



KubeCon



CloudNativeCon

S OPEN SOURCE SUMMIT

China 2019

SIG Cloud Provider Introduction: Charter, current work, roadmap

Chris Hoge, OpenStack Foundation
Walter Fender Google
Steven Wong, VMware

Abstract

Hidden slide during presentation – included for those finding deck online later

SIG Cloud Provider is focused on ensuring a consistent and high-quality user experience across providers and acts as a central group for developing the Kubernetes project in a way that ensures all providers share common privileges and responsibilities.

New vendors providing support for Kubernetes should feel equally empowered to do so as any of today's existing cloud providers.

We will go over historical context, status, and direction of efforts currently underway including the transition of individual provider SIGs to sub-projects and ongoing work in extracting provider code from the main Kubernetes repository.

Presenters

Chris Hoge



Portland, Oregon

Co-chair, Kubernetes SIG Cloud
Provider, OpenStack SIG

Senior Strategic Program
Manager, OpenStack Foundation

PTL OpenStack Loci (lightweight
container images for OpenStack)

GitHub: @hogepodge

Steven Wong



Los Angeles

Co-chair, VMware SIG

Open Source Community
Relations Engineer, VMware

Active in Kubernetes storage and
IoT+Edge communities

GitHub: @cantbewong

Walter Fender



Graduated from U.C. Berkeley.

Working at Google and on
Kubernetes API Machinery for two
years

GitHub: @cheftako

Agenda

Historical context of cloud provider

Efforts currently underway

- Standardization efforts
- Testing requirements

Organization changes

- Transition of individual provider SIGs to sub-projects
- User groups

SIG Cloud Provider

Mission

Ensures that the Kubernetes ecosystem is evolving in a way that is neutral to all public and private cloud providers.

Responsible for establishing standards and requirements that must be met by all providers to ensure optimal integration with Kubernetes.

Simplify, develop, and maintain cloud provider integrations as extensions, or add-ons, to Kubernetes clusters

Important KEPs:

- Cloud Controller Manager <https://git.io/fjBk2>
- TestGrid conformance <https://git.io/fjBkV>
- Removing In-Tree Cloud Provider Code <https://git.io/fjBkr>

Kubernetes Cloud Providers – origin story

in tree cloud providers – included and built right into the fabric of Kubernetes

As of Kubernetes 1.14, there are several in-tree cloud providers. When you download Kubernetes, you run these by default through direct configuration.

- 
- AWS
 - Azure
 - Cloudstack
 - GCE
 - OpenStack
 - OVirt
 - Photon
 - vSphere

Photo by [JJ Ying](#) on [Unsplash](#)

Why in-tree is a problem

Or, where is my cloud provider?

- Kubernetes should be an orchestration kernel, with drivers maintained independently by domain experts.
- For any particular deployment, there is a large body of code that is entirely irrelevant for that particular deployment scenario.
- Inclusion can imply endorsement or support for a select set of providers.
- Critical updates are tightly coupled to the Kubernetes release cycle.
- Development work has stopped on some providers, leaving code untested and unmaintained.

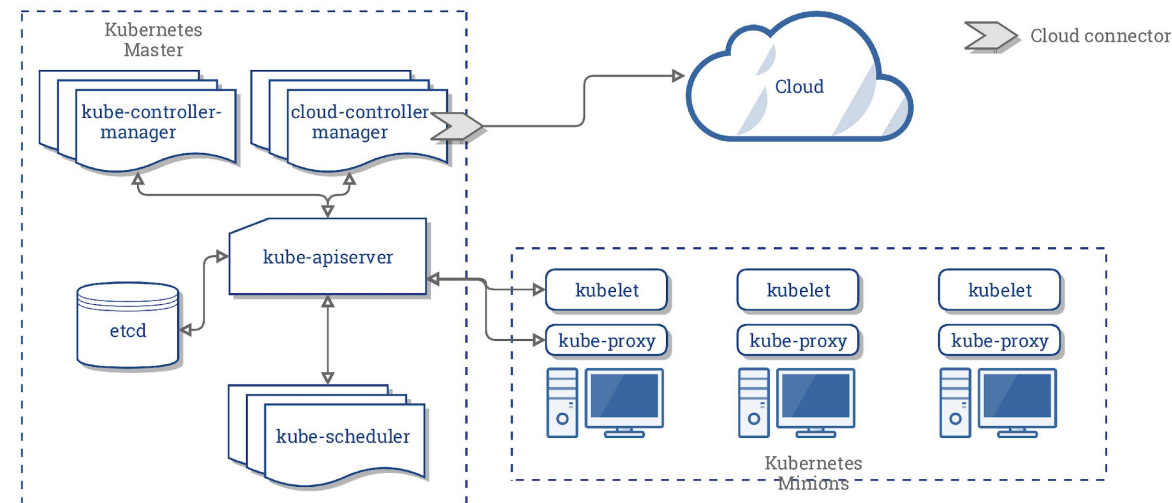


Photo by [Andrey Grinkevich](#) on [Unsplash](#)

How do we fix cloud providers for everyone?

To support independent work

- a Cloud Controller Manager interface standard was developed
- allows for external (out of tree) providers



Work is underway to make all cloud providers out of tree.

- For supported in-tree providers, the provider code has been officially deprecated and their dependencies are being moved to staging.
- With in-tree provider moved to staging after the upcoming release, they have a path to be removed in the following release. By the end of 2019, there will be no in-tree provider code.
- Meanwhile, SIG-Cloud-Provider is working on a migration path to transition running clouds from in-tree to out-of-tree providers.
- Unmaintained provider code will be removed completely.

Introduction to the Cloud Controller Manager

The Cloud Controller Manager (CCM) replaces the Kube Controller Manager (KCM), and is daemon that embeds the following cloud-specific control loops:

- Node Controller
- Route Controller
- Service Controller

Where did the (storage) volume controller go?

Storage plugins needs to go out of tree too – same reasoning as for cloud providers

Because of complexity of volume management, the CCM developers decided to not move volume control to the CCM

Instead, Kubernetes added support for external drivers implementing the cross-orchestrator Container Storage Interface (CSI) ([KEP](#))



Your own Cloud Controller Manager

In three easy steps

Go code

Create a go package that satisfies the cloud provider interface.

Copy and Import these

Create a copy of the Cloud Controller Manager main.go and import your package, making sure there is an init block available.

Publish

Building, testing, packaging, maintaining...

I want to host Kubernetes on a new platform

How do I implement a Cloud Provider?

At a high level, to build an out-of-tree controller manager, you need to implement the Cloud Provider interface.

- <https://github.com/kubernetes/cloud-provider/blob/master/cloud.go>

Interfaces to implement (all are optional):

- Load Balancer: cloud-specific ingress controller.
- Instances: cloud-specific information about nodes in your cluster.
- Zones: cloud-specific information about the host availability zones.
- Clusters: cloud-specific information about running clusters.
- Routes: cloud-specific information about networking.



Photo Micky Aldridge from Finland [CC BY 2.0
(<https://creativecommons.org/licenses/by/2.0/>)]

```

Type Interface interface {
    // Initialize provides the cloud with a kubernetes client builder and may spawn goroutines
    // to perform housekeeping or run custom controllers specific to the cloud provider.
    // Any tasks started here should be cleaned up when the stop channel closes.
    Initialize(clientBuilder ControllerClientBuilder, stop <-chan struct{})

    // LoadBalancer returns a balancer interface. Also returns true if the interface is supported, false otherwise.
    LoadBalancer() (LoadBalancer, bool)

    // Instances returns an instances interface. Also returns true if the interface is supported, false otherwise.
    Instances() (Instances, bool)

    // Zones returns a zones interface. Also returns true if the interface is supported, false otherwise.
    Zones() (Zones, bool)

    // Clusters returns a clusters interface. Also returns true if the interface is supported, false otherwise.
    Clusters() (Clusters, bool)

    // Routes returns a routes interface along with whether the interface is supported.
    Routes() (Routes, bool)

    // ProviderName returns the cloud provider ID.
    ProviderName() string

    // HasClusterID returns true if a ClusterID is required and set
    HasClusterID() bool
}

```



Cloud Controller Binary

The external cloud controller runs as a separate binary that interacts with the Kubernetes API service.

It is configured with a standard set of options, which can be extended to match the requirements of your cloud.

A starting template is provided in the Kubernetes CCM directory.

<https://github.com/kubernetes/kubernetes/blob/master/cmd/cloud-controller-manager>

```

import (
    "fmt"
    "math/rand"
    "os"
    "time"

    "k8s.io/component-base/logs"
    "k8s.io/kubernetes/cmd/cloud-controller-manager/app"
    "<my_cloud_provider>"
    _ "k8s.io/kubernetes/pkg/util/prometheusclientgo" // load all the prometheus client-go plugins
    _ "k8s.io/kubernetes/pkg/version/prometheus"      // for version metric registration
)

func main() {
    rand.Seed(time.Now().UnixNano())

    command := app.NewCloudControllerManagerCommand()

    // TODO: once we switch everything over to Cobra commands, we can go back to calling
    // utilflag.InitFlags() (by removing its pflag.Parse() call). For now, we have to set the
    // normalize func and add the go flag set by hand.
    // utilflag.InitFlags()
    logs.InitLogs()
    defer logs.FlushLogs()

    if err := command.Execute(); err != nil {
        fmt.Fprintf(os.Stderr, "error: %v\n", err)
        os.Exit(1)
    }
}

```



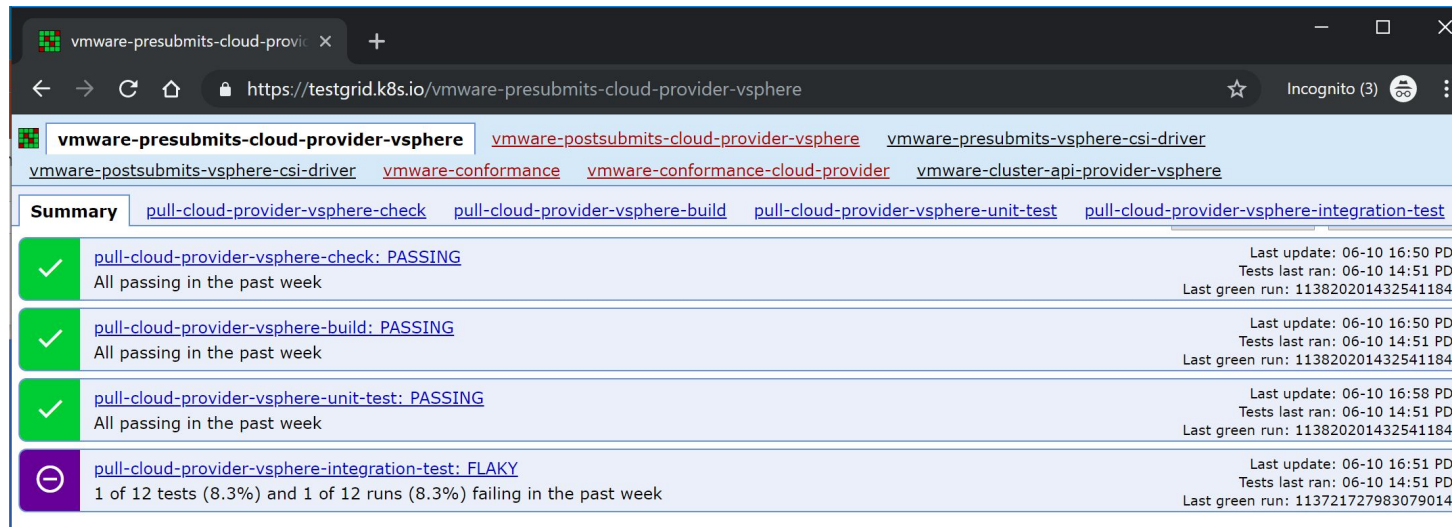
Testing

Two Types - Minimum

Your implementation needs *at least*:

- Unit tests with appropriate mocks to guarantee your implementation behaves as expected functionally.
- End to end (e2e) testing with the provider enabled on an instance of your cloud for full integration and conformance testing.

If you want to enable release gating against your cloud, both must be implemented and e2e must be reporting to test-grid.



The screenshot shows a web browser window displaying the TestGrid dashboard for the project `vmware-presubmits-cloud-provider-vsphere`. The dashboard lists several test suites, all of which are passing except for one integration test.

Test Suite	Status	Last update	Tests last ran	Last green run
pull-cloud-provider-vsphere-check	PASSING	06-10 16:50 PDT	06-10 14:51 PDT	1138202014325411840
pull-cloud-provider-vsphere-build	PASSING	06-10 16:50 PDT	06-10 14:51 PDT	1138202014325411841
pull-cloud-provider-vsphere-unit-test	PASSING	06-10 16:58 PDT	06-10 14:51 PDT	1138202014325411842
pull-cloud-provider-vsphere-integration-test	FLAKY	06-10 16:51 PDT	06-10 14:51 PDT	1137217279830790145

The integration test is marked as 'FLAKY' with a purple icon and indicates that 1 of 12 tests (8.3%) and 1 of 12 runs (8.3%) failed in the past week.

Running a Cloud Provider

(the application)

Arguments to kube-api-server:

- `--cloud-provider=external`

Arguments to start your CCM binary

- `--cloud-provider=<your cloud provider name>`
- `--cloud-config=<path to your cloud configuration>`

The similarity in options comes from the scaffolding code. This comes at the cost of a lot of unneeded options also being pulled over to the provider.

Running a Cloud Provider

In production

When running in production, use a daemonset!

Your cluster behavior will change in a few ways. Notably:

- “kubelets specifying `--cloud-provider=external` will add a taint `node.cloudprovider.kubernetes.io/uninitialized` with an effect `NoSchedule` during initialization.”
- “Cloud information about nodes in the cluster will no longer be retrieved using local metadata, but instead all API calls to retrieve node information will go through cloud controller manager. ...for larger clusters you may want to consider if cloud controller manager will hit rate limits since it is now responsible for almost all API calls to your cloud from within the cluster.”

```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cloud-controller-manager
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:cloud-controller-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: cloud-controller-manager
  namespace: kube-system
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    k8s-app: cloud-controller-manager
  name: cloud-controller-manager
  namespace: kube-system
spec:
  selector:
    matchLabels:
      k8s-app: cloud-controller-manager
  template:
    metadata:
      labels:
        k8s-app: cloud-controller-manager

```

```

spec:
  serviceAccountName: cloud-controller-manager
  containers:
    - name: cloud-controller-manager
      # for in-tree providers we use
      k8s.gcr.io/cloud-controller-manager
      # this can be replaced with any other image for
      out-of-tree providers
      image: k8s.gcr.io/cloud-controller-manager:v1.8.0
      command:
        - /usr/local/bin/cloud-controller-manager
        - --cloud-provider=<YOUR_CLOUD_PROVIDER>
        # Add your own cloud provider here!
        - --leader-elect=true
        - --use-service-account-credentials
        # these flags will vary for every cloud provider
        - --allocate-node-cidrs=true
        - --configure-cloud-routes=true
        - --cluster-cidr=172.17.0.0/16
      tolerations:
        # this is required so CCM can bootstrap itself
        - key: node.cloudprovider.kubernetes.io/uninitialized
          value: "true"
          effect: NoSchedule
        # this is to have the daemonset runnable on master nodes
        # the taint may vary depending on your cluster setup
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
        # this is to restrict CCM to only run on master nodes
        # the node selector may vary depending on your cluster
        setup
        nodeSelector:
          node-role.kubernetes.io/master: ""

```

Implementation Details

Consider:

What library will you use to interact with your cloud?

How do you want to handle authentication and authorization?

- It's likely that your cloud controller isn't the only thing interacting with your cloud.
- For example, you may have a Cluster API provider or storage provider in the works.
- Consolidate your efforts, otherwise you'll wind up with fragmented and inconsistent auth configuration methods across your projects.

Thank You

Q&A

Contacts

This deck: link tbd

Join the Cloud Provider SIG

- Slack channel: <https://kubernetes.slack.com/messages/sig-cloud-provider>
- List: <https://groups.google.com/forum/#!forum/kubernetes-sig-cloud-provider>
- Zoom meetings (join mailing list group for schedule)

SIG chairs

Chris Hoge

@hogepodge

Jago Macleod

@jagosan

Andrew Sy Kim

@andrewsykim