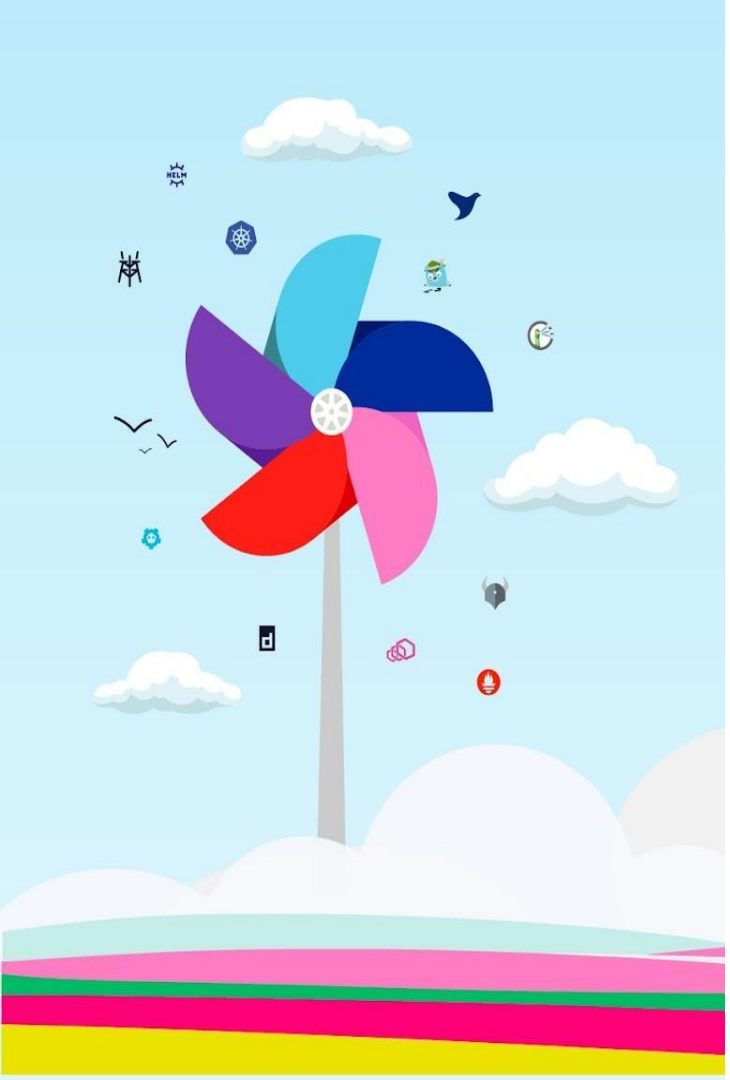


KubeCon | CloudNativeCon
Europe 2023

The Ins and Outs of the Cloud Provider in Kubernetes

Bridget Kromhout
Principal Product Manager
Microsoft

Michael McCune
Principal Software Engineer
Red Hat

Joel Speed
Principal Software Engineer
Red Hat

Hi!

- Bridget Kromhout
- Product Manager @ Microsoft
- Upstream open source
- Kubernetes ecosystem for 6 years?!?
- bridgetkromhout on GitHub

- Michael McCune
- Software Engineer @ Red Hat
- Cloud Infrastructure and Autoscaling
- Kubernetes ecosystem for 6 years
- elmiko on GitHub

- Joel Speed
- Software Engineer @ Red Hat
- Cloud Infrastructure
- ex-SRE/ex-CKA
- Kubernetes ecosystem for about 6 years
- joelspeed on GitHub

Setting Foundational Context

What is the Cloud Provider in Kubernetes?

- “Cloud Provider” can have multiple meanings when using it in reference to Kubernetes.
- Most commonly it refers to the infrastructure provider that Kubernetes is deployed on, for example: AWS, Azure, GCP, OpenStack, etc.
- It can also be used to refer to the specific Kubernetes component that handles interactions with the infrastructure, for example: “is this cluster configured for an external cloud provider?”

KCM, CPI, CCM: Three Acronyms You Should Know

- KCM is the kube-controller-manager, a process that runs the core control loops for Kubernetes.
- KCM also contains “cloud controllers”, which handle infrastructure-specific integrations.
- CPI is the “Cloud Provider Interface”, a reference interface for creating cloud controllers that can interact with Kubernetes.
  - Note: this is not CAPI
- CCM is the cloud-controller-manager, a process that runs cloud controllers in their own containers.

Why Should You Care About CCMs?

The Kubernetes community has been working on a refactoring of the cloud controller managers that will make the core Kubernetes code easier to maintain, and also give infrastructure providers newfound freedom to innovate.

- “Out-of-tree” soon to be default in Kubernetes.
- Infrastructure specific code migrating out of core Kubernetes.
- Older code is being removed from the core Kubernetes.
- Cloud Controller Managers are maintained in separate repositories.

“In-tree” and “Out-of-tree” Providers

As introduced in KEP 1179, the Kubernetes community is working to migrate provider specific (e.g. AWS, Azure, GCP) code from the core of Kubernetes (kubernetes/kubernetes repository) into separate repositories. The new provider specific repositories contain the code for those provider’s Cloud Controller Manager.

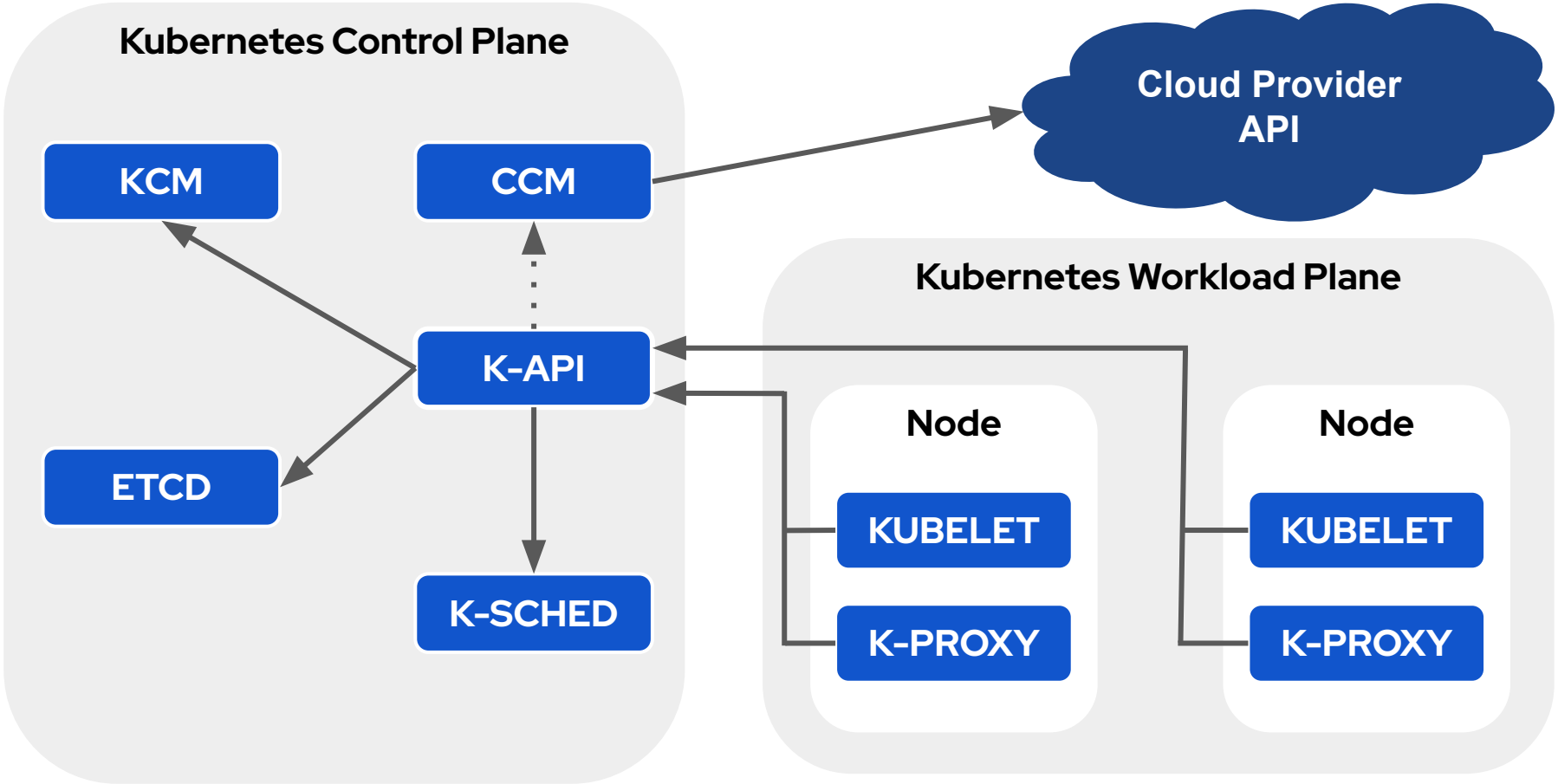
This process has been colloquially referred to as “moving out of tree”. The “tree” in this context is the kubernetes/kubernetes code. When people refer to “in-tree cloud providers”, they are referring to the KCM-embedded cloud controllers. When referring to “out-of-tree”, they are referencing the new CCMs.

What are Cloud Controller Managers?

Containers that run within a Kubernetes cluster to perceive and manage integration with the underlying infrastructure provider.

- Four controllers; Node, Node Lifecycle, Route, and Service.
- Node and Node Lifecycle controllers inform about instance presence and zonal awareness metadata.
- Route controller enables connections between containers running on different nodes .
- Service controller integrates with infrastructure to associate platform resources with Service objects.

CCM within the Cluster



Node Controller Details

- Responsible for updating status and labels on Nodes.
- Removes the `node.cloudprovider.kubernetes.io/uninitialized` taint from new Nodes.
- Adds network addresses and hostnames to the Node status.
- Synchronizes zonal and regional information between instances and Node by applying the well-known Kubernetes labels:
  - `topology.kubernetes.io/region`
  - `topology.kubernetes.io/zone`
- The cloud-provider library also has support for converting and synchronizing the deprecated labels for zones and regions. Note: not all cloud-providers implement this in the same manner.

Node Lifecycle Controller Details

- Responsible for updating and deleting Nodes that have been removed or shutdown in the cloud provider.
- Deletes Nodes where the instances have been removed from the cloud provider.
- Adds the `node.cloudprovider.kubernetes.io/shutdown` taint when an instance is in the process of being shutdown by the cloud provider.
- Removes the `node.cloudprovider.kubernetes.io/shutdown` taint if a previously shutdown instance becomes active again by advertising the `Ready` condition as true.

Route Controller Details

- Responsible for configuring routes appropriately in the cloud provider so that containers on different Nodes can communicate with each other.
- Reacts to Node addition, deletion, and update.
- Tracks Pod CIDRs on a Node to maintain network routes.
  - This behavior is highly provider specific; not all providers implement this interface at all, or in the same manner.

Service Controller Details

- Responsible for creating, deleting, and updating load balancers in the cloud provider to match Service objects.
- Reacts to changes in the `LoadBalancers` field of Service objects, creating load balancers when set to true, and deleting them when changing from true to false.
- Updates load balancer targets when Node objects change IP address.

Deprecation Status in 1.27 Kubernetes

- AWS and OpenStack in-tree provider code has been removed
- Azure, GCE, and vSphere in-tree code still available
  - These providers all have CCMs that are released as general availability.
  - Research the provider you are using, certain combinations of networking and storage with CCMs have reported issues. These issues are unique to each platform and care should be taken during the planning stages of any migration.
  - These providers are planned for removal, but no dates have been announced yet.

Operating Cloud Controller Managers

When to Run CCMs

- Early!
- The CCM becomes part of the bootstrap process of the cluster
- Kubelet taints new nodes `node.cloudprovider.kubernetes.io/uninitialized` and expects the node controller to initialise the node with cloud specific information
- You may need to run this before CNI

Where to Run CCMs

- Can be run anywhere
- Treat them as if they are kube-controller-manager like
- Likely to be run on the control plane nodes (higher security/fewer workloads)
- The controllers will need provider credentials, so will want to think about the implications of that privilege
- Some providers may require an additional daemon on each node (eg. Azure); this should be run as a DaemonSet

How to Run CCMs

- Provider may include an example set of manifests to show how they can be run
- If you run other control plane pods as static pods, you can do that for CCM too, but can be run as a Deployment
- Your CNI may depend on the CCM to initialise the Node addresses, in which case your CCM must:
  - Tolerate the `node.kubernetes.io/not-ready` taint
  - Use host networking, with a direct connection to the Kube API server
- Optional: Use the `--use-service-account-credentials` flag to use built-in, fine grained permissions
- Kubernetes CCM docs have more detail

Planning for High Availability

- To avoid downtime, may want to run multiple replicas of the CCM
- Use leader election when running multiple replicas to avoid conflicting control loops
- A PodDisruptionBudget can help to avoid disruption by preserving a minimum number of Pods during scaling events
- When no CCMs are running, you may see:
  - Delays in initialising Nodes
  - Delays in removing Nodes
  - Delays in configuring load balancers, in-turn leading to service disruption

Migrating from In-tree to Out-of-tree

- For a single replica, ensure Kube Controller Manager disables the in-tree cloud provider, then start the CCM
- For multiple replicas, can either:
  - Ensure all KCM pods are configured to not have the cloud provider running, then enable CCM (may result in a small downtime for the cloud loops)
  - Use the HA leader migration mechanism developed to aid this process, which hands over the control from the KCM to CCM during an upgrade
- Detailed migration steps can be found in the Kubernetes CCM migration docs

How to Spot Problems

- Is your Node still tainted as uninitialized?
  - Is the CCM actually running?
- Is your Node correctly labelled with the topology information?
- Is your Node ready?
  - Has your CNI initialised the Node? Is the CCM blocking it for some reason?
- Is your Node registered with the load balancer(s)?
  - Do you have a Service of type LoadBalancer?
  - Is it set with externalTrafficPolicy Local or Cluster? (Yes, this matters!)
  - Does the node require load balancer membership for outbound traffic?

Example Issue on Azure

- Azure instances can become wedged when they are unready and need outbound connections to become ready
- github.com/kubernetes-sigs/cloud-provider-azure/issues/3500
- This demonstrates how specific implementations of the CCM can differ on the various cloud providers, and the importance of investigating your provider's known issues.

Each provider is unique

Go read the issues!!!

Building Your Own CCM

Cloud Provider Framework

SIG Cloud Provider maintains a common library to simplify the implementation of the cloud provider interfaces and mechanisms. This library should be the starting point for any planning towards creating a new cloud controller manager.

github.com/kubernetes/cloud-provider

Interface Contract

Defined in cloud.go, the cloud provider interface is relatively brief and documented thoroughly in the code. As of Kubernetes 1.27 it contains 9 functions.

```
Initialize(clientBuilder ControllerClientBuilder, stop <-chan struct{})
LoadBalancer() (LoadBalancer, bool)
Instances() (Instances, bool)
InstancesV2() (InstancesV2, bool)
Zones() (Zones, bool)
Clusters() (Clusters, bool)
Routes() (Routes, bool)
ProviderName() string
HasClusterID() bool
```

Optional Interfaces

Many of the cloud provider interface functions return a boolean value in addition to the retrieved information. This allows for optional implementation of interfaces based on the capabilities of the cloud provider.

For example, not all providers have capability for load balancing Services. Returning **false** from the **LoadBalancer** interface function will let Kubernetes know to warn users when they attempt to create LoadBalancer type **Service** objects.

You might be wondering “does my CCM support this interface?”. The best way to determine which interfaces are supported is to check the logs for the CCM. For example, when the **LoadBalancer** interface is not support you will see this:

“the cloud provider does not support external load balancers”

Instances or InstancesV2?

There are two interface functions for returning information about hosts running in the cloud provider: **Instances**, and **InstancesV2**.

Instances is the original interface used by the in-tree cloud controllers. It exists in the cloud provider interface as a compatibility with the in-tree cloud controllers.

InstancesV2 is an implementation of the **Instances** interface that is designed specifically for external cloud controllers. It is behaviorally identical to **Instances** but is optimized to significantly reduce API calls to the cloud provider. Implementation of this interface will disable calls to the **Zones** interface.

Deprecated Zones Interface

Zones is for accessing information about instance zone and region information. It is designed for use with the **Instances** interface function.

The **Zones** interface function is deprecated in favor of retrieving zone and region information directly through the **InstancesV2** interface. **Zones** will not be called if the **InstancesV2** interface is available.

There is currently no time table for the removal of **Zones** from the cloud provider interface.

Use the sample CCM

The cloud-provider library contains a sample cloud controller manager contained in the “sample” directory.

This sample demonstrates a basic main function for creating a cloud controller manager, a function to start the CCM controllers (Node, Service, and Route), and a function to initialize the provider.

github.com/kubernetes/cloud-provider

(in the “sample” directory)

Learn From Other Provider's Examples

The Kubernetes and Kubernetes-sigs organizations on GitHub contains several cloud providers that are actively maintained and utilized in production environments. These repositories are also a wealth of information for learning about design patterns and interface usages.

Searching "cloud-provider" in the Kubernetes and Kubernetes-sigs organization repositories will produce a good starter list for review.

- One note of caution, the project named "cloud-provider-sample" is an alpha that was started several years ago and is not current. Looking at the concrete providers (e.g. AWS, Azure, GCP, OpenStack, etc) will give a better experience.
- There is no difference between providers in Kubernetes organization and those in Kubernetes-sigs, the separation is merely due to organic growth. New provider should all be in the "-sigs" organization

Getting Involved

Join the SIG

The Cloud Provider Special Interest Group ensures that the Kubernetes ecosystem is evolving in a way that is neutral to all (public and private) cloud providers. It is responsible for establishing standards and requirements that must be met by all providers to ensure optimal integration with Kubernetes.

Regular SIG meetings occur on alternating **Wednesdays at 18:00 CET**

For more detailed contact information, meeting notes, and recordings, please visit the “sig-cloud-provider” directory in

github.com/kubernetes/community

Get Involved: Active Development

There are a few efforts in progress that cut across cloud providers and are of general interest to the entire Kubernetes community. These are excellent opportunities to find collaborators and get involved with active development.

- Webhook hosting capability KEP-2699
  - Enables providers to replace the persistent volume labeller
  - The webhook framework has been added in k/k#108838
  - An AWS reference exists; other providers will need updating
- Make end-to-end testing less cloud specific
  - Create a better E2E testing experience for all cloud providers
  - This work is still in the early enhancement phase

Interact with the SIG

Bedankt!

Join the mailing list group

- groups.google.com/g/kubernetes-sig-cloud-provider

Attend a meeting

- Regular SIG meeting - Biweekly on Wednesdays at 18:00 CET
- Cloud Provider Extraction meeting - Biweekly on Thursdays at 18:30 CET, opposite weeks from the SIG meeting
  - This meeting is focused on the extraction of in-tree code and related issues.
- More details @ www.kubernetes.dev/resources/calendar

Report an issue or open a merge request

- github.com/kubernetes/cloud-provider