The Manufactory of Kubernetes: The kube-controller-manager

Learn about the kube-controller-manager.

We'll cover the following

- The kube-controller-manager
  - What does the kube-controller-manager do?
  - What's in the kube-controller-manager?
- Summary

The `kube-controller-manager`

In this course, we won't make any customizations of the `kube-controller-manager`. So, this lesson will only give us a basic understanding of what the `kube-controller-manager` does. At the same time, we could benefit a lot from the design of the `kube-controller-manager`. This is where the operator pattern originates from.

What does the `kube-controller-manager` do?

As we discussed previously, the control plane is responsible for driving the actual state of the system toward the desired state. However, the `kube-apiserver` focuses on storing resources and providing RESTful services for all clients. The `kube-scheduler` assigns unassigned pods with the best-matched nodes. In the control plane, we need a component that can do the convergences. This is done by the `kube-controller-manager`.

The `kube-controller-manager` performs cluster-level functions. Primarily, it manages a group of controllers that's responsible for reconciling the state of objects—such as `Deployment` and `Service`—and performing routine tasks. For instance, a replication controller ensures that the desired number of `Pod` objects are running healthily in the cluster by scaling up or down when the desired number isn't met.

What's in the `kube-controller-manager`?

Below is a code snippet of the `kube-controller-manager`:

```
1  // Codes from <https://github.com/kubernetes/kubernetes/blob/master/cmd/kube-controller-manager/app/cont
2  // NewControllerInitializers is a public map of named controller groups (you can start more than one in
3  // paired to their InitFunc.  This allows for structured downstream composition and subdivision.
4  func NewControllerInitializers(loopMode ControllerLoopMode) map[string]InitFunc {
5      controllers := map[string]InitFunc{}
6      controllers["endpoint"] = startEndpointController
7      controllers["endpointslice"] = startEndpointSliceController
8      controllers["endpointslicemirroring"] = startEndpointSliceMirroringController
```

```
    controllers["replicationcontroller"] = startReplicationController
    controllers["podgc"] = startPodGCController
    controllers["resourcequota"] = startResourceQuotaController
    controllers["namespace"] = startNamespaceController
    controllers["serviceaccount"] = startServiceAccountController
    controllers["garbagecollector"] = startGarbageCollectorController
    controllers["daemonset"] = startDaemonSetController
    controllers["job"] = startJobController
    controllers["deployment"] = startDeploymentController
    controllers["replicaset"] = startReplicaSetController
    controllers["horizontalpodautoscaling"] = startHPAController
    controllers["disruption"] = startDisruptionController
    controllers["statefulset"] = startStatefulSetController
    controllers["cronjob"] = startCronJobController
    controllers["csrsigning"] = startCSRSigningController
    controllers["csrapproving"] = startCSRApprovingController
    controllers["csrcleaner"] = startCSRCleanerController
    controllers["ttl"] = startTTLController
    controllers["bootstrapsigner"] = startBootstrapSignerController
    controllers["tokencleaner"] = startTokenCleanerController
    controllers["nodeipam"] = startNodeIpamController
    controllers["nodelifecycle"] = startNodeLifecycleController
    if loopMode == IncludeCloudLoops {
```

Controller initializers in the kube-controller-manager

More than 30 controllers are running inside the `kube-controller-manager`. It's evident what each of these controllers does from its name. We can enable or disable some controllers as needed. This can be done with the flag `--controllers` of the `kube-controller-manager`.

Let's take a closer look at one controller, the `ServiceController`:

```
// Codes from <https://github.com/kubernetes/kubernetes/blob/b74d023e70d6064c7f3f77031e7d26ec38497fc9/cr
func startServiceController(ctx context.Context, controllerContext ControllerContext) (controller.Interf
   serviceController, err := servicecontroller.New(
      controllerContext.Cloud,
      controllerContext.ClientBuilder.ClientOrDie("service-controller"),
      controllerContext.InformerFactory.Core().V1().Services(),
      controllerContext.InformerFactory.Core().V1().Nodes(),
      controllerContext.ComponentConfig.KubeCloudShared.ClusterName,
      utilfeature.DefaultFeatureGate,
   )
   if err != nil {
      // This error shouldn't fail. It lives like this as a legacy.
      klog.Errorf("Failed to start service controller: %v", err)
      return nil, false, nil
   }
   go serviceController.Run(ctx, int(controllerContext.ComponentConfig.ServiceController.ConcurrentServi
   return nil, true, nil
}
```

The ServiceController in the kube-controller-manager

This controller runs as a standalone goroutine to perform actual work for deployed `Services`. From here, we see that the `kube-controller-manager` starts multiple, distinct goroutines in the background.

This is to watch the `kube-apiserver` for changes (including creating, updating, and deleting) to separate resources and perform operations for each change.

Summary

Every controller uses the `WATCH` mechanism to get every change from the informer. Controllers also run a reconciliation loop that keeps reconciling the actual state with the desired state. Then, it will report back the newest status to the `kube-apiserver`. In general, controllers never talk to each other. Each controller watches the `kube-apiserver` to get changes purely on self-interested and responsible resources. This is a very illustrative example on how to design and write a good controller and/or operator. The golden rule for this is "Each one does things in their own way."

Back

The Conductor of Kubernetes: The kube-scheduler

Next

Kubernetes Worker Nodes