded signed certificates should then be saved alongside the key files, using ".crt" as the file extension, or in the case of kubeconfig files, the PEM encoded signed certificate should be base64 encoded and added to the kubeconfig file in the "users > user > client-certificate-data" field.

```
kubeadm certs generate-csr [flags]
```

Examples

```
# The following command will generate keys and CSRs for all control-plane certificates and kubeconfig files:
kubeadm certs generate-csr --kubeconfig-dir /tmp/etc-k8s --cert-dir /tmp/etc-k8s/pki
```

Options

--cert-dir string
The path where to save the certificates
--config string
Path to a kubeadm configuration file.
-h, --help
help for generate-csr
--kubeconfig-dir string Default: "/etc/kubernetes"
The path where to save the kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`

---

kubeadm token

Bootstrap tokens are used for establishing bidirectional trust between a node joining the cluster and a control-plane node, as described in authenticating with bootstrap tokens.

`kubeadm init` creates an initial token with a 24-hour TTL. The following commands allow you to manage such a token and also to create and manage new ones.

kubeadm token create

Create bootstrap tokens on the server

Synopsis

This command will create a bootstrap token for you. You can specify the usages for this token, the "time to live" and an optional human friendly description.

The [token] is the actual token to write. This should be a securely generated random token of the form "[a-z0-9]{6}.[a-z0-9]{16}". If no [token] is given, kubeadm will generate a random token instead.

```
kubeadm token create [token]
```

Options

--certificate-key string
When used together with '--print-join-command', print the full 'kubeadm join' flag needed to join the cluster as a control-plane. To create a new certificate key you must use 'kubeadm init phase upload-certs --upload-certs'.
--config string
Path to a kubeadm configuration file.
--description string
A human friendly description of how this token is used.
--groups strings Default: "system:bootstrappers:kubeadm:default-node-token"
Extra groups that this token will authenticate as when used for authentication. Must match "\Asystem:bootstrappers:[a-z0-9:-]{0,255}[a-z0-9]\z"
-h, --help
help for create
--print-join-command
Instead of printing only the token, print the full 'kubeadm join' flag needed to join the cluster using the token.
--ttl duration Default: 24h0m0s
The duration before the token is automatically deleted (e.g. 1s, 2m, 3h). If set to '0', the token will never expire
--usages strings Default: "signing,authentication"
Describes the ways in which this token can be used. You can pass --usages multiple times or provide a comma separated list of options. Valid options: [signing,authentication]

Options inherited from parent commands

--dry-run
  Whether to enable dry-run mode or not
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm token delete

Delete bootstrap tokens on the server

Synopsis

This command will delete a list of bootstrap tokens for you.

The [token-value] is the full Token of the form "[a-z0-9]{6}.[a-z0-9]{16}" or the Token ID of the form "[a-z0-9]{6}" to delete.

```
kubeadm token delete [token-value] ...
```

Options

-h, --help
  help for delete

Options inherited from parent commands

--dry-run
  Whether to enable dry-run mode or not
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm token generate

Generate and print a bootstrap token, but do not create it on the server

Synopsis

This command will print out a randomly-generated bootstrap token that can be used with the "init" and "join" commands.

You don't have to use this command in order to generate a token. You can do so yourself as long as it is in the format "[a-z0-9]{6}.[a-z0-9]{16}". This command is provided for convenience to generate tokens in the given format.

You can also use "kubeadm init" without specifying a token and it will generate and print one for you.

```
kubeadm token generate [flags]
```

Options

-h, --help
  help for generate

Options inherited from parent commands

--dry-run
  Whether to enable dry-run mode or not
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm token list

List bootstrap tokens on the server

Synopsis

This command will list all bootstrap tokens for you.

```
kubeadm token list [flags]
```

Options

--allow-missing-template-keys Default: true
  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
-h, --help
  help for list
-o, --output string Default: "text"
  Output format. One of: text|json|yaml|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file.
--show-managed-fields
  If true, keep the managedFields when printing objects in JSON or YAML format.

Options inherited from parent commands

--dry-run
  Whether to enable dry-run mode or not
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

What's next

- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster

---

kubeadm init phase

`kubeadm init phase` enables you to invoke atomic steps of the bootstrap process. Hence, you can let kubeadm do some of the work and you can fill in the gaps if you wish to apply customization.

`kubeadm init phase` is consistent with the kubeadm init workflow, and behind the scene both use the same code.

kubeadm init phase preflight

Using this command you can execute preflight checks on a control-plane node.

- preflight

Synopsis

Run pre-flight checks for kubeadm init.

```
kubeadm init phase preflight [flags]
```

Examples

```
  # Run pre-flight checks for kubeadm init using a config file.
  kubeadm init phase preflight --config kubeadm-config.yaml
```

Options

--config string
  Path to a kubeadm configuration file.
--cri-socket string
  Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for preflight
--ignore-preflight-errors strings
  A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.
--image-repository string Default: "registry.k8s.io"
  Choose a container registry to pull control plane images from

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase kubelet-start

This phase will write the kubelet configuration file and environment file and then start the kubelet.

- kubelet-start

Write kubelet settings and (re)start the kubelet

Synopsis

Write a file with KubeletConfiguration and an environment file with node specific kubelet settings, and then (re)start kubelet.

```
kubeadm init phase kubelet-start [flags]
```

Examples

```
  # Writes a dynamic environment file with kubelet flags from a InitConfiguration file.
  kubeadm init phase kubelet-start --config config.yaml
```

Options

--config string
  Path to a kubeadm configuration file.
--cri-socket string
  Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for kubelet-start
--image-repository string Default: "registry.k8s.io"
  Choose a container registry to pull control plane images from
--node-name string
  Specify the node name.
--patches string
  Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase certs

Can be used to create all required certificates by kubeadm.

- certs
- all
- ca
- apiserver
- apiserver-kubelet-client
- front-proxy-ca
- front-proxy-client
- etcd-ca
- etcd-server
- etcd-peer
- healthcheck-client
- apiserver-etcd-client
- sa

Synopsis

Certificate generation

```
kubeadm init phase certs [flags]
```

Options

-h, --help
  help for certs

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate all certificates

```
kubeadm init phase certs all [flags]
```

Options

--apiserver-advertise-address string
  The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-cert-extra-sans strings
  Optional extra Subject Alternative Names (SANs) to use for the API Server serving certificate. Can be both IP addresses and DNS names.
--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for all
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
--service-cidr string Default: "10.96.0.0/12"
  Use alternative range of IP address for service VIPs.
--service-dns-domain string Default: "cluster.local"
  Use alternative domain for services, e.g. "myorg.internal".

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the self-signed Kubernetes CA to provision identities for other Kubernetes components, and save them into ca.crt and ca.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs ca [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for ca
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the certificate for serving the Kubernetes API, and save them into apiserver.crt and apiserver.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs apiserver [flags]
```

Options

--apiserver-advertise-address string
  The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-cert-extra-sans strings
  Optional extra Subject Alternative Names (SANs) to use for the API Server serving certificate. Can be both IP addresses and DNS names.
--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for apiserver
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
--service-cidr string Default: "10.96.0.0/12"
  Use alternative range of IP address for service VIPs.
--service-dns-domain string Default: "cluster.local"
  Use alternative domain for services, e.g. "myorg.internal".

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the certificate for the API server to connect to kubelet, and save them into apiserver-kubelet-client.crt and apiserver-kubelet-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs apiserver-kubelet-client [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for apiserver-kubelet-client
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the self-signed CA to provision identities for front proxy, and save them into front-proxy-ca.crt and front-proxy-ca.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs front-proxy-ca [flags]
```

Options

```
--cert-dir string     Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for front-proxy-ca
--kubernetes-version string     Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
```

Options inherited from parent commands

```
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.
```

Synopsis

Generate the certificate for the front proxy client, and save them into front-proxy-client.crt and front-proxy-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs front-proxy-client [flags]
```

Options

```
--cert-dir string     Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for front-proxy-client
--kubernetes-version string     Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
```

Options inherited from parent commands

```
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.
```

Synopsis

Generate the self-signed CA to provision identities for etcd, and save them into etcd/ca.crt and etcd/ca.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs etcd-ca [flags]
```

Options

```
--cert-dir string     Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for etcd-ca
--kubernetes-version string     Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
```

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the certificate for serving etcd, and save them into etcd/server.crt and etcd/server.key files.

Default SANs are localhost, 127.0.0.1, 127.0.0.1, ::1

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs etcd-server [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for etcd-server
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the certificate for etcd nodes to communicate with each other, and save them into etcd/peer.crt and etcd/peer.key files.

Default SANs are localhost, 127.0.0.1, 127.0.0.1, ::1

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs etcd-peer [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for etcd-peer
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the certificate for liveness probes to healthcheck etcd, and save them into etcd/healthcheck-client.crt and etcd/healthcheck-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs etcd-healthcheck-client [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for etcd-healthcheck-client
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the certificate the apiserver uses to access etcd, and save them into apiserver-etcd-client.crt and apiserver-etcd-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs apiserver-etcd-client [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for apiserver-etcd-client
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Generate a private key for signing service account tokens along with its public key

Synopsis

Generate the private key for signing service account tokens along with its public key, and save them into sa.key and sa.pub files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

```
kubeadm init phase certs sa [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for sa
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase kubeconfig

You can create all required kubeconfig files by calling the `all` subcommand or call them individually.

- kubeconfig
- all
- admin
- kubelet
- controller-manager
- scheduler
- super-admin

Synopsis

Generate all kubeconfig files necessary to establish the control plane and the admin kubeconfig file

```
kubeadm init phase kubeconfig [flags]
```

Options

-h, --help
The help for kubeconfig

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate all kubeconfig files

```
kubeadm init phase kubeconfig all [flags]
```

Options

--apiserver-advertise-address string
The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
Port for the API Server to bind to.
--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save and store the certificates.
--config string
Path to a kubeadm configuration file.
--control-plane-endpoint string
Specify a stable IP address or DNS name for the control plane.
--dry-run
Don't apply any changes; just output what would be done.
-h, --help
help for all
--kubeconfig-dir string Default: "/etc/kubernetes"
The path where to save the kubeconfig file.
--kubernetes-version string Default: "stable-1"
Choose a specific Kubernetes version for the control plane.
--node-name string
Specify the node name.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Generate a kubeconfig file for the admin to use and for kubeadm itself

Synopsis

Generate the kubeconfig file for the admin and for kubeadm itself, and save it to admin.conf file.

```
kubeadm init phase kubeconfig admin [flags]
```

Options

```
--apiserver-advertise-address string
  The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32     Default: 6443
  Port for the API Server to bind to.
--cert-dir string     Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for admin
--kubeconfig-dir string     Default: "/etc/kubernetes"
  The path where to save the kubeconfig file.
--kubernetes-version string     Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
```

Options inherited from parent commands

```
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.
```

Generate a kubeconfig file for the kubelet to use *only* for cluster bootstrapping purposes

Synopsis

Generate the kubeconfig file for the kubelet to use and save it to kubelet.conf file.

Please note that this should *only* be used for cluster bootstrapping purposes. After your control plane is up, you should request all kubelet credentials from the CSR API.

```
kubeadm init phase kubeconfig kubelet [flags]
```

Options

```
--apiserver-advertise-address string
  The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32     Default: 6443
  Port for the API Server to bind to.
--cert-dir string     Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for kubelet
--kubeconfig-dir string     Default: "/etc/kubernetes"
  The path where to save the kubeconfig file.
--kubernetes-version string     Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
--node-name string
  Specify the node name.
```

Options inherited from parent commands

```
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.
```

Generate a kubeconfig file for the controller manager to use

Synopsis

Generate the kubeconfig file for the controller manager to use and save it to controller-manager.conf file

```
kubeadm init phase kubeconfig controller-manager [flags]
```

Options

--apiserver-advertise-address string
  The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
  Port for the API Server to bind to.
--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for controller-manager
--kubeconfig-dir string Default: "/etc/kubernetes"
  The path where to save the kubeconfig file.
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Generate a kubeconfig file for the scheduler to use

Synopsis

Generate the kubeconfig file for the scheduler to use and save it to scheduler.conf file.

```
kubeadm init phase kubeconfig scheduler [flags]
```

Options

--apiserver-advertise-address string
  The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
  Port for the API Server to bind to.
--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for scheduler
--kubeconfig-dir string Default: "/etc/kubernetes"
  The path where to save the kubeconfig file.
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate a kubeconfig file for the super-admin, and save it to super-admin.conf file.

```
kubeadm init phase kubeconfig super-admin [flags]
```

Options

--apiserver-advertise-address string
  The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
  Port for the API Server to bind to.
--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for super-admin
--kubeconfig-dir string Default: "/etc/kubernetes"
  The path where to save the kubeconfig file.
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase control-plane

Using this phase you can create all required static Pod files for the control plane components.

- control-plane
- all
- apiserver
- controller-manager
- scheduler

Synopsis

Generate all static Pod manifest files necessary to establish the control plane

```
kubeadm init phase control-plane [flags]
```

Options

-h, --help
  help for control-plane

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate all static Pod manifest files

```
kubeadm init phase control-plane all [flags]
```

Examples

```
  # Generates all static Pod manifest files for control plane components,
  # functionally equivalent to what is generated by kubeadm init.
  kubeadm init phase control-plane all
```

```
# Generates all static Pod manifest files using options read from a configuration file.
kubeadm init phase control-plane all --config config.yaml
```

Options

--apiserver-advertise-address string
The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
Port for the API Server to bind to.
--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save and store the certificates.
--config string
Path to a kubeadm configuration file.
--control-plane-endpoint string
Specify a stable IP address or DNS name for the control plane.
--dry-run
Don't apply any changes; just output what would be done.
--feature-gates string
A set of key=value pairs that describe feature gates for various features. Options are:
ControlPlaneKubeletLocalMode=true|false (BETA - default=true)
NodeLocalCRISocket=true|false (BETA - default=true)
PublicKeysECDSA=true|false (DEPRECATED - default=false)
RootlessControlPlane=true|false (ALPHA - default=false)
WaitForAllControlPlaneComponents=true|false (default=true)
-h, --help
help for all
--image-repository string Default: "registry.k8s.io"
Choose a container registry to pull control plane images from
--kubernetes-version string Default: "stable-1"
Choose a specific Kubernetes version for the control plane.
--patches string
Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.
--pod-network-cidr string
Specify range of IP addresses for the pod network. If set, the control plane will automatically allocate CIDRs for every node.
--service-cidr string Default: "10.96.0.0/12"
Use alternative range of IP address for service VIPs.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generates the kube-apiserver static Pod manifest

```
kubeadm init phase control-plane apiserver [flags]
```

Options

--apiserver-advertise-address string
The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
Port for the API Server to bind to.
--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save and store the certificates.
--config string
Path to a kubeadm configuration file.
--control-plane-endpoint string
Specify a stable IP address or DNS name for the control plane.
--dry-run
Don't apply any changes; just output what would be done.
--feature-gates string

A set of key=value pairs that describe feature gates for various features. Options are:
ControlPlaneKubeletLocalMode=true|false (BETA - default=true)
NodeLocalCRISocket=true|false (BETA - default=true)
PublicKeysECDSA=true|false (DEPRECATED - default=false)
RootlessControlPlane=true|false (ALPHA - default=false)
WaitForAllControlPlaneComponents=true|false (default=true)

-h, --help
help for apiserver

--image-repository string Default: "registry.k8s.io"
Choose a container registry to pull control plane images from

--kubernetes-version string Default: "stable-1"
Choose a specific Kubernetes version for the control plane.

--patches string
Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

--service-cidr string Default: "10.96.0.0/12"
Use alternative range of IP address for service VIPs.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generates the kube-controller-manager static Pod manifest

```
kubeadm init phase control-plane controller-manager [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save and store the certificates.

--config string
Path to a kubeadm configuration file.

--dry-run
Don't apply any changes; just output what would be done.

-h, --help
help for controller-manager

--image-repository string Default: "registry.k8s.io"
Choose a container registry to pull control plane images from

--kubernetes-version string Default: "stable-1"
Choose a specific Kubernetes version for the control plane.

--patches string
Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

--pod-network-cidr string
Specify range of IP addresses for the pod network. If set, the control plane will automatically allocate CIDRs for every node.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generates the kube-scheduler static Pod manifest

```
kubeadm init phase control-plane scheduler [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"

The path where to save and store the certificates.

--config string

Path to a kubeadm configuration file.

--dry-run

Don't apply any changes; just output what would be done.

-h, --help

help for scheduler

--image-repository string Default: "registry.k8s.io"

Choose a container registry to pull control plane images from

--kubernetes-version string Default: "stable-1"

Choose a specific Kubernetes version for the control plane.

--patches string

Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase etcd

Use the following phase to create a local etcd instance based on a static Pod file.

- etcd
- local

Synopsis

Generate static Pod manifest file for local etcd

```
kubeadm init phase etcd [flags]
```

Options

-h, --help

help for etcd

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the static Pod manifest file for a local, single-node local etcd instance

```
kubeadm init phase etcd local [flags]
```

Examples

```
# Generates the static Pod manifest file for etcd, functionally
# equivalent to what is generated by kubeadm init.
kubeadm init phase etcd local

# Generates the static Pod manifest file for etcd using options
# read from a configuration file.
kubeadm init phase etcd local --config config.yaml
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"

The path where to save and store the certificates.

--config string

Path to a kubeadm configuration file.

--dry-run

Don't apply any changes; just output what would be done.

-h, --help

help for local

--image-repository string Default: "registry.k8s.io"
Choose a container registry to pull control plane images from

--patches string
Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase upload-config

You can use this command to upload the kubeadm configuration to your cluster. Alternatively, you can use kubeadm config.

- upload-config
- all
- kubeadm
- kubelet

Synopsis

Upload the kubeadm and kubelet configuration to a ConfigMap

```
kubeadm init phase upload-config [flags]
```

Options

-h, --help
help for upload-config

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upload all configuration to a config map

```
kubeadm init phase upload-config all [flags]
```

Options

--config string
Path to a kubeadm configuration file.

--cri-socket string
Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.

--dry-run
Don't apply any changes; just output what would be done.

-h, --help
help for all

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upload the kubeadm ClusterConfiguration to a ConfigMap called kubeadm-config in the kube-system namespace. This enables correct configuration of system components and a seamless user experience when upgrading.

Alternatively, you can use kubeadm config.

```
kubeadm init phase upload-config kubeadm [flags]
```

Examples

```
  # upload the configuration of your cluster
  kubeadm init phase upload-config kubeadm --config=myConfig.yaml
```

Options

--config string
  Path to a kubeadm configuration file.
--cri-socket string
  Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for kubeadm
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Upload the kubelet component config to a ConfigMap

Synopsis

Upload the kubelet configuration extracted from the kubeadm InitConfiguration object to a kubelet-config ConfigMap in the cluster

```
kubeadm init phase upload-config kubelet [flags]
```

Examples

```
  # Upload the kubelet configuration from the kubeadm Config file to a ConfigMap in the cluster.
  kubeadm init phase upload-config kubelet --config kubeadm.yaml
```

Options

--config string
  Path to a kubeadm configuration file.
--cri-socket string
  Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for kubelet
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase upload-certs

Use the following phase to upload control-plane certificates to the cluster. By default the certs and encryption key expire after two hours.

- upload-certs

Upload certificates to kubeadm-certs

Synopsis

Upload control plane certificates to the kubeadm-certs Secret

```
kubeadm init phase upload-certs [flags]
```

Options

--certificate-key string
  Key used to encrypt the control-plane certificates in the kubeadm-certs Secret. The certificate key is a hex encoded string that is an AES key of size 32 bytes.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for upload-certs
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--skip-certificate-key-print
  Don't print the key used to encrypt the control-plane certificates.
--upload-certs
  Upload control-plane certificates to the kubeadm-certs Secret.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase mark-control-plane

Use the following phase to label and taint the node as a control plane node.

- mark-control-plane

Synopsis

Mark a node as a control-plane

```
kubeadm init phase mark-control-plane [flags]
```

Examples

```
  # Applies control-plane label and taint to the current node, functionally equivalent to what executed by kubeadm init.
  kubeadm init phase mark-control-plane --config config.yaml

  # Applies control-plane label and taint to a specific node
  kubeadm init phase mark-control-plane --node-name myNode
```

Options

--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for mark-control-plane
--node-name string
  Specify the node name.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase bootstrap-token

Use the following phase to configure bootstrap tokens.

- bootstrap-token

Generates bootstrap tokens used to join a node to a cluster

Synopsis

Bootstrap tokens are used for establishing bidirectional trust between a node joining the cluster and a control-plane node.

This command makes all the configurations required to make bootstrap tokens works and then creates an initial token.

```
kubeadm init phase bootstrap-token [flags]
```

Examples

```
  # Make all the bootstrap token configurations and create an initial token, functionally
  # equivalent to what generated by kubeadm init.
  kubeadm init phase bootstrap-token
```

Options

--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for bootstrap-token
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--skip-token-print
  Skip printing of the default bootstrap token generated by 'kubeadm init'.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase kubelet-finalize

Use the following phase to update settings relevant to the kubelet after TLS bootstrap. You can use the `all` subcommand to run all `kubelet-finalize` phases.

- kubelet-finalize
- kubelet-finalize-all
- kubelet-finalize-enable-client-cert-rotation

Synopsis

Updates settings relevant to the kubelet after TLS bootstrap

```
kubeadm init phase kubelet-finalize [flags]
```

Examples

```
  # Updates settings relevant to the kubelet after TLS bootstrap"
  kubeadm init phase kubelet-finalize all --config
```

Options

-h, --help
  help for kubelet-finalize

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Run all kubelet-finalize phases

```
kubeadm init phase kubelet-finalize all [flags]
```

Examples

```
  # Updates settings relevant to the kubelet after TLS bootstrap"
  kubeadm init phase kubelet-finalize all --config
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path where to save and store the certificates.
--config string

Path to a kubeadm configuration file.

--dry-run
Don't apply any changes; just output what would be done.

-h, --help
help for all

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Enable kubelet client certificate rotation

```
kubeadm init phase kubelet-finalize enable-client-cert-rotation [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
The path where to save and store the certificates.

--config string
Path to a kubeadm configuration file.

--dry-run
Don't apply any changes; just output what would be done.

-h, --help
help for enable-client-cert-rotation

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm init phase addon

You can install all the available addons with the `all` subcommand, or install them selectively.

- addon
- all
- coredns
- kube-proxy

Synopsis

Install required addons for passing conformance tests

```
kubeadm init phase addon [flags]
```

Options

-h, --help
help for addon

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Install all the addons

```
kubeadm init phase addon all [flags]
```

Options

--apiserver-advertise-address string
The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.

--apiserver-bind-port int32 Default: 6443
Port for the API Server to bind to.

--config string
  Path to a kubeadm configuration file.
--control-plane-endpoint string
  Specify a stable IP address or DNS name for the control plane.
--dry-run
  Don't apply any changes; just output what would be done.
--feature-gates string
  A set of key=value pairs that describe feature gates for various features. Options are:
  ControlPlaneKubeletLocalMode=truelfalse (BETA - default=true)
  NodeLocalCRISocket=truelfalse (BETA - default=true)
  PublicKeysECDSA=truelfalse (DEPRECATED - default=false)
  RootlessControlPlane=truelfalse (ALPHA - default=false)
  WaitForAllControlPlaneComponents=truelfalse (default=true)
-h, --help
  help for all
--image-repository string Default: "registry.k8s.io"
  Choose a container registry to pull control plane images from
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
--pod-network-cidr string
  Specify range of IP addresses for the pod network. If set, the control plane will automatically allocate CIDRs for every node.
--service-cidr string Default: "10.96.0.0/12"
  Use alternative range of IP address for service VIPs.
--service-dns-domain string Default: "cluster.local"
  Use alternative domain for services, e.g. "myorg.internal".

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Install the CoreDNS addon to a Kubernetes cluster

Synopsis

Install the CoreDNS addon components via the API server. Please note that although the DNS server is deployed, it will not be scheduled until CNI is installed.

```
kubeadm init phase addon coredns [flags]
```

Options

--config string
  Path to a kubeadm configuration file.
--dry-run
  Don't apply any changes; just output what would be done.
--feature-gates string
  A set of key=value pairs that describe feature gates for various features. Options are:
  ControlPlaneKubeletLocalMode=truelfalse (BETA - default=true)
  NodeLocalCRISocket=truelfalse (BETA - default=true)
  PublicKeysECDSA=truelfalse (DEPRECATED - default=false)
  RootlessControlPlane=truelfalse (ALPHA - default=false)
  WaitForAllControlPlaneComponents=truelfalse (default=true)
-h, --help
  help for coredns
--image-repository string Default: "registry.k8s.io"
  Choose a container registry to pull control plane images from
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
--print-manifest
  Print the addon manifests to STDOUT instead of installing them
--service-cidr string Default: "10.96.0.0/12"

Use alternative range of IP address for service VIPs.
--service-dns-domain string Default: "cluster.local"
Use alternative domain for services, e.g. "myorg.internal".

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Install the kube-proxy addon to a Kubernetes cluster

Synopsis

Install the kube-proxy addon components via the API server.

```
kubeadm init phase addon kube-proxy [flags]
```

Options

--apiserver-advertise-address string
The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
Port for the API Server to bind to.
--config string
Path to a kubeadm configuration file.
--control-plane-endpoint string
Specify a stable IP address or DNS name for the control plane.
--dry-run
Don't apply any changes; just output what would be done.
-h, --help
help for kube-proxy
--image-repository string Default: "registry.k8s.io"
Choose a container registry to pull control plane images from
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--kubernetes-version string Default: "stable-1"
Choose a specific Kubernetes version for the control plane.
--pod-network-cidr string
Specify range of IP addresses for the pod network. If set, the control plane will automatically allocate CIDRs for every node.
--print-manifest
Print the addon manifests to STDOUT instead of installing them

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

For more details on each field in the `v1beta4` configuration you can navigate to our API reference pages.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm alpha to try experimental functionality

---

kubeadm upgrade

`kubeadm upgrade` is a user-friendly command that wraps complex upgrading logic behind one command, with support for both planning an upgrade and actually performing it.

kubeadm upgrade guidance

The steps for performing an upgrade using kubeadm are outlined in this document. For older versions of kubeadm, please refer to older documentation sets of the Kubernetes website.

You can use `kubeadm upgrade diff` to see the changes that would be applied to static pod manifests.

In Kubernetes v1.15.0 and later, `kubeadm upgrade apply` and `kubeadm upgrade node` will also automatically renew the kubeadm managed certificates on this node, including those stored in kubeconfig files. To opt-out, it is possible to pass the flag `--certificate-renewal=false`. For more details about certificate renewal see the certificate management documentation.

Note:

The commands `kubeadm upgrade apply` and `kubeadm upgrade plan` have a legacy `--config` flag which makes it possible to reconfigure the cluster, while performing planning or upgrade of that particular control-plane node. Please be aware that the upgrade workflow was not designed for this scenario and there are reports of unexpected results.

kubeadm upgrade plan

Synopsis

Check which versions are available to upgrade to and validate whether your current cluster is upgradeable. This command can only run on the control plane nodes where the kubeconfig file "admin.conf" exists. To skip the internet check, pass in the optional [version] parameter.

```
kubeadm upgrade plan [version] [flags]
```

Options

--allow-experimental-upgrades
  Show unstable versions of Kubernetes as an upgrade alternative and allow upgrading to an alpha/beta/release candidate versions of Kubernetes.
--allow-missing-template-keys Default: true
  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
--allow-release-candidate-upgrades
  Show release candidate versions of Kubernetes as an upgrade alternative and allow upgrading to a release candidate versions of Kubernetes.
--config string
  Path to a kubeadm configuration file.
--etcd-upgrade Default: true
  Perform the upgrade of etcd.
-h, --help
  help for plan
--ignore-preflight-errors strings
  A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
-o, --output string Default: "text"
  Output format. One of: text|json|yaml|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file.
--print-config
  Specifies whether the configuration file that will be used in the upgrade should be printed or not.
--show-managed-fields
  If true, keep the managedFields when printing objects in JSON or YAML format.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm upgrade apply

Synopsis

Upgrade your Kubernetes cluster to the specified version

The "apply [version]" command executes the following phases:

```
preflight            Run preflight checks before upgrade
control-plane        Upgrade the control plane
upload-config        Upload the kubeadm and kubelet configurations to ConfigMaps
  /kubeadm             Upload the kubeadm ClusterConfiguration to a ConfigMap
  /kubelet             Upload the kubelet configuration to a ConfigMap
kubelet-config       Upgrade the kubelet configuration for this node
bootstrap-token      Configures bootstrap token and cluster-info RBAC rules
addon                Upgrade the default kubeadm addons
  /coredns             Upgrade the CoreDNS addon
  /kube-proxy          Upgrade the kube-proxy addon
post-upgrade         Run post upgrade tasks
```

```
kubeadm upgrade apply [version]
```

Options

--allow-experimental-upgrades
  Show unstable versions of Kubernetes as an upgrade alternative and allow upgrading to an alpha/beta/release candidate versions of Kubernetes.
--allow-release-candidate-upgrades
  Show release candidate versions of Kubernetes as an upgrade alternative and allow upgrading to a release candidate versions of Kubernetes.
--certificate-renewal Default: true
  Perform the renewal of certificates used by component changed during upgrades.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Do not change any state, just output what actions would be performed.
--etcd-upgrade Default: true
  Perform the upgrade of etcd.
-f, --force
  Force upgrading although some requirements might not be met. This also implies non-interactive mode.
-h, --help
  help for apply
--ignore-preflight-errors strings
  A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--patches string
  Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.
--print-config
  Specifies whether the configuration file that will be used in the upgrade should be printed or not.
--skip-phases strings
  List of phases to be skipped
-y, --yes
  Perform the upgrade and do not prompt for confirmation (non-interactive mode).

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm upgrade diff

Synopsis

Show what differences would be applied to existing static pod manifests. See also: kubeadm upgrade apply --dry-run

```
kubeadm upgrade diff [version] [flags]
```

Options

--config string
  Path to a kubeadm configuration file.
-c, --context-lines int Default: 3
  How many lines of context in the diff
-h, --help
  help for diff
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm upgrade node

Synopsis

Upgrade commands for a node in the cluster

The "node" command executes the following phases:

```
preflight       Run upgrade node pre-flight checks
control-plane   Upgrade the control plane instance deployed on this node, if any
kubelet-config  Upgrade the kubelet configuration for this node
addon           Upgrade the default kubeadm addons
  /coredns        Upgrade the CoreDNS addon
  /kube-proxy     Upgrade the kube-proxy addon
post-upgrade    Run post upgrade tasks
```

```
kubeadm upgrade node [flags]
```

Options

--certificate-renewal Default: true
Perform the renewal of certificates used by component changed during upgrades.

--config string
Path to a kubeadm configuration file.

--dry-run
Do not change any state, just output the actions that would be performed.

--etcd-upgrade Default: true
Perform the upgrade of etcd.

-h, --help
help for node

--ignore-preflight-errors strings
A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

--patches string
Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

--skip-phases strings
List of phases to be skipped

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

What's next

- kubeadm config if you initialized your cluster using kubeadm v1.7.x or lower, to configure your cluster for `kubeadm upgrade`

Scheduler Configuration

FEATURE STATE: `Kubernetes v1.25 [stable]`

You can customize the behavior of the `kube-scheduler` by writing a configuration file and passing its path as a command line argument.

A scheduling Profile allows you to configure the different stages of scheduling in the kube-scheduler. Each stage is exposed in an extension point. Plugins provide scheduling behaviors by implementing one or more of these extension points.

You can specify scheduling profiles by running `kube-scheduler --config <filename>`, using the KubeSchedulerConfiguration v1 struct.

A minimal configuration looks as follows:

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationclientConnection:  kubeconfig: /etc/srv/kubernetes/kube-scheduler/kubeconfig
```

Note:

KubeSchedulerConfiguration v1beta3 is deprecated in v1.26 and is removed in v1.29. Please migrate KubeSchedulerConfiguration to v1.

Profiles

A scheduling Profile allows you to configure the different stages of scheduling in the kube-scheduler. Each stage is exposed in an extension point. Plugins provide scheduling behaviors by implementing one or more of these extension points.

You can configure a single instance of `kube-scheduler` to run multiple profiles.

Extension points

Scheduling happens in a series of stages that are exposed through the following extension points:

1. `queueSort`: These plugins provide an ordering function that is used to sort pending Pods in the scheduling queue. Exactly one queue sort plugin may be enabled at a time.
2. `preFilter`: These plugins are used to pre-process or check information about a Pod or the cluster before filtering. They can mark a pod as unschedulable.
3. `filter`: These plugins are the equivalent of Predicates in a scheduling Policy and are used to filter out nodes that can not run the Pod. Filters are called in the configured order. A pod is marked as unschedulable if no nodes pass all the filters.
4. `postFilter`: These plugins are called in their configured order when no feasible nodes were found for the pod. If any `postFilter` plugin marks the Pod *schedulable*, the remaining plugins are not called.
5. `preScore`: This is an informational extension point that can be used for doing pre-scoring work.
6. `score`: These plugins provide a score to each node that has passed the filtering phase. The scheduler will then select the node with the highest weighted scores sum.
7. `reserve`: This is an informational extension point that notifies plugins when resources have been reserved for a given Pod. Plugins also implement an `Unreserve` call that gets called in the case of failure during or after `Reserve`.
8. `permit`: These plugins can prevent or delay the binding of a Pod.
9. `preBind`: These plugins perform any work required before a Pod is bound.
10. `bind`: The plugins bind a Pod to a Node. `bind` plugins are called in order and once one has done the binding, the remaining plugins are skipped. At least one bind plugin is required.
11. `postBind`: This is an informational extension point that is called after a Pod has been bound.
12. `multiPoint`: This is a config-only field that allows plugins to be enabled or disabled for all of their applicable extension points simultaneously.

For each extension point, you could disable specific default plugins or enable your own. For example:

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationprofiles:  - plugins:      score:        disabled:        - name: PodTopologySpread        enabled
```

You can use `*` as name in the disabled array to disable all default plugins for that extension point. This can also be used to rearrange plugins order, if desired.

Scheduling plugins

The following plugins, enabled by default, implement one or more of these extension points:

- `ImageLocality`: Favors nodes that already have the container images that the Pod runs. Extension points: `score`.
- `TaintToleration`: Implements taints and tolerations. Implements extension points: `filter`, `preScore`, `score`.
- `NodeName`: Checks if a Pod spec node name matches the current node. Extension points: `filter`.
- `NodePorts`: Checks if a node has free ports for the requested Pod ports. Extension points: `preFilter`, `filter`.
- `NodeAffinity`: Implements node selectors and node affinity. Extension points: `filter`, `score`.
- `PodTopologySpread`: Implements Pod topology spread. Extension points: `preFilter`, `filter`, `preScore`, `score`.
- `NodeUnschedulable`: Filters out nodes that have `.spec.unschedulable` set to true. Extension points: `filter`.
- `NodeResourcesFit`: Checks if the node has all the resources that the Pod is requesting. The score can use one of three strategies: `LeastAllocated` (default), `MostAllocated` and `RequestedToCapacityRatio`. Extension points: `preFilter`, `filter`, `score`.
- `NodeResourcesBalancedAllocation`: Favors nodes that would obtain a more balanced resource usage if the Pod is scheduled there. Extension points: `score`.
- `VolumeBinding`: Checks if the node has or if it can bind the requested volumes. Extension points: `preFilter`, `filter`, `reserve`, `preBind`, `score`.

  Note:

  `score` extension point is enabled when `StorageCapacityScoring` feature is enabled. It prioritizes the smallest PVs that can fit the requested volume size.
- `VolumeRestrictions`: Checks that volumes mounted in the node satisfy restrictions that are specific to the volume provider. Extension points: `filter`.
- `VolumeZone`: Checks that volumes requested satisfy any zone requirements they might have. Extension points: `filter`.
- `NodeVolumeLimits`: Checks that CSI volume limits can be satisfied for the node. Extension points: `filter`.
- `EBSLimits`: Checks that AWS EBS volume limits can be satisfied for the node. Extension points: `filter`.
- `GCEPDLimits`: Checks that GCP-PD volume limits can be satisfied for the node. Extension points: `filter`.
- `AzureDiskLimits`: Checks that Azure disk volume limits can be satisfied for the node. Extension points: `filter`.
- `InterPodAffinity`: Implements inter-Pod affinity and anti-affinity. Extension points: `preFilter`, `filter`, `preScore`, `score`.
- `PrioritySort`: Provides the default priority based sorting. Extension points: `queueSort`.
- `DefaultBinder`: Provides the default binding mechanism. Extension points: `bind`.
- `DefaultPreemption`: Provides the default preemption mechanism. Extension points: `postFilter`.

You can also enable the following plugins, through the component config APIs, that are not enabled by default:

- `CinderLimits`: Checks that OpenStack Cinder volume limits can be satisfied for the node. Extension points: `filter`.

Multiple profiles

You can configure `kube-scheduler` to run more than one profile. Each profile has an associated scheduler name and can have a different set of plugins configured in its extension points.

With the following sample configuration, the scheduler will run with two profiles: one with the default plugins and one with all scoring plugins disabled.

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationprofiles:  - schedulerName: default-scheduler  - schedulerName: no-scoring-scheduler    plugins:
```

Pods that want to be scheduled according to a specific profile can include the corresponding scheduler name in its `.spec.schedulerName`.

By default, one profile with the scheduler name `default-scheduler` is created. This profile includes the default plugins described above. When declaring more than one profile, a unique scheduler name for each of them is required.

If a Pod doesn't specify a scheduler name, kube-apiserver will set it to `default-scheduler`. Therefore, a profile with this scheduler name should exist to get those pods scheduled.

Note:

Pod's scheduling events have `.spec.schedulerName` as their `reportingController`. Events for leader election use the scheduler name of the first profile in the list.

For more information, please refer to the `reportingController` section under Event API Reference.

Note:

All profiles must use the same plugin in the `queueSort` extension point and have the same configuration parameters (if applicable). This is because the scheduler only has one pending pods queue.

Plugins that apply to multiple extension points

Starting from `kubescheduler.config.k8s.io/v1beta3`, there is an additional field in the profile config, `multiPoint`, which allows for easily enabling or disabling a plugin across several extension points. The intent of `multiPoint` config is to simplify the configuration needed for users and administrators when using custom profiles.

Consider a plugin, `MyPlugin`, which implements the `preScore`, `score`, `preFilter`, and `filter` extension points. To enable `MyPlugin` for all its available extension points, the profile config looks like:

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationprofiles:  - schedulerName: multipoint-scheduler    plugins:      multiPoint:        enabled:
```

This would equate to manually enabling `MyPlugin` for all of its extension points, like so:

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationprofiles:  - schedulerName: non-multipoint-scheduler    plugins:      preScore:        enabled:
```

One benefit of using `multiPoint` here is that if `MyPlugin` implements another extension point in the future, the `multiPoint` config will automatically enable it for the new extension.

Specific extension points can be excluded from `MultiPoint` expansion using the `disabled` field for that extension point. This works with disabling default plugins, non-default plugins, or with the wildcard (`'*'`) to disable all plugins. An example of this, disabling `Score` and `PreScore`, would be:

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationprofiles:  - schedulerName: non-multipoint-scheduler    plugins:      multiPoint:        enabled:
```

Starting from `kubescheduler.config.k8s.io/v1beta3`, all default plugins are enabled internally through `MultiPoint`. However, individual extension points are still available to allow flexible reconfiguration of the default values (such as ordering and Score weights). For example, consider two Score plugins `DefaultScore1` and `DefaultScore2`, each with a weight of `1`. They can be reordered with different weights like so:

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationprofiles:  - schedulerName: multipoint-scheduler    plugins:      score:        enabled:        - 
```

In this example, it's unnecessary to specify the plugins in `MultiPoint` explicitly because they are default plugins. And the only plugin specified in `Score` is `DefaultScore2`. This is because plugins set through specific extension points will always take precedence over `MultiPoint` plugins. So, this snippet essentially re-orders the two plugins without needing to specify both of them.

The general hierarchy for precedence when configuring `MultiPoint` plugins is as follows:

1. Specific extension points run first, and their settings override whatever is set elsewhere
2. Plugins manually configured through `MultiPoint` and their settings
3. Default plugins and their default settings

To demonstrate the above hierarchy, the following example is based on these plugins:

Plugin	Extension Points
`DefaultQueueSort`	`QueueSort`
`CustomQueueSort`	`QueueSort`
`DefaultPlugin1`	`Score`, `Filter`
`DefaultPlugin2`	`Score`
`CustomPlugin1`	`Score`, `Filter`
`CustomPlugin2`	`Score`, `Filter`

A valid sample configuration for these plugins would be:

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfigurationprofiles:  - schedulerName: multipoint-scheduler    plugins:      multiPoint:        enabled:
```

Note that there is no error for re-declaring a `MultiPoint` plugin in a specific extension point. The re-declaration is ignored (and logged), as specific extension points take precedence.

Besides keeping most of the config in one spot, this sample does a few things:

- Enables the custom `queueSort` plugin and disables the default one
- Enables `CustomPlugin1` and `CustomPlugin2`, which will run first for all of their extension points

- Disables `DefaultPlugin1`, but only for `filter`
- Reorders `DefaultPlugin2` to run first in `score` (even before the custom plugins)

In versions of the config before `v1beta3`, without `multiPoint`, the above snippet would equate to this:

```
apiVersion: kubescheduler.config.k8s.io/v1beta2
kind: KubeSchedulerConfigurationprofiles:  - schedulerName: multipoint-scheduler    plugins:      # Disable the default QueueSort
```

While this is a complicated example, it demonstrates the flexibility of `MultiPoint` config as well as its seamless integration with the existing methods for configuring extension points.

Scheduler configuration migrations

- v1beta1 → v1beta2
- v1beta2 → v1beta3
- v1beta3 → v1

- With the v1beta2 configuration version, you can use a new score extension for the `NodeResourcesFit` plugin. The new extension combines the functionalities of the `NodeResourcesLeastAllocated`, `NodeResourcesMostAllocated` and `RequestedToCapacityRatio` plugins. For example, if you previously used the `NodeResourcesMostAllocated` plugin, you would instead use `NodeResourcesFit` (enabled by default) and add a `pluginConfig` with a `scoreStrategy` that is similar to:

  ```
  apiVersion: kubescheduler.config.k8s.io/v1beta2
  kind: KubeSchedulerConfigurationprofiles:- pluginConfig:  - args:      scoringStrategy:        resources:        - name: cpu
  ```

- The scheduler plugin `NodeLabel` is deprecated; instead, use the `NodeAffinity` plugin (enabled by default) to achieve similar behavior.
- The scheduler plugin `ServiceAffinity` is deprecated; instead, use the `InterPodAffinity` plugin (enabled by default) to achieve similar behavior.
- The scheduler plugin `NodePreferAvoidPods` is deprecated; instead, use node taints to achieve similar behavior.
- A plugin enabled in a v1beta2 configuration file takes precedence over the default configuration for that plugin.
- Invalid `host` or `port` configured for scheduler healthz and metrics bind address will cause validation failure.
- Three plugins' weight are increased by default:
  - `InterPodAffinity` from 1 to 2
  - `NodeAffinity` from 1 to 2
  - `TaintToleration` from 1 to 3
- The scheduler plugin `SelectorSpread` is removed, instead, use the `PodTopologySpread` plugin (enabled by default) to achieve similar behavior.

What's next

- Read the kube-scheduler reference
- Learn about scheduling
- Read the kube-scheduler configuration (v1) reference

Implementation details

FEATURE STATE: `Kubernetes v1.10 [stable]`

`kubeadm init` and `kubeadm join` together provide a nice user experience for creating a bare Kubernetes cluster from scratch, that aligns with the best-practices. However, it might not be obvious *how* kubeadm does that.

This document provides additional details on what happens under the hood, with the aim of sharing knowledge on the best practices for a Kubernetes cluster.

Core design principles

The cluster that `kubeadm init` and `kubeadm join` set up should be:

- **Secure**: It should adopt latest best-practices like:
  - enforcing RBAC
  - using the Node Authorizer
  - using secure communication between the control plane components
  - using secure communication between the API server and the kubelets
  - lock-down the kubelet API
  - locking down access to the API for system components like the kube-proxy and CoreDNS
  - locking down what a Bootstrap Token can access
- **User-friendly**: The user should not have to run anything more than a couple of commands:
  - `kubeadm init`
  - `export KUBECONFIG=/etc/kubernetes/admin.conf`
  - `kubectl apply -f <network-plugin-of-choice.yaml>`
  - `kubeadm join --token <token> <endpoint>:<port>`
- **Extendable**:
  - It should *not* favor any particular network provider. Configuring the cluster network is out-of-scope
  - It should provide the possibility to use a config file for customizing various parameters

Constants and well-known values and paths

In order to reduce complexity and to simplify development of higher level tools that build on top of kubeadm, it uses a limited set of constant values for well-known paths and file names.

The Kubernetes directory `/etc/kubernetes` is a constant in the application, since it is clearly the given path in a majority of cases, and the most intuitive location; other constant paths and file names are:

- `/etc/kubernetes/manifests` as the path where the kubelet should look for static Pod manifests. Names of static Pod manifests are:
  - `etcd.yaml`
  - `kube-apiserver.yaml`
  - `kube-controller-manager.yaml`
  - `kube-scheduler.yaml`
- `/etc/kubernetes/` as the path where kubeconfig files with identities for control plane components are stored. Names of kubeconfig files are:
  - `kubelet.conf` (`bootstrap-kubelet.conf` during TLS bootstrap)
  - `controller-manager.conf`
  - `scheduler.conf`
  - `admin.conf` for the cluster admin and kubeadm itself
  - `super-admin.conf` for the cluster super-admin that can bypass RBAC
- Names of certificates and key files:
  - `ca.crt`, `ca.key` for the Kubernetes certificate authority
  - `apiserver.crt`, `apiserver.key` for the API server certificate
  - `apiserver-kubelet-client.crt`, `apiserver-kubelet-client.key` for the client certificate used by the API server to connect to the kubelets securely
  - `sa.pub`, `sa.key` for the key used by the controller manager when signing ServiceAccount
  - `front-proxy-ca.crt`, `front-proxy-ca.key` for the front proxy certificate authority
  - `front-proxy-client.crt`, `front-proxy-client.key` for the front proxy client

The kubeadm configuration file format

Most kubeadm commands support a `--config` flag which allows passing a configuration file from disk. The configuration file format follows the common Kubernetes API `apiVersion` / `kind` scheme, but is considered a component configuration format. Several Kubernetes components, such as the kubelet, also support file-based configuration.

Different kubeadm subcommands require a different `kind` of configuration file. For example, `InitConfiguration` for `kubeadm init`, `JoinConfiguration` for `kubeadm join`, `UpgradeConfiguration` for `kubeadm upgrade` and `ResetConfiguration` for `kubeadm reset`.

The command `kubeadm config migrate` can be used to migrate an older format configuration file to a newer (current) configuration format. The kubeadm tool only supports migrating from deprecated configuration formats to the current format.

See the kubeadm configuration reference page for more details.

kubeadm init workflow internal design

The `kubeadm init` consists of a sequence of atomic work tasks to perform, as described in the `kubeadm init` internal workflow.

The `kubeadm init phase` command allows users to invoke each task individually, and ultimately offers a reusable and composable API/toolbox that can be used by other Kubernetes bootstrap tools, by any IT automation tool or by an advanced user for creating custom clusters.

Preflight checks

Kubeadm executes a set of preflight checks before starting the init, with the aim to verify preconditions and avoid common cluster startup problems. The user can skip specific preflight checks or all of them with the `--ignore-preflight-errors` option.

- [Warning] if the Kubernetes version to use (specified with the `--kubernetes-version` flag) is at least one minor version higher than the kubeadm CLI version.
- Kubernetes system requirements:
  - if running on linux:
    - [Error] if Kernel is older than the minimum required version
    - [Error] if required cgroups subsystem aren't set up
- [Error] if the CRI endpoint does not answer
- [Error] if user is not root
- [Error] if the machine hostname is not a valid DNS subdomain
- [Warning] if the host name cannot be reached via network lookup
- [Error] if kubelet version is lower that the minimum kubelet version supported by kubeadm (current minor -1)
- [Error] if kubelet version is at least one minor higher than the required controlplane version (unsupported version skew)
- [Warning] if kubelet service does not exist or if it is disabled
- [Warning] if firewalld is active
- [Error] if API server bindPort or ports 10250/10251/10252 are used
- [Error] if `/etc/kubernetes/manifest` folder already exists and it is not empty
- [Error] if swap is on
- [Error] if `ip`, `iptables`, `mount`, `nsenter` commands are not present in the command path
- [Warning] if `ethtool`, `tc`, `touch` commands are not present in the command path
- [Warning] if extra arg flags for API server, controller manager, scheduler contains some invalid options
- [Warning] if connection to https://API.AdvertiseAddress:API.BindPort goes through proxy

- [Warning] if connection to services subnet goes through proxy (only first address checked)
- [Warning] if connection to Pods subnet goes through proxy (only first address checked)
- If external etcd is provided:
  - [Error] if etcd version is older than the minimum required version
  - [Error] if etcd certificates or keys are specified, but not provided
- If external etcd is NOT provided (and thus local etcd will be installed):
  - [Error] if ports 2379 is used
  - [Error] if Etcd.DataDir folder already exists and it is not empty
- If authorization mode is ABAC:
  - [Error] if abac_policy.json does not exist
- If authorization mode is WebHook
  - [Error] if webhook_authz.conf does not exist

Note:

Preflight checks can be invoked individually with the `kubeadm init phase preflight` command.

Generate the necessary certificates

Kubeadm generates certificate and private key pairs for different purposes:

- A self signed certificate authority for the Kubernetes cluster saved into `ca.crt` file and `ca.key` private key file

- A serving certificate for the API server, generated using `ca.crt` as the CA, and saved into `apiserver.crt` file with its private key `apiserver.key`. This certificate should contain the following alternative names:

  - The Kubernetes service's internal clusterIP (the first address in the services CIDR, e.g. `10.96.0.1` if service subnet is `10.96.0.0/12`)
  - Kubernetes DNS names, e.g. `kubernetes.default.svc.cluster.local` if `--service-dns-domain` flag value is `cluster.local`, plus default DNS names `kubernetes.default.svc`, `kubernetes.default`, `kubernetes`
  - The node-name
  - The `--apiserver-advertise-address`
  - Additional alternative names specified by the user

- A client certificate for the API server to connect to the kubelets securely, generated using `ca.crt` as the CA and saved into `apiserver-kubelet-client.crt` file with its private key `apiserver-kubelet-client.key`. This certificate should be in the `system:masters` organization

- A private key for signing ServiceAccount Tokens saved into `sa.key` file along with its public key `sa.pub`

- A certificate authority for the front proxy saved into `front-proxy-ca.crt` file with its key `front-proxy-ca.key`

- A client certificate for the front proxy client, generated using `front-proxy-ca.crt` as the CA and saved into `front-proxy-client.crt` file with its private key`front-proxy-client.key`

Certificates are stored by default in `/etc/kubernetes/pki`, but this directory is configurable using the `--cert-dir` flag.

Please note that:

1. If a given certificate and private key pair both exist, and their content is evaluated to be compliant with the above specs, the existing files will be used and the generation phase for the given certificate will be skipped. This means the user can, for example, copy an existing CA to `/etc/kubernetes/pki/ca.{crt,key}`, and then kubeadm will use those files for signing the rest of the certs. See also using custom certificates
2. For the CA, it is possible to provide the `ca.crt` file but not the `ca.key` file. If all other certificates and kubeconfig files are already in place, kubeadm recognizes this condition and activates the ExternalCA, which also implies the `csrsigner` controller in controller-manager won't be started
3. If kubeadm is running in external CA mode; all the certificates must be provided by the user, because kubeadm cannot generate them by itself
4. In case kubeadm is executed in the `--dry-run` mode, certificate files are written in a temporary folder
5. Certificate generation can be invoked individually with the `kubeadm init phase certs all` command

Generate kubeconfig files for control plane components

Kubeadm generates kubeconfig files with identities for control plane components:

- A kubeconfig file for the kubelet to use during TLS bootstrap - `/etc/kubernetes/bootstrap-kubelet.conf`. Inside this file, there is a bootstrap-token or embedded client certificates for authenticating this node with the cluster.

  This client certificate should:

  - Be in the `system:nodes` organization, as required by the Node Authorization module
  - Have the Common Name (CN) `system:node:<hostname-lowercased>`

- A kubeconfig file for controller-manager, `/etc/kubernetes/controller-manager.conf`; inside this file is embedded a client certificate with controller-manager identity. This client certificate should have the CN `system:kube-controller-manager`, as defined by default RBAC core components roles

- A kubeconfig file for scheduler, `/etc/kubernetes/scheduler.conf`; inside this file is embedded a client certificate with scheduler identity. This client certificate should have the CN `system:kube-scheduler`, as defined by default RBAC core components roles

Additionally, a kubeconfig file for kubeadm as an administrative entity is generated and stored in `/etc/kubernetes/admin.conf`. This file includes a certificate with `Subject: O = kubeadm:cluster-admins, CN = kubernetes-admin`. `kubeadm:cluster-admins` is a group managed by kubeadm. It is bound to the `cluster-admin` ClusterRole during `kubeadm init`, by using the `super-admin.conf` file, which does not require RBAC. This `admin.conf` file must remain on control plane nodes and should not be shared with additional users.

During `kubeadm init` another kubeconfig file is generated and stored in `/etc/kubernetes/super-admin.conf`. This file includes a certificate with `Subject: O = system:masters, CN = kubernetes-super-admin`. `system:masters` is a superuser group that bypasses RBAC and makes `super-`

`admin.conf` useful in case of an emergency where a cluster is locked due to RBAC misconfiguration. The `super-admin.conf` file must be stored in a safe location and should not be shared with additional users.

See RBAC user facing role bindings for additional information on RBAC and built-in ClusterRoles and groups.

You can run `kubeadm kubeconfig user` to generate kubeconfig files for additional users.

Caution:

The generated configuration files include an embedded authentication key, and you should treat them as confidential.

Also note that:

1. `ca.crt` certificate is embedded in all the kubeconfig files.
2. If a given kubeconfig file exists, and its content is evaluated as compliant with the above specs, the existing file will be used and the generation phase for the given kubeconfig will be skipped
3. If kubeadm is running in ExternalCA mode, all the required kubeconfig must be provided by the user as well, because kubeadm cannot generate any of them by itself
4. In case kubeadm is executed in the `--dry-run` mode, kubeconfig files are written in a temporary folder
5. Generation of kubeconfig files can be invoked individually with the `kubeadm init phase kubeconfig all` command

Generate static Pod manifests for control plane components

Kubeadm writes static Pod manifest files for control plane components to `/etc/kubernetes/manifests`. The kubelet watches this directory for Pods to be created on startup.

Static Pod manifests share a set of common properties:

- All static Pods are deployed on `kube-system` namespace
- All static Pods get `tier:control-plane` and `component:{component-name}` labels
- All static Pods use the `system-node-critical` priority class
- `hostNetwork: true` is set on all static Pods to allow control plane startup before a network is configured; as a consequence:
  - The `address` that the controller-manager and the scheduler use to refer to the API server is `127.0.0.1`
  - If the etcd server is set up locally, the `etcd-server` address will be set to `127.0.0.1:2379`
- Leader election is enabled for both the controller-manager and the scheduler
- Controller-manager and the scheduler will reference kubeconfig files with their respective, unique identities
- All static Pods get any extra flags or patches that you specify, as described in passing custom arguments to control plane components
- All static Pods get any extra Volumes specified by the user (Host path)

Please note that:

1. All images will be pulled from registry.k8s.io by default. See using custom images for customizing the image repository
2. In case kubeadm is executed in the `--dry-run` mode, static Pod files are written in a temporary folder
3. Static Pod manifest generation for control plane components can be invoked individually with the `kubeadm init phase control-plane all` command

API server

The static Pod manifest for the API server is affected by the following parameters provided by the users:

- The `apiserver-advertise-address` and `apiserver-bind-port` to bind to; if not provided, those values default to the IP address of the default network interface on the machine and port 6443
- The `service-cluster-ip-range` to use for services
- If an external etcd server is specified, the `etcd-servers` address and related TLS settings (`etcd-cafile`, `etcd-certfile`, `etcd-keyfile`); if an external etcd server is not provided, a local etcd will be used (via host network)
- If a cloud provider is specified, the corresponding `--cloud-provider` parameter is configured together with the `--cloud-config` path if such file exists (this is experimental, alpha and will be removed in a future version)

Other API server flags that are set unconditionally are:

- `--insecure-port=0` to avoid insecure connections to the api server
- `--enable-bootstrap-token-auth=true` to enable the `BootstrapTokenAuthenticator` authentication module. See TLS Bootstrapping for more details
- `--allow-privileged` to `true` (required e.g. by kube proxy)
- `--requestheader-client-ca-file` to `front-proxy-ca.crt`
- `--enable-admission-plugins` to:
  - `NamespaceLifecycle` e.g. to avoid deletion of system reserved namespaces
  - `LimitRanger` and `ResourceQuota` to enforce limits on namespaces
  - `ServiceAccount` to enforce service account automation

    - `PersistentVolumeLabel` attaches region or zone labels to PersistentVolumes as defined by the cloud provider (This admission controller is deprecated and will be removed in a future version. It is not deployed by kubeadm by default with v1.9 onwards when not explicitly opting into using `gce` or `aws` as cloud providers)
    - `DefaultStorageClass` to enforce default storage class on `PersistentVolumeClaim` objects
    - `DefaultTolerationSeconds`
    - `NodeRestriction` to limit what a kubelet can modify (e.g. only pods on this node)

- `--kubelet-preferred-address-types` to `InternalIP,ExternalIP,Hostname;` this makes `kubectl logs` and other API server-kubelet communication work in environments where the hostnames of the nodes aren't resolvable

- Flags for using certificates generated in previous steps:

    - `--client-ca-file` to `ca.crt`
    - `--tls-cert-file` to `apiserver.crt`
    - `--tls-private-key-file` to `apiserver.key`
    - `--kubelet-client-certificate` to `apiserver-kubelet-client.crt`
    - `--kubelet-client-key` to `apiserver-kubelet-client.key`
    - `--service-account-key-file` to `sa.pub`
    - `--requestheader-client-ca-file` to `front-proxy-ca.crt`
    - `--proxy-client-cert-file` to `front-proxy-client.crt`
    - `--proxy-client-key-file` to `front-proxy-client.key`

- Other flags for securing the front proxy (API Aggregation) communications:

    - `--requestheader-username-headers=X-Remote-User`
    - `--requestheader-group-headers=X-Remote-Group`
    - `--requestheader-extra-headers-prefix=X-Remote-Extra-`
    - `--requestheader-allowed-names=front-proxy-client`

Controller manager

The static Pod manifest for the controller manager is affected by following parameters provided by the users:

- If kubeadm is invoked specifying a `--pod-network-cidr`, the subnet manager feature required for some CNI network plugins is enabled by setting:

    - `--allocate-node-cidrs=true`
    - `--cluster-cidr` and `--node-cidr-mask-size` flags according to the given CIDR

Other flags that are set unconditionally are:

- `--controllers` enabling all the default controllers plus `BootstrapSigner` and `TokenCleaner` controllers for TLS bootstrap. See TLS Bootstrapping for more details.

- `--use-service-account-credentials` to `true`

- Flags for using certificates generated in previous steps:

    - `--root-ca-file` to `ca.crt`
    - `--cluster-signing-cert-file` to `ca.crt`, if External CA mode is disabled, otherwise to `""`
    - `--cluster-signing-key-file` to `ca.key`, if External CA mode is disabled, otherwise to `""`
    - `--service-account-private-key-file` to `sa.key`

Scheduler

The static Pod manifest for the scheduler is not affected by parameters provided by the user.

Generate static Pod manifest for local etcd

If you specified an external etcd, this step will be skipped, otherwise kubeadm generates a static Pod manifest file for creating a local etcd instance running in a Pod with following attributes:

- listen on `localhost:2379` and use `HostNetwork=true`
- make a `hostPath` mount out from the `dataDir` to the host's filesystem
- Any extra flags specified by the user

Please note that:

1. The etcd container image will be pulled from `registry.gcr.io` by default. See using custom images for customizing the image repository.
2. If you run kubeadm in `--dry-run` mode, the etcd static Pod manifest is written into a temporary folder.
3. You can directly invoke static Pod manifest generation for local etcd, using the `kubeadm init phase etcd local` command.

Wait for the control plane to come up

On control plane nodes, kubeadm waits up to 4 minutes for the control plane components and the kubelet to be available. It does that by performing a health check on the respective component `/healthz` or `/livez` endpoints.

After the control plane is up, kubeadm completes the tasks described in following paragraphs.

Save the kubeadm ClusterConfiguration in a ConfigMap for later reference

kubeadm saves the configuration passed to `kubeadm init` in a ConfigMap named `kubeadm-config` under `kube-system` namespace.

This will ensure that kubeadm actions executed in future (e.g `kubeadm upgrade`) will be able to determine the actual/current cluster state and make new decisions based on that data.

Please note that:

1. Before saving the ClusterConfiguration, sensitive information like the token is stripped from the configuration
2. Upload of control plane node configuration can be invoked individually with the command `kubeadm init phase upload-config`.

Mark the node as control-plane

As soon as the control plane is available, kubeadm executes the following actions:

- Labels the node as control-plane with `node-role.kubernetes.io/control-plane=""`
- Taints the node with `node-role.kubernetes.io/control-plane:NoSchedule`

Please note that the phase to mark the control-plane phase can be invoked individually with the `kubeadm init phase mark-control-plane` command.

Configure TLS-Bootstrapping for node joining

Kubeadm uses Authenticating with Bootstrap Tokens for joining new nodes to an existing cluster; for more details see also design proposal.

`kubeadm init` ensures that everything is properly configured for this process, and this includes following steps as well as setting API server and controller flags as already described in previous paragraphs.

Note:

TLS bootstrapping for nodes can be configured with the command `kubeadm init phase bootstrap-token`, executing all the configuration steps described in following paragraphs; alternatively, each step can be invoked individually.

Create a bootstrap token

`kubeadm init` creates a first bootstrap token, either generated automatically or provided by the user with the `--token` flag; as documented in bootstrap token specification, token should be saved as a secret with name `bootstrap-token-<token-id>` under `kube-system` namespace.

Please note that:

1. The default token created by `kubeadm init` will be used to validate temporary user during TLS bootstrap process; those users will be member of `system:bootstrappers:kubeadm:default-node-token` group
2. The token has a limited validity, default 24 hours (the interval may be changed with the `–token-ttl` flag)
3. Additional tokens can be created with the `kubeadm token` command, that provide other useful functions for token management as well.

Allow joining nodes to call CSR API

Kubeadm ensures that users in `system:bootstrappers:kubeadm:default-node-token` group are able to access the certificate signing API.

This is implemented by creating a ClusterRoleBinding named `kubeadm:kubelet-bootstrap` between the group above and the default RBAC role `system:node-bootstrapper`.

Set up auto approval for new bootstrap tokens

Kubeadm ensures that the Bootstrap Token will get its CSR request automatically approved by the csrapprover controller.

This is implemented by creating ClusterRoleBinding named `kubeadm:node-autoapprove-bootstrap` between the `system:bootstrappers:kubeadm:default-node-token` group and the default role `system:certificates.k8s.io:certificatesigningrequests:nodeclient`.

The role `system:certificates.k8s.io:certificatesigningrequests:nodeclient` should be created as well, granting POST permission to `/apis/certificates.k8s.io/certificatesigningrequests/nodeclient`.

Set up nodes certificate rotation with auto approval

Kubeadm ensures that certificate rotation is enabled for nodes, and that a new certificate request for nodes will get its CSR request automatically approved by the csrapprover controller.

This is implemented by creating ClusterRoleBinding named `kubeadm:node-autoapprove-certificate-rotation` between the `system:nodes` group and the default role `system:certificates.k8s.io:certificatesigningrequests:selfnodeclient`.

Create the public cluster-info ConfigMap

This phase creates the `cluster-info` ConfigMap in the `kube-public` namespace.

Additionally, it creates a Role and a RoleBinding granting access to the ConfigMap for unauthenticated users (i.e. users in RBAC group `system:unauthenticated`).

Note:

The access to the `cluster-info` ConfigMap *is not* rate-limited. This may or may not be a problem if you expose your cluster's API server to the internet; worst-case scenario here is a DoS attack where an attacker uses all the in-flight requests the kube-apiserver can handle to serve the `cluster-info` ConfigMap.

Install addons

Kubeadm installs the internal DNS server and the kube-proxy addon components via the API server.

Note:

This phase can be invoked individually with the command `kubeadm init phase addon all`.

proxy

A ServiceAccount for `kube-proxy` is created in the `kube-system` namespace; then kube-proxy is deployed as a DaemonSet:

- The credentials (`ca.crt` and `token`) to the control plane come from the ServiceAccount
- The location (URL) of the API server comes from a ConfigMap
- The `kube-proxy` ServiceAccount is bound to the privileges in the `system:node-proxier` ClusterRole

DNS

- The CoreDNS service is named `kube-dns` for compatibility reasons with the legacy `kube-dns` addon.
- A ServiceAccount for CoreDNS is created in the `kube-system` namespace.
- The `coredns` ServiceAccount is bound to the privileges in the `system:coredns` ClusterRole

In Kubernetes version 1.21, support for using `kube-dns` with kubeadm was removed. You can use CoreDNS with kubeadm even when the related Service is named `kube-dns`.

kubeadm join phases internal design

Similarly to `kubeadm init`, also `kubeadm join` internal workflow consists of a sequence of atomic work tasks to perform.

This is split into discovery (having the Node trust the Kubernetes API Server) and TLS bootstrap (having the Kubernetes API Server trust the Node).

see Authenticating with Bootstrap Tokens or the corresponding design proposal.

Preflight checks

`kubeadm` executes a set of preflight checks before starting the join, with the aim to verify preconditions and avoid common cluster startup problems.

Also note that:

1. `kubeadm join` preflight checks are basically a subset of `kubeadm init` preflight checks
2. If you are joining a Windows node, Linux specific controls are skipped.
3. In any case the user can skip specific preflight checks (or eventually all preflight checks) with the `--ignore-preflight-errors` option.

Discovery cluster-info

There are 2 main schemes for discovery. The first is to use a shared token along with the IP address of the API server. The second is to provide a file (that is a subset of the standard kubeconfig file).

Shared token discovery

If `kubeadm join` is invoked with `--discovery-token`, token discovery is used; in this case the node basically retrieves the cluster CA certificates from the `cluster-info` ConfigMap in the `kube-public` namespace.

In order to prevent "man in the middle" attacks, several steps are taken:

- First, the CA certificate is retrieved via insecure connection (this is possible because `kubeadm init` is granted access to `cluster-info` users for `system:unauthenticated`)
- Then the CA certificate goes through following validation steps:
  - Basic validation: using the token ID against a JWT signature
  - Pub key validation: using provided `--discovery-token-ca-cert-hash`. This value is available in the output of `kubeadm init` or can be calculated using standard tools (the hash is calculated over the bytes of the Subject Public Key Info (SPKI) object as in RFC7469). The `--discovery-token-ca-cert-hash flag` may be repeated multiple times to allow more than one public key.
  - As an additional validation, the CA certificate is retrieved via secure connection and then compared with the CA retrieved initially

Note:

You can skip CA validation by passing the `--discovery-token-unsafe-skip-ca-verification` flag on the command line. This weakens the kubeadm security model since others can potentially impersonate the Kubernetes API server.

File/https discovery

If `kubeadm join` is invoked with `--discovery-file`, file discovery is used; this file can be a local file or downloaded via an HTTPS URL; in case of HTTPS, the host installed CA bundle is used to verify the connection.

With file discovery, the cluster CA certificate is provided into the file itself; in fact, the discovery file is a kubeconfig file with only `server` and `certificate-authority-data` attributes set, as described in the `kubeadm join` reference doc; when the connection with the cluster is established, kubeadm tries to access the `cluster-info` ConfigMap, and if available, uses it.

TLS Bootstrap

Once the cluster info is known, the file `bootstrap-kubelet.conf` is written, thus allowing kubelet to do TLS Bootstrapping.

The TLS bootstrap mechanism uses the shared token to temporarily authenticate with the Kubernetes API server to submit a certificate signing request (CSR) for a locally created key pair.

The request is then automatically approved and the operation completes saving `ca.crt` file and `kubelet.conf` file to be used by the kubelet for joining the cluster, while `bootstrap-kubelet.conf` is deleted.

Note:

- The temporary authentication is validated against the token saved during the `kubeadm init` process (or with additional tokens created with `kubeadm token` command)
- The temporary authentication resolves to a user member of `system:bootstrappers:kubeadm:default-node-token` group which was granted access to the CSR api during the `kubeadm init` process
- The automatic CSR approval is managed by the csrapprover controller, according to the configuration present in the `kubeadm init` process

kubeadm upgrade workflow internal design

`kubeadm upgrade` has sub-commands for handling the upgrade of the Kubernetes cluster created by kubeadm. You must run `kubeadm upgrade apply` on a control plane node (you can choose which one); this starts the upgrade process. You then run `kubeadm upgrade node` on all remaining nodes (both worker nodes and control plane nodes).

Both `kubeadm upgrade apply` and `kubeadm upgrade node` have a `phase` subcommand which provides access to the internal phases of the upgrade process. See `kubeadm upgrade phase` for more details.

Additional utility upgrade commands are `kubeadm upgrade plan` and `kubeadm upgrade diff`.

All upgrade sub-commands support passing a configuration file.

kubeadm upgrade plan

You can optionally run `kubeadm upgrade plan` before you run `kubeadm upgrade apply`. The `plan` subcommand checks which versions are available to upgrade to and validates whether your current cluster is upgradeable.

kubeadm upgrade diff

This shows what differences would be applied to existing static pod manifests for control plane nodes. A more verbose way to do the same thing is running `kubeadm upgrade apply --dry-run` or `kubeadm upgrade node --dry-run`.

kubeadm upgrade apply

`kubeadm upgrade apply` prepares the cluster for the upgrade of all nodes, and also upgrades the control plane node where it's run. The steps it performs are:

- Runs preflight checks similarly to `kubeadm init` and `kubeadm join`, ensuring container images are downloaded and the cluster is in a good state to be upgraded.
- Upgrades the control plane manifest files on disk in `/etc/kubernetes/manifests` and waits for the kubelet to restart the components if the files have changed.
- Uploads the updated kubeadm and kubelet configurations to the cluster in the `kubeadm-config` and the `kubelet-config` ConfigMaps (both in the `kube-system` namespace).
- Writes updated kubelet configuration for this node in `/var/lib/kubelet/config.yaml`, and read the node's `/var/lib/kubelet/instance-config.yaml` file and patch fields like `containerRuntimeEndpoint` from this instance configuration into `/var/lib/kubelet/config.yaml`.
- Configures bootstrap token and the `cluster-info` ConfigMap for RBAC rules. This is the same as in the `kubeadm init` stage and ensures that the cluster continues to support nodes joining with bootstrap tokens.
- Upgrades the kube-proxy and CoreDNS addons conditionally if all existing kube-apiservers in the cluster have already been upgraded to the target version.
- Performs any post-upgrade tasks, such as, cleaning up deprecated features which are release specific.

kubeadm upgrade node

`kubeadm upgrade node` upgrades a single control plane or worker node after the cluster upgrade has started (by running `kubeadm upgrade apply`). The command detects if the node is a control plane node by checking if the file `/etc/kubernetes/manifests/kube-apiserver.yaml` exists. On finding that file, the kubeadm tool infers that there is a running kube-apiserver Pod on this node.

- Runs preflight checks similarly to `kubeadm upgrade apply`.
- For control plane nodes, upgrades the control plane manifest files on disk in `/etc/kubernetes/manifests` and waits for the kubelet to restart the components if the files have changed.
- Writes updated kubelet configuration for this node in `/var/lib/kubelet/config.yaml`, and read the node's `/var/lib/kubelet/instance-config.yaml` file and patch fields like `containerRuntimeEndpoint` from this instance configuration into `/var/lib/kubelet/config.yaml`.
- (For control plane nodes) upgrades the kube-proxy and CoreDNS addons conditionally, provided that all existing API servers in the cluster have already been upgraded to the target version.
- Performs any post-upgrade tasks, such as cleaning up deprecated features which are release specific.

kubeadm reset workflow internal design

You can use the `kubeadm reset` subcommand on a node where kubeadm commands previously executed. This subcommand performs a **best-effort** cleanup of the node. If certain actions fail you must intervene and perform manual cleanup.

The command supports phases. See `kubeadm reset phase` for more details.

The command supports a configuration file.

Additionally:

- IPVS, iptables and nftables rules are **not** cleaned up.
- CNI (network plugin) configuration is **not** cleaned up.
- `.kube/` in the user's home directory is **not** cleaned up.

The command has the following stages:

- Runs preflight checks on the node to determine if its healthy.
- For control plane nodes, removes any local etcd member data.
- Stops the kubelet.
- Stops running containers.
- Unmounts any mounted directories in `/var/lib/kubelet`.
- Deletes any files and directories managed by kubeadm in `/var/lib/kubelet` and `/etc/kubernetes`.

Scheduling

Scheduler Configuration

Scheduling Policies

---

kubeadm version

This command prints the version of kubeadm.

Synopsis

Print the version of kubeadm

```
kubeadm version [flags]
```

Options

-h, --help
  help for version

-o, --output string
  Output format; available options are 'yaml', 'json' and 'short'

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

---

kubeadm config

During `kubeadm init`, kubeadm uploads the `ClusterConfiguration` object to your cluster in a ConfigMap called `kubeadm-config` in the `kube-system` namespace. This configuration is then read during `kubeadm join`, `kubeadm reset` and `kubeadm upgrade`.

You can use `kubeadm config print` to print the default static configuration that kubeadm uses for `kubeadm init` and `kubeadm join`.

Note:

The output of the command is meant to serve as an example. You must manually edit the output of this command to adapt to your setup. Remove the fields that you are not certain about and kubeadm will try to default them on runtime by examining the host.

For more information on `init` and `join` navigate to Using kubeadm init with a configuration file or Using kubeadm join with a configuration file.

For more information on using the kubeadm configuration API navigate to Customizing components with the kubeadm API.

You can use `kubeadm config migrate` to convert your old configuration files that contain a deprecated API version to a newer, supported API version.

`kubeadm config validate` can be used for validating a configuration file.

`kubeadm config images list` and `kubeadm config images pull` can be used to list and pull the images that kubeadm requires.

kubeadm config print

Print configuration

Synopsis

This command prints configurations for subcommands provided. For details, see:
https://pkg.go.dev/k8s.io/kubernetes/cmd/kubeadm/app/apis/kubeadm#section-directories

```
kubeadm config print [flags]
```

Options

-h, --help
  help for print

Options inherited from parent commands

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm config print init-defaults

Print default init configuration, that can be used for 'kubeadm init'

Synopsis

This command prints objects such as the default init configuration that is used for 'kubeadm init'.

Note that sensitive values like the Bootstrap Token fields are replaced with placeholder values like "abcdef.0123456789abcdef" in order to pass validation but not perform the real computation for creating a token.

```
kubeadm config print init-defaults [flags]
```

Options

--component-configs strings
  A comma-separated list for component config API objects to print the default values for. Available values: [KubeProxyConfiguration KubeletConfiguration]. If this flag is not set, no component configs will be printed.
-h, --help
  help for init-defaults

Options inherited from parent commands

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm config print join-defaults

Print default join configuration, that can be used for 'kubeadm join'

Synopsis

This command prints objects such as the default join configuration that is used for 'kubeadm join'.

Note that sensitive values like the Bootstrap Token fields are replaced with placeholder values like "abcdef.0123456789abcdef" in order to pass validation but not perform the real computation for creating a token.

```
kubeadm config print join-defaults [flags]
```

Options

-h, --help
  help for join-defaults

Options inherited from parent commands

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm config migrate

Read an older version of the kubeadm configuration API types from a file, and output the similar config object for the newer version

Synopsis

This command lets you convert configuration objects of older versions to the latest supported version, locally in the CLI tool without ever touching anything in the cluster. In this version of kubeadm, the following API versions are supported:

- kubeadm.k8s.io/v1beta4

Further, kubeadm can only write out config of version "kubeadm.k8s.io/v1beta4", but read both types. So regardless of what version you pass to the --old-config parameter here, the API object will be read, deserialized, defaulted, converted, validated, and re-serialized when written to stdout or --new-config if specified.

In other words, the output of this command is what kubeadm actually would read internally if you submitted this file to "kubeadm init"

```
kubeadm config migrate [flags]
```

Options

--allow-experimental-api
Allow migration to experimental, unreleased APIs.
-h, --help
help for migrate
--new-config string
Path to the resulting equivalent kubeadm config file using the new API version. Optional, if not specified output will be sent to STDOUT.
--old-config string
Path to the kubeadm config file that is using an old API version and should be converted. This flag is mandatory.

Options inherited from parent commands

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm config validate

Read a file containing the kubeadm configuration API and report any validation problems

Synopsis

This command lets you validate a kubeadm configuration API file and report any warnings and errors. If there are no errors the exit status will be zero, otherwise it will be non-zero. Any unmarshaling problems such as unknown API fields will trigger errors. Unknown API versions and fields with invalid values will also trigger errors. Any other errors or warnings may be reported depending on contents of the input file.

In this version of kubeadm, the following API versions are supported:

- kubeadm.k8s.io/v1beta4

```
kubeadm config validate [flags]
```

Options

--allow-experimental-api
Allow validation of experimental, unreleased APIs.
--config string
Path to a kubeadm configuration file.
-h, --help
help for validate

Options inherited from parent commands

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm config images list

Synopsis

Print a list of images kubeadm will use. The configuration file is used in case any images or image repositories are customized

```
kubeadm config images list [flags]
```

Options

--allow-missing-template-keys Default: true
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
--config string
Path to a kubeadm configuration file.
--feature-gates string
A set of key=value pairs that describe feature gates for various features. Options are:
ControlPlaneKubeletLocalMode=true|false (BETA - default=true)
NodeLocalCRISocket=true|false (BETA - default=true)
PublicKeysECDSA=true|false (DEPRECATED - default=false)
RootlessControlPlane=true|false (ALPHA - default=false)
WaitForAllControlPlaneComponents=true|false (default=true)
-h, --help
help for list
--image-repository string Default: "registry.k8s.io"
Choose a container registry to pull control plane images from
--kubernetes-version string Default: "stable-1"
Choose a specific Kubernetes version for the control plane.
-o, --output string Default: "text"
Output format. One of: text|json|yaml|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file.
--show-managed-fields
If true, keep the managedFields when printing objects in JSON or YAML format.

Options inherited from parent commands

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm config images pull

Synopsis

Pull images used by kubeadm

```
kubeadm config images pull [flags]
```

Options

--config string
Path to a kubeadm configuration file.
--cri-socket string
Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--feature-gates string
A set of key=value pairs that describe feature gates for various features. Options are:
ControlPlaneKubeletLocalMode=true|false (BETA - default=true)
NodeLocalCRISocket=true|false (BETA - default=true)
PublicKeysECDSA=true|false (DEPRECATED - default=false)
RootlessControlPlane=true|false (ALPHA - default=false)
WaitForAllControlPlaneComponents=true|false (default=true)
-h, --help

help for pull

--image-repository string Default: "registry.k8s.io"

Choose a container registry to pull control plane images from

--kubernetes-version string Default: "stable-1"

Choose a specific Kubernetes version for the control plane.

Options inherited from parent commands

--kubeconfig string Default: "/etc/kubernetes/admin.conf"

The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

What's next

- kubeadm upgrade to upgrade a Kubernetes cluster to a newer version

---

kubeadm reset phase

`kubeadm reset phase` enables you to invoke atomic steps of the node reset process. Hence, you can let kubeadm do some of the work and you can fill in the gaps if you wish to apply customization.

`kubeadm reset phase` is consistent with the kubeadm reset workflow, and behind the scene both use the same code.

kubeadm reset phase

- phase

Synopsis

Use this command to invoke single phase of the "reset" workflow

```
kubeadm reset phase [flags]
```

Options

-h, --help

help for phase

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm reset phase preflight

Using this phase you can execute preflight checks on a node that is being reset.

- preflight

Run reset pre-flight checks

Synopsis

Run pre-flight checks for kubeadm reset.

```
kubeadm reset phase preflight [flags]
```

Options

--dry-run

Don't apply any changes; just output what would be done.

-f, --force

Reset the node without prompting for confirmation.

-h, --help

help for preflight

--ignore-preflight-errors strings

A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm reset phase remove-etcd-member

Using this phase you can remove this control-plane node's etcd member from the etcd cluster.

- remove-etcd-member

Synopsis

Remove a local etcd member for a control plane node.

```
kubeadm reset phase remove-etcd-member [flags]
```

Options

--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for remove-etcd-member
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm reset phase cleanup-node

Using this phase you can perform cleanup on this node.

- cleanup-node

Synopsis

Run cleanup node.

```
kubeadm reset phase cleanup-node [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"
  The path to the directory where the certificates are stored. If specified, clean this directory.
--cleanup-tmp-dir
  Cleanup the "/etc/kubernetes/tmp" directory
--cri-socket string
  Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for cleanup-node

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm alpha to try experimental functionality

Scheduling Policies

In Kubernetes versions before v1.23, a scheduling policy can be used to specify the *predicates* and *priorities* process. For example, you can set a scheduling policy by running `kube-scheduler --policy-config-file <filename>` or `kube-scheduler --policy-configmap <ConfigMap>`.

This scheduling policy is not supported since Kubernetes v1.23. Associated flags `policy-config-file`, `policy-configmap`, `policy-configmap-namespace` and `use-legacy-policy-config` are also not supported. Instead, use the Scheduler Configuration to achieve similar behavior.

What's next

- Learn about scheduling
- Learn about kube-scheduler Configuration
- Read the kube-scheduler configuration reference (v1)

---

kubeadm upgrade phases

kubeadm upgrade apply phase

Using the phases of `kubeadm upgrade apply`, you can choose to execute the separate steps of the initial upgrade of a control plane node.

- phase
- preflight
- control-plane
- upload-config
- kubelet-config
- bootstrap-token
- addon
- post-upgrade

Synopsis

Use this command to invoke single phase of the "apply" workflow

```
kubeadm upgrade apply phase [flags]
```

Options

-h, --help
  help for phase

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Run preflight checks before upgrade

```
kubeadm upgrade apply phase preflight [flags]
```

Options

--allow-experimental-upgrades
  Show unstable versions of Kubernetes as an upgrade alternative and allow upgrading to an alpha/beta/release candidate versions of Kubernetes.
--allow-release-candidate-upgrades
  Show release candidate versions of Kubernetes as an upgrade alternative and allow upgrading to a release candidate versions of Kubernetes.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Do not change any state, just output what actions would be performed.
-f, --force
  Force upgrading although some requirements might not be met. This also implies non-interactive mode.
-h, --help
  help for preflight
--ignore-preflight-errors strings
  A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

-y, --yes
  Perform the upgrade and do not prompt for confirmation (non-interactive mode).

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upgrade the control plane

```
kubeadm upgrade apply phase control-plane [flags]
```

Options

--certificate-renewal Default: true
  Perform the renewal of certificates used by component changed during upgrades.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Do not change any state, just output what actions would be performed.
--etcd-upgrade Default: true
  Perform the upgrade of etcd.
-h, --help
  help for control-plane
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--patches string
  Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upload the kubeadm and kubelet configurations to ConfigMaps

```
kubeadm upgrade apply phase upload-config [flags]
```

Options

-h, --help
  help for upload-config

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upgrade the kubelet configuration for this node by downloading it from the kubelet-config ConfigMap stored in the cluster

```
kubeadm upgrade apply phase kubelet-config [flags]
```

Options

--config string
  Path to a kubeadm configuration file.
--dry-run
  Do not change any state, just output what actions would be performed.
-h, --help
  help for kubelet-config

--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.
--patches string
  Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Configures bootstrap token and cluster-info RBAC rules

```
kubeadm upgrade apply phase bootstrap-token [flags]
```

Options

--config string
  Path to a kubeadm configuration file.
--dry-run
  Do not change any state, just output what actions would be performed.
-h, --help
  help for bootstrap-token
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upgrade the default kubeadm addons

```
kubeadm upgrade apply phase addon [flags]
```

Options

-h, --help
  help for addon

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Run post upgrade tasks

```
kubeadm upgrade apply phase post-upgrade [flags]
```

Options

--config string
  Path to a kubeadm configuration file.
--dry-run
  Do not change any state, just output what actions would be performed.
-h, --help
  help for post-upgrade
--kubeconfig string Default: "/etc/kubernetes/admin.conf"
  The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm upgrade node phase

Using the phases of `kubeadm upgrade node` you can choose to execute the separate steps of the upgrade of secondary control-plane or worker nodes.

- phase
- preflight
- control-plane
- kubelet-config
- addon
- post-upgrade

Synopsis

Use this command to invoke single phase of the "node" workflow

```
kubeadm upgrade node phase [flags]
```

Options

-h, --help
  help for phase

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Run upgrade node pre-flight checks

Synopsis

Run pre-flight checks for kubeadm upgrade node.

```
kubeadm upgrade node phase preflight [flags]
```

Options

--config string
  Path to a kubeadm configuration file.
-h, --help
  help for preflight
--ignore-preflight-errors strings
  A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upgrade the control plane instance deployed on this node, if any

```
kubeadm upgrade node phase control-plane [flags]
```

Options

--certificate-renewal Default: true
  Perform the renewal of certificates used by component changed during upgrades.
--config string
  Path to a kubeadm configuration file.
--dry-run
  Do not change any state, just output the actions that would be performed.
--etcd-upgrade Default: true
  Perform the upgrade of etcd.
-h, --help
  help for control-plane
--kubeconfig string Default: "/etc/kubernetes/admin.conf"

The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

--patches string

Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upgrade the kubelet configuration for this node by downloading it from the kubelet-config ConfigMap stored in the cluster

```
kubeadm upgrade node phase kubelet-config [flags]
```

Options

--config string

Path to a kubeadm configuration file.

--dry-run

Do not change any state, just output the actions that would be performed.

-h, --help

help for kubelet-config

--kubeconfig string Default: "/etc/kubernetes/admin.conf"

The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

--patches string

Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Upgrade the default kubeadm addons

```
kubeadm upgrade node phase addon [flags]
```

Options

-h, --help

help for addon

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Run post upgrade tasks

```
kubeadm upgrade node phase post-upgrade [flags]
```

Options

--config string

Path to a kubeadm configuration file.

--dry-run

Do not change any state, just output the actions that would be performed.

-h, --help

help for post-upgrade

--kubeconfig string Default: "/etc/kubernetes/admin.conf"

The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm upgrade to upgrade a kubeadm node
- kubeadm alpha to try experimental functionality

Kubeadm

Kubeadm is a tool built to provide `kubeadm init` and `kubeadm join` as best-practice "fast paths" for creating Kubernetes clusters.

kubeadm performs the actions necessary to get a minimum viable cluster up and running. By design, it cares only about bootstrapping, not about provisioning machines. Likewise, installing various nice-to-have addons, like the Kubernetes Dashboard, monitoring solutions, and cloud-specific addons, is not in scope.

Instead, we expect higher-level and more tailored tooling to be built on top of kubeadm, and ideally, using kubeadm as the basis of all deployments will make it easier to create conformant clusters.

How to install

To install kubeadm, see the installation guide.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster
- kubeadm upgrade to upgrade a Kubernetes cluster to a newer version
- kubeadm config if you initialized your cluster using kubeadm v1.7.x or lower, to configure your cluster for `kubeadm upgrade`
- kubeadm token to manage tokens for `kubeadm join`
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm certs to manage Kubernetes certificates
- kubeadm kubeconfig to manage kubeconfig files
- kubeadm version to print the kubeadm version
- kubeadm alpha to preview a set of features made available for gathering feedback from the community

---

kubeadm alpha

Caution:

`kubeadm alpha` provides a preview of a set of features made available for gathering feedback from the community. Please try it out and give us feedback!

Currently there are no experimental commands under `kubeadm alpha`.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`

---

kubeadm join

This command initializes a new Kubernetes node and joins it to the existing cluster.

Run this on any machine you wish to join an existing cluster

Synopsis

When joining a kubeadm initialized cluster, we need to establish bidirectional trust. This is split into discovery (having the Node trust the Kubernetes Control Plane) and TLS bootstrap (having the Kubernetes Control Plane trust the Node).

There are 2 main schemes for discovery. The first is to use a shared token along with the IP address of the API server. The second is to provide a file - a subset of the standard kubeconfig file. The discovery/kubeconfig file supports token, client-go authentication plugins ("exec"), "tokenFile", and "authProvider". This file can be a local file or downloaded via an HTTPS URL. The forms are kubeadm join --discovery-token abcdef.1234567890abcdef 1.2.3.4:6443, kubeadm join --discovery-file path/to/file.conf, or kubeadm join --discovery-file https://url/file.conf. Only one form can be used. If the discovery information is loaded from a URL, HTTPS must be used. Also, in that case the host installed CA bundle is used to verify the connection.

If you use a shared token for discovery, you should also pass the --discovery-token-ca-cert-hash flag to validate the public key of the root certificate authority (CA) presented by the Kubernetes Control Plane. The value of this flag is specified as "<hash-type>:<hex-encoded-value>", where the supported hash type is "sha256". The hash is calculated over the bytes of the Subject Public Key Info (SPKI) object (as in RFC7469). This value is available in the output of "kubeadm init" or can be calculated using standard tools. The --discovery-token-ca-cert-hash flag may be repeated multiple times to allow more than one public key.

If you cannot know the CA public key hash ahead of time, you can pass the --discovery-token-unsafe-skip-ca-verification flag to disable this verification. This weakens the kubeadm security model since other nodes can potentially impersonate the Kubernetes Control Plane.

The TLS bootstrap mechanism is also driven via a shared token. This is used to temporarily authenticate with the Kubernetes Control Plane to submit a certificate signing request (CSR) for a locally created key pair. By default, kubeadm will set up the Kubernetes Control Plane to automatically approve these signing requests. This token is passed in with the --tls-bootstrap-token abcdef.1234567890abcdef flag.

Often times the same token is used for both parts. In this case, the --token flag can be used instead of specifying each token individually.

The "join [api-server-endpoint]" command executes the following phases:

```
preflight              Run join pre-flight checks
control-plane-prepare  Prepare the machine for serving a control plane
  /download-certs        Download certificates shared among control-plane nodes from the kubeadm-certs Secret
  /certs                 Generate the certificates for the new control plane components
  /kubeconfig            Generate the kubeconfig for the new control plane components
  /control-plane         Generate the manifests for the new control plane components
kubelet-start          Write kubelet settings, certificates and (re)start the kubelet
control-plane-join     Join a machine as a control plane instance
  /etcd                  Add a new local etcd member
  /mark-control-plane    Mark a node as a control-plane
wait-control-plane     Wait for the control plane to start
```

```
kubeadm join [api-server-endpoint] [flags]
```

Options

--apiserver-advertise-address string

If the node should host a new control plane instance, the IP address the API Server will advertise it's listening on. If not set the default network interface will be used.

--apiserver-bind-port int32 Default: 6443

If the node should host a new control plane instance, the port for the API Server to bind to.

--certificate-key string

Use this key to decrypt the certificate secrets uploaded by init. The certificate key is a hex encoded string that is an AES key of size 32 bytes.

--config string

Path to a kubeadm configuration file.

--control-plane

Create a new control plane instance on this node

--cri-socket string

Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.

--discovery-file string

For file-based discovery, a file or URL from which to load cluster information.

--discovery-token string

For token-based discovery, the token used to validate cluster information fetched from the API server.

--discovery-token-ca-cert-hash strings

For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").

--discovery-token-unsafe-skip-ca-verification

For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.

--dry-run

Don't apply any changes; just output what would be done.

-h, --help

help for join

--ignore-preflight-errors strings

A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.

--node-name string

Specify the node name.

--patches string

Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

--skip-phases strings
  List of phases to be skipped
--tls-bootstrap-token string
  Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.
--token string
  Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

The join workflow

`kubeadm join` bootstraps a Kubernetes worker node or a control-plane node and adds it to the cluster. This action consists of the following steps for worker nodes:

1. kubeadm downloads necessary cluster information from the API server. By default, it uses the bootstrap token and the CA key hash to verify the authenticity of that data. The root CA can also be discovered directly via a file or URL.
2. Once the cluster information is known, kubelet can start the TLS bootstrapping process.

   The TLS bootstrap uses the shared token to temporarily authenticate with the Kubernetes API server to submit a certificate signing request (CSR); by default the control plane signs this CSR request automatically.
3. Finally, kubeadm configures the local kubelet to connect to the API server with the definitive identity assigned to the node.

For control-plane nodes additional steps are performed:

1. Downloading certificates shared among control-plane nodes from the cluster (if explicitly requested by the user).
2. Generating control-plane component manifests, certificates and kubeconfig.
3. Adding new local etcd member.

Using join phases with kubeadm

Kubeadm allows you join a node to the cluster in phases using `kubeadm join phase`.

To view the ordered list of phases and sub-phases you can call `kubeadm join --help`. The list will be located at the top of the help screen and each phase will have a description next to it. Note that by calling `kubeadm join` all of the phases and sub-phases will be executed in this exact order.

Some phases have unique flags, so if you want to have a look at the list of available options add `--help`, for example:

```
kubeadm join phase kubelet-start --help
```

Similar to the kubeadm init phase command, `kubeadm join phase` allows you to skip a list of phases using the `--skip-phases` flag.

For example:

```
sudo kubeadm join --skip-phases=preflight --config=config.yaml
```

FEATURE STATE: `Kubernetes v1.22 [beta]`

Alternatively, you can use the `skipPhases` field in `JoinConfiguration`.

Discovering what cluster CA to trust

The kubeadm discovery has several options, each with security tradeoffs. The right method for your environment depends on how you provision nodes and the security expectations you have about your network and node lifecycles.

Token-based discovery with CA pinning

This is the default mode in kubeadm. In this mode, kubeadm downloads the cluster configuration (including root CA) and validates it using the token as well as validating that the root CA public key matches the provided hash and that the API server certificate is valid under the root CA.

The CA key hash has the format `sha256:<hex_encoded_hash>`. By default, the hash value is printed at the end of the `kubeadm init` command or in the output from the `kubeadm token create --print-join-command` command. It is in a standard format (see RFC7469) and can also be calculated by 3rd party tools or provisioning systems. For example, using the OpenSSL CLI:

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | se
```

Example `kubeadm join` commands:

For worker nodes:

```
kubeadm join --discovery-token abcdef.1234567890abcdef --discovery-token-ca-cert-hash sha256:1234..cdef 1.2.3.4:6443
```

For control-plane nodes:

```
kubeadm join --discovery-token abcdef.1234567890abcdef --discovery-token-ca-cert-hash sha256:1234..cdef --control-plane 1.2.3.4:64
```

You can also call `join` for a control-plane node with `--certificate-key` to copy certificates to this node, if the `kubeadm init` command was called with `--upload-certs`.

Advantages:

- Allows bootstrapping nodes to securely discover a root of trust for the control-plane node even if other worker nodes or the network are compromised.
- Convenient to execute manually since all of the information required fits into a single `kubeadm join` command.

Disadvantages:

- The CA hash is not normally known until the control-plane node has been provisioned, which can make it more difficult to build automated provisioning tools that use kubeadm. By generating your CA in beforehand, you may workaround this limitation.

Token-based discovery without CA pinning

This mode relies only on the symmetric token to sign (HMAC-SHA256) the discovery information that establishes the root of trust for the control-plane. To use the mode the joining nodes must skip the hash validation of the CA public key, using `--discovery-token-unsafe-skip-ca-verification`. You should consider using one of the other modes if possible.

Example `kubeadm join` command:

```
kubeadm join --token abcdef.1234567890abcdef --discovery-token-unsafe-skip-ca-verification 1.2.3.4:6443
```

Advantages:

- Still protects against many network-level attacks.
- The token can be generated ahead of time and shared with the control-plane node and worker nodes, which can then bootstrap in parallel without coordination. This allows it to be used in many provisioning scenarios.

Disadvantages:

- If an attacker is able to steal a bootstrap token via some vulnerability, they can use that token (along with network-level access) to impersonate the control-plane node to other bootstrapping nodes. This may or may not be an appropriate tradeoff in your environment.

File or HTTPS-based discovery

This provides an out-of-band way to establish a root of trust between the control-plane node and bootstrapping nodes. Consider using this mode if you are building automated provisioning using kubeadm. The format of the discovery file is a regular Kubernetes kubeconfig file.

In case the discovery file does not contain credentials, the TLS discovery token will be used.

Example `kubeadm join` commands:

- `kubeadm join --discovery-file path/to/file.conf` (local file)
- `kubeadm join --discovery-file https://url/file.conf` (remote HTTPS URL)

Advantages:

- Allows bootstrapping nodes to securely discover a root of trust for the control-plane node even if the network or other worker nodes are compromised.

Disadvantages:

- Requires that you have some way to carry the discovery information from the control-plane node to the bootstrapping nodes. If the discovery file contains credentials you must keep it secret and transfer it over a secure channel. This might be possible with your cloud provider or provisioning tool.

Use of custom kubelet credentials with `kubeadm join`

To allow `kubeadm join` to use predefined kubelet credentials and skip client TLS bootstrap and CSR approval for a new node:

1. From a working control plane node in the cluster that has `/etc/kubernetes/pki/ca.key` execute `kubeadm kubeconfig user --org system:nodes --client-name system:node:$NODE > kubelet.conf`. `$NODE` must be set to the name of the new node.
2. Modify the resulted `kubelet.conf` manually to adjust the cluster name and the server endpoint, or run `kubeadm kubeconfig user --config` (it accepts `InitConfiguration`).

If your cluster does not have the `ca.key` file, you must sign the embedded certificates in the `kubelet.conf` externally. For additional information, see PKI certificates and requirements and Certificate Management with kubeadm.

1. Copy the resulting `kubelet.conf` to `/etc/kubernetes/kubelet.conf` on the new node.
2. Execute `kubeadm join` with the flag `--ignore-preflight-errors=FileAvailable--etc-kubernetes-kubelet.conf` on the new node.

Securing your installation even more

The defaults for kubeadm may not work for everyone. This section documents how to tighten up a kubeadm installation at the cost of some usability.

Turning off auto-approval of node client certificates

By default, there is a CSR auto-approver enabled that basically approves any client certificate request for a kubelet when a Bootstrap Token was used when authenticating. If you don't want the cluster to automatically approve kubelet client certs, you can turn it off by executing this command:

```
kubectl delete clusterrolebinding kubeadm:node-autoapprove-bootstrap
```

After that, `kubeadm join` will block until the admin has manually approved the CSR in flight:

1. Using `kubectl get csr`, you can see that the original CSR is in the Pending state.

   ```
   kubectl get csr
   ```

   The output is similar to this:

   ```
   NAME                                                   AGE       REQUESTOR                 CONDITION
   node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ   18s       system:bootstrap:878f07   Pending
   ```

2. `kubectl certificate approve` allows the admin to approve CSR.This action tells a certificate signing controller to issue a certificate to the requestor with the attributes requested in the CSR.

   ```
   kubectl certificate approve node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ
   ```

   The output is similar to this:

   ```
   certificatesigningrequest "node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ" approved
   ```

3. This would change the CSR resource to Active state.

   ```
   kubectl get csr
   ```

   The output is similar to this:

   ```
   NAME                                                   AGE       REQUESTOR                 CONDITION
   node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ   1m        system:bootstrap:878f07   Approved,Issued
   ```

This forces the workflow that `kubeadm join` will only succeed if `kubectl certificate approve` has been run.

Turning off public access to the `cluster-info` ConfigMap

In order to achieve the joining flow using the token as the only piece of validation information, a ConfigMap with some data needed for validation of the control-plane node's identity is exposed publicly by default. While there is no private data in this ConfigMap, some users might wish to turn it off regardless. Doing so will disable the ability to use the `--discovery-token` flag of the `kubeadm join` flow. Here are the steps to do so:

- Fetch the `cluster-info` file from the API Server:

```
kubectl -n kube-public get cm cluster-info -o jsonpath='{.data.kubeconfig}' | tee cluster-info.yaml
```

The output is similar to this:

```
apiVersion: v1
kind: Configclusters:- cluster:    certificate-authority-data: <ca-cert>    server: https://<ip>:<port>  name: ""contexts: []curre
```

- Use the `cluster-info.yaml` file as an argument to `kubeadm join --discovery-file`.
- Turn off public access to the `cluster-info` ConfigMap:

```
kubectl -n kube-public delete rolebinding kubeadm:bootstrap-signer-clusterinfo
```

These commands should be run after `kubeadm init` but before `kubeadm join`.

Using kubeadm join with a configuration file

Caution:

The config file is still considered beta and may change in future versions.

It's possible to configure `kubeadm join` with a configuration file instead of command line flags, and some more advanced features may only be available as configuration file options. This file is passed using the `--config` flag and it must contain a `JoinConfiguration` structure. Mixing `--config` with others flags may not be allowed in some cases.

The default configuration can be printed out using the kubeadm config print command.

If your configuration is not using the latest version it is **recommended** that you migrate using the kubeadm config migrate command.

For more information on the fields and usage of the configuration you can navigate to our API reference.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node.
- kubeadm token to manage tokens for `kubeadm join`.
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`.

---

kubeadm join phase

`kubeadm join phase` enables you to invoke atomic steps of the join process. Hence, you can let kubeadm do some of the work and you can fill in the gaps if you wish to apply customization.

`kubeadm join phase` is consistent with the kubeadm join workflow, and behind the scene both use the same code.

kubeadm join phase

- phase

Synopsis

Use this command to invoke single phase of the "join" workflow

```
kubeadm join phase [flags]
```

Options

-h, --help
  help for phase

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm join phase preflight

Using this phase you can execute preflight checks on a joining node.

- preflight

Run join pre-flight checks

Synopsis

Run pre-flight checks for kubeadm join.

```
kubeadm join phase preflight [api-server-endpoint] [flags]
```

Examples

```
  # Run join pre-flight checks using a config file.
  kubeadm join phase preflight --config kubeadm-config.yaml
```

Options

--apiserver-advertise-address string
  If the node should host a new control plane instance, the IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
  If the node should host a new control plane instance, the port for the API Server to bind to.
--certificate-key string
  Use this key to decrypt the certificate secrets uploaded by init. The certificate key is a hex encoded string that is an AES key of size 32 bytes.
--config string
  Path to a kubeadm configuration file.
--control-plane
  Create a new control plane instance on this node
--cri-socket string
  Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--discovery-file string
  For file-based discovery, a file or URL from which to load cluster information.
--discovery-token string
  For token-based discovery, the token used to validate cluster information fetched from the API server.
--discovery-token-ca-cert-hash strings
  For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").
--discovery-token-unsafe-skip-ca-verification
  For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.
--dry-run
  Don't apply any changes; just output what would be done.

-h, --help
  help for preflight
--ignore-preflight-errors strings
  A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.
--node-name string
  Specify the node name.
--tls-bootstrap-token string
  Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.
--token string
  Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm join phase control-plane-prepare

Using this phase you can prepare a node for serving a control-plane.

- control-plane-prepare
- all
- download-certs
- certs
- kubeconfig
- control-plane

Synopsis

Prepare the machine for serving a control plane

```
kubeadm join phase control-plane-prepare [flags]
```

Examples

```
  # Prepares the machine for serving a control plane
  kubeadm join phase control-plane-prepare all
```

Options

-h, --help
  help for control-plane-prepare

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Prepare the machine for serving a control plane

```
kubeadm join phase control-plane-prepare all [api-server-endpoint] [flags]
```

Options

--apiserver-advertise-address string
  If the node should host a new control plane instance, the IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
  If the node should host a new control plane instance, the port for the API Server to bind to.
--certificate-key string
  Use this key to decrypt the certificate secrets uploaded by init. The certificate key is a hex encoded string that is an AES key of size 32 bytes.
--config string
  Path to a kubeadm configuration file.
--control-plane
  Create a new control plane instance on this node
--discovery-file string
  For file-based discovery, a file or URL from which to load cluster information.
--discovery-token string

For token-based discovery, the token used to validate cluster information fetched from the API server.

--discovery-token-ca-cert-hash strings
For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").

--discovery-token-unsafe-skip-ca-verification
For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.

--dry-run
Don't apply any changes; just output what would be done.

-h, --help
help for all

--node-name string
Specify the node name.

--patches string
Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

--tls-bootstrap-token string
Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.

--token string
Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Download certificates shared among control-plane nodes from the kubeadm-certs Secret

```
kubeadm join phase control-plane-prepare download-certs [api-server-endpoint] [flags]
```

Options

--certificate-key string
Use this key to decrypt the certificate secrets uploaded by init. The certificate key is a hex encoded string that is an AES key of size 32 bytes.

--config string
Path to a kubeadm configuration file.

--control-plane
Create a new control plane instance on this node

--discovery-file string
For file-based discovery, a file or URL from which to load cluster information.

--discovery-token string
For token-based discovery, the token used to validate cluster information fetched from the API server.

--discovery-token-ca-cert-hash strings
For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").

--discovery-token-unsafe-skip-ca-verification
For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.

--dry-run
Don't apply any changes; just output what would be done.

-h, --help
help for download-certs

--tls-bootstrap-token string
Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.

--token string
Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string
The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the certificates for the new control plane components

```
kubeadm join phase control-plane-prepare certs [api-server-endpoint] [flags]
```

Options

--apiserver-advertise-address string
  If the node should host a new control plane instance, the IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--config string
  Path to a kubeadm configuration file.
--control-plane
  Create a new control plane instance on this node
--discovery-file string
  For file-based discovery, a file or URL from which to load cluster information.
--discovery-token string
  For token-based discovery, the token used to validate cluster information fetched from the API server.
--discovery-token-ca-cert-hash strings
  For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").
--discovery-token-unsafe-skip-ca-verification
  For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for certs
--node-name string
  Specify the node name.
--tls-bootstrap-token string
  Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.
--token string
  Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the kubeconfig for the new control plane components

```
kubeadm join phase control-plane-prepare kubeconfig [api-server-endpoint] [flags]
```

Options

--certificate-key string
  Use this key to decrypt the certificate secrets uploaded by init. The certificate key is a hex encoded string that is an AES key of size 32 bytes.
--config string
  Path to a kubeadm configuration file.
--control-plane
  Create a new control plane instance on this node
--discovery-file string
  For file-based discovery, a file or URL from which to load cluster information.
--discovery-token string
  For token-based discovery, the token used to validate cluster information fetched from the API server.
--discovery-token-ca-cert-hash strings
  For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").
--discovery-token-unsafe-skip-ca-verification
  For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for kubeconfig
--tls-bootstrap-token string
  Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.
--token string
  Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Generate the manifests for the new control plane components

```
kubeadm join phase control-plane-prepare control-plane [flags]
```

Options

--apiserver-advertise-address string
  If the node should host a new control plane instance, the IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--apiserver-bind-port int32 Default: 6443
  If the node should host a new control plane instance, the port for the API Server to bind to.
--config string
  Path to a kubeadm configuration file.
--control-plane
  Create a new control plane instance on this node
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for control-plane
--patches string
  Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm join phase kubelet-start

Using this phase you can write the kubelet settings, certificates and (re)start the kubelet.

- kubelet-start

Write kubelet settings, certificates and (re)start the kubelet

Synopsis

Write a file with KubeletConfiguration and an environment file with node specific kubelet settings, and then (re)start kubelet.

```
kubeadm join phase kubelet-start [api-server-endpoint] [flags]
```

Options

--config string
  Path to a kubeadm configuration file.
--cri-socket string
  Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--discovery-file string
  For file-based discovery, a file or URL from which to load cluster information.
--discovery-token string
  For token-based discovery, the token used to validate cluster information fetched from the API server.
--discovery-token-ca-cert-hash strings
  For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").
--discovery-token-unsafe-skip-ca-verification
  For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.
--dry-run
  Don't apply any changes; just output what would be done.

-h, --help
  help for kubelet-start
--node-name string
  Specify the node name.
--patches string
  Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.
--tls-bootstrap-token string
  Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.
--token string
  Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

kubeadm join phase control-plane-join

Using this phase you can join a node as a control-plane instance.

- control-plane-join
- all
- etcd
- mark-control-plane

Synopsis

Join a machine as a control plane instance

```
kubeadm join phase control-plane-join [flags]
```

Examples

```
  # Joins a machine as a control plane instance
  kubeadm join phase control-plane-join all
```

Options

-h, --help
  help for control-plane-join

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Join a machine as a control plane instance

```
kubeadm join phase control-plane-join all [flags]
```

Options

--apiserver-advertise-address string
  If the node should host a new control plane instance, the IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
--config string
  Path to a kubeadm configuration file.
--control-plane
  Create a new control plane instance on this node
--dry-run
  Don't apply any changes; just output what would be done.
-h, --help
  help for all
--node-name string

Specify the node name.

--patches string

Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Add a new local etcd member

```
kubeadm join phase control-plane-join etcd [flags]
```

Options

--apiserver-advertise-address string

If the node should host a new control plane instance, the IP address the API Server will advertise it's listening on. If not set the default network interface will be used.

--config string

Path to a kubeadm configuration file.

--control-plane

Create a new control plane instance on this node

--dry-run

Don't apply any changes; just output what would be done.

-h, --help

help for etcd

--node-name string

Specify the node name.

--patches string

Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Synopsis

Mark a node as a control-plane

```
kubeadm join phase control-plane-join mark-control-plane [flags]
```

Options

--config string

Path to a kubeadm configuration file.

--control-plane

Create a new control plane instance on this node

--dry-run

Don't apply any changes; just output what would be done.

-h, --help

help for mark-control-plane

--node-name string

Specify the node name.

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm alpha to try experimental functionality

---

kubeadm init

This command initializes a Kubernetes control plane node.

Synopsis

Run this command in order to set up the Kubernetes control plane

The "init" command executes the following phases:

```
preflight                    Run pre-flight checks
certs                        Certificate generation
  /ca                          Generate the self-signed Kubernetes CA to provision identities for other Kubernetes components
  /apiserver                   Generate the certificate for serving the Kubernetes API
  /apiserver-kubelet-client    Generate the certificate for the API server to connect to kubelet
  /front-proxy-ca              Generate the self-signed CA to provision identities for front proxy
  /front-proxy-client          Generate the certificate for the front proxy client
  /etcd-ca                     Generate the self-signed CA to provision identities for etcd
  /etcd-server                 Generate the certificate for serving etcd
  /etcd-peer                   Generate the certificate for etcd nodes to communicate with each other
  /etcd-healthcheck-client     Generate the certificate for liveness probes to healthcheck etcd
  /apiserver-etcd-client       Generate the certificate the apiserver uses to access etcd
  /sa                          Generate a private key for signing service account tokens along with its public key
kubeconfig                   Generate all kubeconfig files necessary to establish the control plane and the admin kubeconfig file
  /admin                       Generate a kubeconfig file for the admin to use and for kubeadm itself
  /super-admin                 Generate a kubeconfig file for the super-admin
  /kubelet                     Generate a kubeconfig file for the kubelet to use *only* for cluster bootstrapping purposes
  /controller-manager          Generate a kubeconfig file for the controller manager to use
  /scheduler                   Generate a kubeconfig file for the scheduler to use
etcd                         Generate static Pod manifest file for local etcd
  /local                       Generate the static Pod manifest file for a local, single-node local etcd instance
control-plane                Generate all static Pod manifest files necessary to establish the control plane
  /apiserver                   Generates the kube-apiserver static Pod manifest
  /controller-manager          Generates the kube-controller-manager static Pod manifest
  /scheduler                   Generates the kube-scheduler static Pod manifest
kubelet-start                Write kubelet settings and (re)start the kubelet
wait-control-plane           Wait for the control plane to start
upload-config                Upload the kubeadm and kubelet configuration to a ConfigMap
  /kubeadm                     Upload the kubeadm ClusterConfiguration to a ConfigMap
  /kubelet                     Upload the kubelet component config to a ConfigMap
upload-certs                 Upload certificates to kubeadm-certs
mark-control-plane           Mark a node as a control-plane
bootstrap-token              Generates bootstrap tokens used to join a node to a cluster
kubelet-finalize             Updates settings relevant to the kubelet after TLS bootstrap
  /enable-client-cert-rotation  Enable kubelet client certificate rotation
addon                        Install required addons for passing conformance tests
  /coredns                     Install the CoreDNS addon to a Kubernetes cluster
  /kube-proxy                  Install the kube-proxy addon to a Kubernetes cluster
show-join-command            Show the join command for control-plane and worker node
```

```
kubeadm init [flags]
```

Options

--apiserver-advertise-address string

The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.

--apiserver-bind-port int32 Default: 6443

Port for the API Server to bind to.

--apiserver-cert-extra-sans strings

Optional extra Subject Alternative Names (SANs) to use for the API Server serving certificate. Can be both IP addresses and DNS names.

--cert-dir string Default: "/etc/kubernetes/pki"

The path where to save and store the certificates.

--certificate-key string

Key used to encrypt the control-plane certificates in the kubeadm-certs Secret. The certificate key is a hex encoded string that is an AES key of size 32 bytes.

--config string

Path to a kubeadm configuration file.

--control-plane-endpoint string

Specify a stable IP address or DNS name for the control plane.

--cri-socket string

Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.

--dry-run
  Don't apply any changes; just output what would be done.
--feature-gates string
  A set of key=value pairs that describe feature gates for various features. Options are:
  ControlPlaneKubeletLocalMode=true|false (BETA - default=true)
  NodeLocalCRISocket=true|false (BETA - default=true)
  PublicKeysECDSA=true|false (DEPRECATED - default=false)
  RootlessControlPlane=true|false (ALPHA - default=false)
  WaitForAllControlPlaneComponents=true|false (default=true)
-h, --help
  help for init
--ignore-preflight-errors strings
  A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.
--image-repository string Default: "registry.k8s.io"
  Choose a container registry to pull control plane images from
--kubernetes-version string Default: "stable-1"
  Choose a specific Kubernetes version for the control plane.
--node-name string
  Specify the node name.
--patches string
  Path to a directory that contains files named "target[suffix][+patchtype].extension". For example, "kube-apiserver0+merge.yaml" or just "etcd.json". "target" can be one of "kube-apiserver", "kube-controller-manager", "kube-scheduler", "etcd", "kubeletconfiguration", "corednsdeployment". "patchtype" can be one of "strategic", "merge" or "json" and they match the patch formats supported by kubectl. The default "patchtype" is "strategic". "extension" must be either "json" or "yaml". "suffix" is an optional string that can be used to determine which patches are applied first alpha-numerically.
--pod-network-cidr string
  Specify range of IP addresses for the pod network. If set, the control plane will automatically allocate CIDRs for every node.
--service-cidr string Default: "10.96.0.0/12"
  Use alternative range of IP address for service VIPs.
--service-dns-domain string Default: "cluster.local"
  Use alternative domain for services, e.g. "myorg.internal".
--skip-certificate-key-print
  Don't print the key used to encrypt the control-plane certificates.
--skip-phases strings
  List of phases to be skipped
--skip-token-print
  Skip printing of the default bootstrap token generated by 'kubeadm init'.
--token string
  The token to use for establishing bidirectional trust between nodes and control-plane nodes. The format is [a-z0-9]{6}.[a-z0-9]{16} - e.g. abcdef.0123456789abcdef
--token-ttl duration Default: 24h0m0s
  The duration before the token is automatically deleted (e.g. 1s, 2m, 3h). If set to '0', the token will never expire
--upload-certs
  Upload control-plane certificates to the kubeadm-certs Secret.

Options inherited from parent commands

--rootfs string
  The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Init workflow

`kubeadm init` bootstraps a Kubernetes control plane node by executing the following steps:

1. Runs a series of pre-flight checks to validate the system state before making changes. Some checks only trigger warnings, others are considered errors and will exit kubeadm until the problem is corrected or the user specifies `--ignore-preflight-errors=<list-of-errors>`.

2. Generates a self-signed CA to set up identities for each component in the cluster. The user can provide their own CA cert and/or key by dropping it in the cert directory configured via `--cert-dir` (`/etc/kubernetes/pki` by default). The API server certs will have additional SAN entries for any `--apiserver-cert-extra-sans` arguments, lowercased if necessary.

3. Writes kubeconfig files in `/etc/kubernetes/` for the kubelet, the controller-manager, and the scheduler to connect to the API server, each with its own identity. Also additional kubeconfig files are written, for kubeadm as administrative entity (`admin.conf`) and for a super admin user that can bypass RBAC (`super-admin.conf`).

4. Generates static Pod manifests for the API server, controller-manager and scheduler. In case an external etcd is not provided, an additional static Pod manifest is generated for etcd.

   Static Pod manifests are written to `/etc/kubernetes/manifests`; the kubelet watches this directory for Pods to create on startup.

Once control plane Pods are up and running, the `kubeadm init` sequence can continue.

5. Apply labels and taints to the control plane node so that no additional workloads will run there.
6. Generates the token that additional nodes can use to register themselves with a control plane in the future. Optionally, the user can provide a token via `--token`, as described in the kubeadm token documents.
7. Makes all the necessary configurations for allowing node joining with the Bootstrap Tokens and TLS Bootstrap mechanism:
   - Write a ConfigMap for making available all the information required for joining, and set up related RBAC access rules.
   - Let Bootstrap Tokens access the CSR signing API.
   - Configure auto-approval for new CSR requests.

   See kubeadm join for additional information.
8. Installs a DNS server (CoreDNS) and the kube-proxy addon components via the API server. In Kubernetes version 1.11 and later CoreDNS is the default DNS server. Please note that although the DNS server is deployed, it will not be scheduled until CNI is installed.

   Warning:

   kube-dns usage with kubeadm is deprecated as of v1.18 and is removed in v1.21.

Using init phases with kubeadm

kubeadm allows you to create a control plane node in phases using the `kubeadm init phase` command.

To view the ordered list of phases and sub-phases you can call `kubeadm init --help`. The list will be located at the top of the help screen and each phase will have a description next to it. Note that by calling `kubeadm init` all of the phases and sub-phases will be executed in this exact order.

Some phases have unique flags, so if you want to have a look at the list of available options add `--help`, for example:

```
sudo kubeadm init phase control-plane controller-manager --help
```

You can also use `--help` to see the list of sub-phases for a certain parent phase:

```
sudo kubeadm init phase control-plane --help
```

`kubeadm init` also exposes a flag called `--skip-phases` that can be used to skip certain phases. The flag accepts a list of phase names and the names can be taken from the above ordered list.

An example:

```
sudo kubeadm init phase control-plane all --config=configfile.yaml
sudo kubeadm init phase etcd local --config=configfile.yaml
# you can now modify the control plane and etcd manifest files
sudo kubeadm init --skip-phases=control-plane,etcd --config=configfile.yaml
```

What this example would do is write the manifest files for the control plane and etcd in `/etc/kubernetes/manifests` based on the configuration in `configfile.yaml`. This allows you to modify the files and then skip these phases using `--skip-phases`. By calling the last command you will create a control plane node with the custom manifest files.

FEATURE STATE: `Kubernetes v1.22 [beta]`

Alternatively, you can use the `skipPhases` field under `InitConfiguration`.

Using kubeadm init with a configuration file

Caution:

The configuration file is still considered beta and may change in future versions.

It's possible to configure `kubeadm init` with a configuration file instead of command line flags, and some more advanced features may only be available as configuration file options. This file is passed using the `--config` flag and it must contain a `ClusterConfiguration` structure and optionally more structures separated by `---\n`. Mixing `--config` with others flags may not be allowed in some cases.

The default configuration can be printed out using the kubeadm config print command.

If your configuration is not using the latest version it is **recommended** that you migrate using the kubeadm config migrate command.

For more information on the fields and usage of the configuration you can navigate to our API reference page.

Using kubeadm init with feature gates

kubeadm supports a set of feature gates that are unique to kubeadm and can only be applied during cluster creation with `kubeadm init`. These features can control the behavior of the cluster. Feature gates are removed after a feature graduates to GA.

To pass a feature gate you can either use the `--feature-gates` flag for `kubeadm init`, or you can add items into the `featureGates` field when you pass a configuration file using `--config`.

Passing feature gates for core Kubernetes components directly to kubeadm is not supported. Instead, it is possible to pass them by Customizing components with the kubeadm API.

List of feature gates:

Feature	Default	Alpha	Beta	GA
`ControlPlaneKubeletLocalMode`	`true`	1.31	1.33	-
`NodeLocalCRISocket`	`true`	1.32	1.34	-
`WaitForAllControlPlaneComponents`	`true`	1.30	1.33	1.34

Note:

Once a feature gate goes GA its value becomes locked to `true` by default.

Feature gate descriptions:

`ControlPlaneKubeletLocalMode`
: With this feature gate enabled, when joining a new control plane node, kubeadm will configure the kubelet to connect to the local kube-apiserver. This ensures that there will not be a violation of the version skew policy during rolling upgrades.

`NodeLocalCRISocket`
: With this feature gate enabled, kubeadm will read/write the CRI socket for each node from/to the file `/var/lib/kubelet/instance-config.yaml` instead of reading/writing it from/to the annotation `kubeadm.alpha.kubernetes.io/cri-socket` on the Node object. The new file is applied as an instance configuration patch, before any other user managed patches are applied when the `--patches` flag is used. It contains a single field `containerRuntimeEndpoint` from the KubeletConfiguration file format. If the feature gate is enabled during upgrade, but the file `/var/lib/kubelet/instance-config.yaml` does not exist yet, kubeadm will attempt to read the CRI socket value from the file `/var/lib/kubelet/kubeadm-flags.env`.

`WaitForAllControlPlaneComponents`
: With this feature gate enabled, kubeadm will wait for all control plane components (kube-apiserver, kube-controller-manager, kube-scheduler) on a control plane node to report status 200 on their `/livez` or `/healthz` endpoints. These checks are performed on `https://ADDRESS:PORT/ENDPOINT`.
  - `PORT` is taken from `--secure-port` of a component.
  - `ADDRESS` is `--advertise-address` for kube-apiserver and `--bind-address` for the kube-controller-manager and kube-scheduler.
  - `ENDPOINT` is only `/healthz` for kube-controller-manager until it supports `/livez` as well.

  If you specify custom `ADDRESS` or `PORT` in the kubeadm configuration they will be respected. Without the feature gate enabled, kubeadm will only wait for the kube-apiserver on a control plane node to become ready. The wait process starts right after the kubelet on the host is started by kubeadm. You are advised to enable this feature gate in case you wish to observe a ready state from all control plane components during the `kubeadm init` or `kubeadm join` command execution.

List of deprecated feature gates:

Feature	Default	Alpha	Beta	GA	Deprecated
`PublicKeysECDSA`	`false`	1.19	-	-	1.31
`RootlessControlPlane`	`false`	1.22	-	-	1.31

Feature gate descriptions:

`PublicKeysECDSA`
: Can be used to create a cluster that uses ECDSA certificates instead of the default RSA algorithm. Renewal of existing ECDSA certificates is also supported using `kubeadm certs renew`, but you cannot switch between the RSA and ECDSA algorithms on the fly or during upgrades. Kubernetes versions before v1.31 had a bug where keys in generated kubeconfig files were set use RSA, even when you had enabled the `PublicKeysECDSA` feature gate. This feature gate is deprecated in favor of the `encryptionAlgorithm` functionality available in kubeadm v1beta4.

`RootlessControlPlane`
: Setting this flag configures the kubeadm deployed control plane component static Pod containers for `kube-apiserver`, `kube-controller-manager`, `kube-scheduler` and `etcd` to run as non-root users. If the flag is not set, those components run as root. You can change the value of this feature gate before you upgrade to a newer version of Kubernetes.

List of removed feature gates:

Feature	Alpha	Beta	GA	Removed
`EtcdLearnerMode`	1.27	1.29	1.32	1.33
`IPv6DualStack`	1.16	1.21	1.23	1.24
`UnversionedKubeletConfigMap`	1.22	1.23	1.25	1.26
`UpgradeAddonsBeforeControlPlane`	1.28	-	-	1.31

Feature gate descriptions:

`EtcdLearnerMode`
: When joining a new control plane node, a new etcd member will be created as a learner and promoted to a voting member only after the etcd data are fully aligned.

`IPv6DualStack`
: This flag helps to configure components dual stack when the feature is in progress. For more details on Kubernetes dual-stack support see Dual-stack support with kubeadm.

`UnversionedKubeletConfigMap`
: This flag controls the name of the ConfigMap where kubeadm stores kubelet configuration data. With this flag not specified or set to `true`, the ConfigMap is named `kubelet-config`. If you set this flag to `false`, the name of the ConfigMap includes the major and minor version for Kubernetes (for example: `kubelet-config-1.34`). Kubeadm ensures that RBAC rules for reading and writing that ConfigMap are appropriate for the value you set. When kubeadm writes this ConfigMap (during `kubeadm init` or `kubeadm upgrade apply`), kubeadm respects the value of `UnversionedKubeletConfigMap`. When reading that ConfigMap (during `kubeadm join`, `kubeadm reset`, `kubeadm upgrade`...), kubeadm attempts to use unversioned ConfigMap name first. If that does not succeed, kubeadm falls back to using the legacy (versioned) name for that ConfigMap.

`UpgradeAddonsBeforeControlPlane`
: This feature gate has been removed. It was introduced in v1.28 as a deprecated feature and then removed in v1.31. For documentation on older versions, please switch to the corresponding website version.

Adding kube-proxy parameters

For information about kube-proxy parameters in the kubeadm configuration see:

- kube-proxy reference

For information about enabling IPVS mode with kubeadm see:

- IPVS

Passing custom flags to control plane components

For information about passing flags to control plane components see:

- control-plane-flags

Running kubeadm without an Internet connection

For running kubeadm without an Internet connection you have to pre-pull the required control plane images.

You can list and pull the images using the `kubeadm config images` sub-command:

```
kubeadm config images list
kubeadm config images pull
```

You can pass `--config` to the above commands with a kubeadm configuration file to control the `kubernetesVersion` and `imageRepository` fields.

All default `registry.k8s.io` images that kubeadm requires support multiple architectures.

Using custom images

By default, kubeadm pulls images from `registry.k8s.io`. If the requested Kubernetes version is a CI label (such as `ci/latest`) `gcr.io/k8s-staging-ci-images` is used.

You can override this behavior by using kubeadm with a configuration file. Allowed customization are:

- To provide `kubernetesVersion` which affects the version of the images.
- To provide an alternative `imageRepository` to be used instead of `registry.k8s.io`.
- To provide a specific `imageRepository` and `imageTag` for etcd or CoreDNS.

Image paths between the default `registry.k8s.io` and a custom repository specified using `imageRepository` may differ for backwards compatibility reasons. For example, one image might have a subpath at `registry.k8s.io/subpath/image`, but be defaulted to `my.customrepository.io/image` when using a custom repository.

To ensure you push the images to your custom repository in paths that kubeadm can consume, you must:

- Pull images from the defaults paths at `registry.k8s.io` using `kubeadm config images {list|pull}`.
- Push images to the paths from `kubeadm config images list --config=config.yaml`, where `config.yaml` contains the custom `imageRepository`, and/or `imageTag` for etcd and CoreDNS.
- Pass the same `config.yaml` to `kubeadm init`.

Custom sandbox (pause) images

To set a custom image for these you need to configure this in your container runtime to use the image. Consult the documentation for your container runtime to find out how to change this setting; for selected container runtimes, you can also find advice within the Container Runtimes topic.

Uploading control plane certificates to the cluster

By adding the flag `--upload-certs` to `kubeadm init` you can temporary upload the control plane certificates to a Secret in the cluster. Please note that this Secret will expire automatically after 2 hours. The certificates are encrypted using a 32byte key that can be specified using `--certificate-key`. The same key can be used to download the certificates when additional control plane nodes are joining, by passing `--control-plane` and `--certificate-key` to `kubeadm join`.

The following phase command can be used to re-upload the certificates after expiration:

```
kubeadm init phase upload-certs --upload-certs --config=SOME_YAML_FILE
```

Note:

A predefined `certificateKey` can be provided in `InitConfiguration` when passing the configuration file with `--config`.

If a predefined certificate key is not passed to `kubeadm init` and `kubeadm init phase upload-certs` a new key will be generated automatically.

The following command can be used to generate a new key on demand:

```
kubeadm certs certificate-key
```

Certificate management with kubeadm

For detailed information on certificate management with kubeadm see Certificate Management with kubeadm. The document includes information about using external CA, custom certificates and certificate renewal.

Managing the kubeadm drop-in file for the kubelet

The `kubeadm` package ships with a configuration file for running the `kubelet` by `systemd`. Note that the kubeadm CLI never touches this drop-in file. This drop-in file is part of the kubeadm DEB/RPM package.

For further information, see Managing the kubeadm drop-in file for systemd.

Use kubeadm with CRI runtimes

By default, kubeadm attempts to detect your container runtime. For more details on this detection, see the kubeadm CRI installation guide.

Setting the node name

By default, kubeadm assigns a node name based on a machine's host address. You can override this setting with the `--node-name` flag. The flag passes the appropriate `--hostname-override` value to the kubelet.

Be aware that overriding the hostname can interfere with cloud providers.

Automating kubeadm

Rather than copying the token you obtained from `kubeadm init` to each node, as in the basic kubeadm tutorial, you can parallelize the token distribution for easier automation. To implement this automation, you must know the IP address that the control plane node will have after it is started, or use a DNS name or an address of a load balancer.

1. Generate a token. This token must have the form `<6 character string>.<16 character string>`. More formally, it must match the regex: `[a-z0-9]{6}\.[a-z0-9]{16}`.

   kubeadm can generate a token for you:

   ```
   kubeadm token generate
   ```

2. Start both the control plane node and the worker nodes concurrently with this token. As they come up they should find each other and form the cluster. The same `--token` argument can be used on both `kubeadm init` and `kubeadm join`.

3. Similar can be done for `--certificate-key` when joining additional control plane nodes. The key can be generated using:

   ```
   kubeadm certs certificate-key
   ```

Once the cluster is up, you can use the `/etc/kubernetes/admin.conf` file from a control plane node to talk to the cluster with administrator credentials or Generating kubeconfig files for additional users.

Note that this style of bootstrap has some relaxed security guarantees because it does not allow the root CA hash to be validated with `--discovery-token-ca-cert-hash` (since it's not generated when the nodes are provisioned). For details, see the kubeadm join.

What's next

- kubeadm init phase to understand more about `kubeadm init` phases
- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster
- kubeadm upgrade to upgrade a Kubernetes cluster to a newer version
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`

Setup tools

Kubeadm

---

kubeadm reset

Performs a best effort revert of changes made by `kubeadm init` or `kubeadm join`.

Synopsis

Performs a best effort revert of changes made to this host by 'kubeadm init' or 'kubeadm join'

The "reset" command executes the following phases:

```
preflight           Run reset pre-flight checks
remove-etcd-member  Remove a local etcd member.
cleanup-node        Run cleanup node.
```

```
kubeadm reset [flags]
```

Options

--cert-dir string Default: "/etc/kubernetes/pki"

The path to the directory where the certificates are stored. If specified, clean this directory.

--cleanup-tmp-dir

Cleanup the "/etc/kubernetes/tmp" directory

--config string

Path to a kubeadm configuration file.

--cri-socket string

Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.

--dry-run

Don't apply any changes; just output what would be done.

-f, --force

Reset the node without prompting for confirmation.

-h, --help

help for reset

--ignore-preflight-errors strings

A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.

--kubeconfig string Default: "/etc/kubernetes/admin.conf"

The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file.

--skip-phases strings

List of phases to be skipped

Options inherited from parent commands

--rootfs string

The path to the 'real' host root filesystem. This will cause kubeadm to chroot into the provided path.

Reset workflow

`kubeadm reset` is responsible for cleaning up a node local file system from files that were created using the `kubeadm init` or `kubeadm join` commands. For control-plane nodes `reset` also removes the local stacked etcd member of this node from the etcd cluster.

`kubeadm reset phase` can be used to execute the separate phases of the above workflow. To skip a list of phases you can use the `--skip-phases` flag, which works in a similar way to the `kubeadm join` and `kubeadm init` phase runners.

`kubeadm reset` also supports the `--config` flag for passing a `ResetConfiguration` structure.

Cleanup of external etcd members

`kubeadm reset` will not delete any etcd data if external etcd is used. This means that if you run `kubeadm init` again using the same etcd endpoints, you will see state from previous clusters.

To wipe etcd data it is recommended you use a client like etcdctl, such as:

```
etcdctl del "" --prefix
```

See the etcd documentation for more information.

Cleanup of CNI configuration

CNI plugins use the directory `/etc/cni/net.d` to store their configuration. The `kubeadm reset` command does not cleanup that directory. Leaving the configuration of a CNI plugin on a host can be problematic if the same host is later used as a new Kubernetes node and a different CNI plugin happens to be deployed in that cluster. It can result in a configuration conflict between CNI plugins.

To cleanup the directory, backup its contents if needed and then execute the following command:

```
sudo rm -rf /etc/cni/net.d
```

Cleanup of network traffic rules

The `kubeadm reset` command does not clean any iptables, nftables or IPVS rules applied to the host by kube-proxy. A control loop in kube-proxy ensures that the rules on each node host are synchronized. For additional details please see Virtual IPs and Service Proxies.

Leaving the rules without cleanup should not cause any issues if the host is later reused as a Kubernetes node or if it will serve a different purpose.

If you wish to perform this cleanup, you can use the same kube-proxy container which was used in your cluster and the `--cleanup` flag of the `kube-proxy` binary:

```
docker run --privileged --rm registry.k8s.io/kube-proxy:v1.34.0 sh -c "kube-proxy --cleanup && echo DONE"
```

The output of the above command should print `DONE` at the end. Instead of Docker, you can use your preferred container runtime to start the container.

Cleanup of $HOME/.kube

The `$HOME/.kube` directory typically contains configuration files and kubectl cache. While not cleaning the contents of `$HOME/.kube/cache` is not an issue, there is one important file in the directory. That is `$HOME/.kube/config` and it is used by kubectl to authenticate to the Kubernetes API server. After `kubeadm init` finishes, the user is instructed to copy the `/etc/kubernetes/admin.conf` file to the `$HOME/.kube/config` location and grant the current user access to it.

The `kubeadm reset` command does not clean any of the contents of the `$HOME/.kube` directory. Leaving the `$HOME/.kube/config` file without deleting it, can be problematic depending on who will have access to this host after `kubeadm reset` was called. If the same cluster continues to exist, it is highly recommended to delete the file, as the admin credentials stored in it will continue to be valid.

To cleanup the directory, examine its contents, perform backup if needed and execute the following command:

```
rm -rf $HOME/.kube
```

Graceful kube-apiserver shutdown

If you have your `kube-apiserver` configured with the `--shutdown-delay-duration` flag, you can run the following commands to attempt a graceful shutdown for the running API server Pod, before you run `kubeadm reset`:

```
yq eval -i '.spec.containers[0].command = []' /etc/kubernetes/manifests/kube-apiserver.yaml
timeout 60 sh -c 'while pgrep kube-apiserver >/dev/null; do sleep 1; done' || true
```

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster