ResetConfiguration and UpgradeConfiguration API definition

Dave Chen (@chendave)

Motivation

Kubeadm has the dedicated configuration file for the command of *init* and *join*, but there is no such configuration available for ***reset*** and ***upgrade***, the problem is that:

- kubeadm has to accept quite a lot of flags and no way to centralize the configuration for those flags.
- It's quite confusing that the upgrade command accepts a configuration which is a kind of *InitConfiguration* or *ClusterConfiguration* or even componentConfig.

As this issue 2489 said, it's better for kubeadm to provide two separate configuration files for *upgrade* and *reset* respectively to make it consistent with the init and join command.

Goals

- Define ResetConfiguration and UpgradeConfiguration API types for *reset* and *upgrade* commands. (Can we start it from v1beta3 as InitConfiguration and JoinConfiguration?)

- Make sure the upgrade command accepts only the UpgradeConfiguration (Deprecate the the acceptance of *InitConfiguration, ClusterConfiguration or ComponentConfig?*)

Proposal

- ResetConfiguration

There are no such lot of configuration needed for reset, overall, the resetConfiguration will looks like this,

Code path: k8s.io/kubernetes/cmd/kubeadm/app/apis/kubeadm/v1beta3/types.go

```
// ResetConfiguration contains a list of elements ...
type ResetConfiguration struct {

metav1.TypeMeta `json:",inline"`

// SkipPhases is a list of phases to skip during command execution.
// The list of phases can be obtained with the "kubeadm reset phase --help"
command.
// The flag "--skip-phases" takes precedence over this field.
// +optional
SkipPhases []string `json:"skipPhases,omitempty"`

// ForceReset flag instruct kubeadm to reset the node without prompting for
confirmation
// The flag "--force" takes precedence over this field.
// +optional
ForceReset bool `json:"forceReset,omitempty"`

// CertificatesDir specifies where to store or look for all required
certificates.
// cannot mix the config for both the resetConfig and flag to make it
consistent with kubeadm init.
// +optional
CertificatesDir string `json:"certificatesDir,omitempty"`

// CRISocket is used to retrieve container runtime info. This information
will be annotated to the Node API object, for later re-use. (call
SetNodeRegistrationDynamicDefaults to set the default value)
// +optional
CRISocket string `json:"criSocket,omitempty"`

// IgnorePreflightErrors provides a slice of pre-flight errors to be
ignored when the current node is registered.
// +optional
IgnorePreflightErrors []string `json:"ignorePreflightErrors,omitempty"`

// CleanupTmpDir cleanup the "/etc/kubernetes/tmp" directory if specified.
// +optional
CleanupTmpDir bool `json:"cleanupTmpDir ,omitempty"`
}
```

Update the structure of resetData to be something like this,

```
type resetData struct {
certificatesDir         string
client                  clientset.Interface
criSocketPath           string
forceReset              bool
ignorePreflightErrors sets.String
inputReader             io.Reader
outputWriter            io.Writer
cfg                     *kubeadmapi.InitConfiguration
resetCfg                *kubeadmapi.ResetConfiguration // this will hold
the the reset config file
dryRun                  bool
}
```

Several of notes on the flags,

```
resetCfg
```

The below flag will point to the reset configuration file if it is specified,

```
options.AddConfigFlag(fs, &flags.cfgPath)
```

```
certificatesDir
```

Set the `certificatesDir` of `resetData` to be the value defined from the configuration file, and this is not allowed to be both specified by the flag and config file to respect the legacy allowed mixed arguments which is defined by the method of,

```
func isAllowedFlag(flagName string) bool
```

```
 criSocketPath
```

To be consistent with the *join* command, the value specified by the flag will overwrite the value defined in the configuration file.

```
ignorePreflightErrors
```

Kubeadm will honor the way it does in the join / init, the ignored errors specified by the flag and configuration will be merged together.

`dryRun`

This is not configurable in the config file to make it consistent with init /join, i.e. if a user wants to perform a dry run, he/she can only pass it via flag. **Open for discussion.**

`kubeconfig`

The use of kubeconfig is quite simple, it is normally used to build a client, and the flag always has a default value set, does this need to be a configure item in the config file? **Open for discussion.**

- UpgradeConfiguration

Went through each flags defined for kubeadm upgrade, the versioned UpgradeConfiguration might look like this,

Code path: k8s.io/kubernetes/cmd/kubeadm/app/apis/kubeadm/v1beta3/types.go

```
// UpgradeConfiguration contains a list of elements ...
type UpgradeConfiguration struct {

metav1.TypeMeta `json:",inline"`

// This should be embedded in the clusterCfg
// ComponentConfigs is the path to the component configs known to kubeadm, e.g. kube-proxy or kubelet config.
// flags are ignored if the config is defined, or else the flag will overwrite the empty config.
// +optional
ComponentConfigs string `json:"componentConfigs,omitempty"`

// AllowExperimentalUpgrades instruct kubeadm to show unstable versions of Kubernetes as an upgrade
// alternative and allow upgrading to an alpha/beta/release candidate version of Kubernetes.
// +optional
AllowExperimentalUpgrades bool `json:"allowExperimentalUpgrades,omitempty"`
```

```
// Enable AllowRCUpgrades  will show release candidate versions of
Kubernetes as an upgrade alternative and
// allow upgrading to a release candidate version of Kubernetes.
// +optional
AllowRCUpgrades bool `json:"allowRCUpgrades ,omitempty"`

// IgnorePreflightErrors provides a slice of pre-flight errors to be
ignored when the current node is registered.
// +optional
IgnorePreflightErrors []string `json:"ignorePreflightErrors,omitempty"`

// PrintConfig specifies whether the configuration file that will be used
in the upgrade should be printed or not.
// +optional
PrintConfig bool `json:"printConfig,omitempty"`

// ForceUpgrade flag instruct kubeadm to upgrade the cluster without
prompting for confirmation
// The flag "--force" takes precedence over this field.
// +optional
ForceUpgrade bool `json:"forceUpgrade,omitempty"`


// LocalAPIEndpoint represents the endpoint of the API server instance
that's deployed on this control plane node
// In HA setups, this differs from
ClusterConfiguration.ControlPlaneEndpoint in the sense that
ControlPlaneEndpoint
// is the global endpoint for the cluster, which then loadbalances the
requests to each individual API server. This
// configuration object lets you customize what IP/DNS name and port the
local API server advertises it's accessible
// on. By default, kubeadm tries to auto-detect the IP of the default
interface and use that, but in case that process
// fails you may set the desired value here.
// +optional
LocalAPIEndpoint APIEndpoint `json:"localAPIEndpoint,omitempty"` (move to
internal type)

// CertificateRenewal instructs kubeadm to execute certificate renewal
during upgrades
// Default: true
// +optional
```

```
CertificateRenewal bool `json:"certificateRenewal,omitempty"`

// EtcdUpgrade instructs kubeadm to execute etcd upgrade during upgrades
// Default: true
// +optional
EtcdUpgrade bool `json:"etcdUpgrade,omitempty"`

// Patches contains options related to applying patches to components
deployed by kubeadm during `kubeadm upgrade`.
// +optional
~~PatchesDir string `json:"patchesDir,omitempty"`~~
Patches *Patches `json:"patches,omitempty"`

// SkipPhases is a list of phases to skip during command execution.
// The list of phases can be obtained with the "kubeadm reset phase --help"
command.
// The flag "--skip-phases" takes precedence over this field.
// +optional
SkipPhases []string `json:"skipPhases,omitempty"`

// merge below three config into one config item: manifestPath
// ApiServerManifestPath is the path to API server manifest (default value:
/etc/kubernetes/manifest/api...)
// +optional
ApiServerManifestPath string `json:"apiServerManifestPath,omitempty"`

// ControllerManagerManifestPath is the path to the API server manifest
(default value: /etc/kubernetes/manifest/control...)
// +optional
ControllerManagerManifestPath string
`json:"controllerManagerManifestPath,omitempty"`

// SchedulerManifestPath is the path to API server manifest (default value:
/etc/kubernetes/manifest/sche...)
// +optional
SchedulerManifestPath string `json:"schedulerManifestPath,omitempty"`


// ContextLines is the number of lines of context in the diff
// +optional
ContextLines int32 `json:"contextLines,omitempty"

// timeout controls the timeout for different purpose, e.g. the timeout
```

```
// that is used for the API server to appear or timeout of upgrading the
// static pod manifest.
// see: TimeoutForControlPlane or UpgradeManifestTimeout
// + optional
timeout map[string]*metav1.Duration
}
```

Following what the init command does, Kubeadm will get the clusterConfiguration from the cluster, and set this as a field for internal UpgradeConfiguration.

E.g.
Code path: k8s.io/kubernetes/cmd/kubeadm/app/apis/kubeadm/types.go

```
type UpgradeConfiguration struct {
metav1.TypeMeta

// ClusterConfiguration holds the cluster-wide information, and embeds that
struct (which can be (un)marshalled separately as well)
// When InitConfiguration is marshalled to bytes in the external version,
this information IS NOT preserved (which can be seen from
// the `json:"-"` tag in the external variant of these API types.
ClusterConfiguration `json:"-"`

// LocalAPIEndpoint represents the endpoint of the API server instance
that's deployed on this control plane node
// In HA setups, this differs from
ClusterConfiguration.ControlPlaneEndpoint in the sense that
ControlPlaneEndpoint
// is the global endpoint for the cluster, which then loadbalances the
requests to each individual API server. This
// configuration object lets you customize what IP/DNS name and port the
local API server advertises it's accessible
// on. By default, kubeadm tries to auto-detect the IP of the default
interface and use that, but in case that process
// fails you may set the desired value here.
// +optional
LocalAPIEndpoint APIEndpoint `json:"localAPIEndpoint,omitempty"`
…

}
```

Kubeadm doesn't need to build a type like upgradeData in the top level, as kubeadm upgrade doesn't mean to be able run on its own, instead each sub-command should accept the flag to set the path of the upgrade configuration. This is already true for the below command,

```
kubeadm upgrade plan
kubeadm upgrade diff
kubeadm upgrade apply
```

But the cfg here may imply a different things,,

```
fs.StringVar(cfgPath, CfgPath, *cfgPath, "Path to a kubeadm configuration file.")
```

For example, for the command *kubeadm upgrade apply,* the config might be either a initCfg or ComponentCfg. But for the case like this, kubeadm would deprecate the use of the initCfg, clusterCfg or ComponentCfg, but instead, it will only accept the *UpgradeConfiguration* in the long term.

Kubeadm still needs to fetch an initCfg or clusterCfg from the cluster to build some data for cluster upgrade, but this is invisible to the end-user, there is no need to pass a initCfg or clusterCfg from the flag.

As a concrete example, kubeadm will fetch the initCfg from the cluster and pass the value of *LocalAPIEndpoint* to the *UpgradeConfiguration.*

NOTEs on couple of the flags,

```
dryRun
```

This is not configurable in the configuration to make it consistent with init /join, i.e. if a user wants to perform a dry run, he/she can only pass it via flag. **Open for discussion.**

```
kubeconfig
```

The use of kubeconfig is quite simple, it is normally used to build a client, and the flag always has a default value set, does this need to be a configure item in the config file? **Open for discussion.**

```
feature-gates
```

Not sure if we need to make it as a config item, **Open for discussion.**
("upgrade apply" can set the feature gate, will merge this in the clustercfg per the suggestion from the pacoxu)

```
Output and show-managed-fields
```

Both of them are from an experimental feature, we can defer to the discussion when the feature is stable and not marked with experimental.

```
ComponentConfigs
```

ComponentConfigs is the path to the component configs known to kubeadm, e.g. *kube-proxy* or *kubelet-config*. We need this to be compatible with current implementation, since the *--config* might point to a component config and consume it.

Note that flags are ignored if this is configured in the configuration file, or else the flag will overwrite the empty config(or might continue to be ignored), **I didn't see a similar case for this, which is the correct way to tackle it?**

Mixed configuration is only allowed for the flag defined by the below method, no new mix configuration will be added to the list.

```
func isAllowedFlag(flagName string) bool
```

Reference

https://github.com/kubernetes/kubeadm/issues/2489