COMPONENTS OF THE CONTROL PLANE

The Control Plane is what controls and makes the whole cluster function. To refresh

your memory, the components that make up the Control Plane are

 **The etcd distributed persistent storage**

 **The API server**

 **The Scheduler**

 **The Controller Manager**

These components store and manage the state of the cluster, but they aren’t what runs

the application containers.

COMPONENTS RUNNING ON THE WORKER NODES

The task of running your containers is up to the components running on each

worker node:

 **The Kubelet**

 **The Kubernetes Service Proxy (kube-proxy)**

 **The Container Runtime (Docker, rkt, or others)**

ADD-ON COMPONENTS

Beside the Control Plane components and the components running on the nodes, a

few add-on components are required for the cluster to provide everything discussed

so far. This includes

 The Kubernetes DNS server

 The Dashboard

 An Ingress controller

 Heapster, which we’ll talk about in chapter 14

 The Container Network Interface network plugin

HOW THESE COMPONENTS COMMUNICATE

Kubernetes system components communicate only with the API server. They don’t

talk to each other directly. The API server is the only component that communicates

with etcd. None of the other components communicate with etcd directly, but instead

modify the cluster state by talking to the API server.

Although the components on the worker nodes all need to run on the same node,

the components of the Control Plane can easily be split across multiple servers.

While multiple instances of etcd and API server can be active at the

same time and do perform their jobs in parallel, only a single instance of the Scheduler

and the Controller Manager may be active at a given time

HOW COMPONENTS ARE RUN :

The Control Plane components, as well as kube-proxy, can either be deployed on the

system directly or they can run as pods (as shown in listing 11.1). You may be surprised

to hear this, but it will all make sense later when we talk about the Kubelet.

The Kubelet is the only component that always runs as a regular system component,

and it’s the Kubelet that then runs all the other components as pods. To run the

Control Plane components as pods, the Kubelet is also deployed on the master.

***How Kubernetes uses etcd***

All the objects you’ve created throughout this book—Pods, ReplicationControllers,

Services, Secrets, and so on—need to be stored somewhere in a persistent manner so

their manifests survive API server restarts and failures. For this, Kubernetes uses etcd,ods

which is a fast, distributed, and consistent key-value store. Because it’s distributed,

you can run more than one etcd instance to provide both high availability and better

performance.

It’s worth emphasizing that etcd is the *only* place Kubernetes stores cluster state and metadata.

Kubernetes stores all its data in etcd under /registry.

Usually, for large clusters, an etcd cluster of five or seven nodes is sufficient. It can handle a two- or a three-node failure, respectively, which suffices in almost all situations.

***What the API server does***

The Kubernetes API server is the central component used by all other components and by clients, such as kubectl.

It provides a CRUD (Create, Read, Update, Delete) interface for querying and modifying the cluster state over a RESTful API. It stores that state in etcd.

The API server doesn’t do anything else except what we’ve discussed. For example, it doesn’t create pods when you create a ReplicaSet resource and it doesn’t manage the endpoints of a service. That’s what controllers in the Controller Manager do.

But the API server doesn’t even tell these controllers what to do. All it does is enable those controllers and other components to observe changes to deployed resources. A Control Plane component can request to be notified when a resource is created, modified, or deleted. This enables the component to perform whatever task it needs in response to a change of the cluster metadata.

Clients watch for changes by opening an HTTP connection to the API server.

Through this connection, the client will then receive a stream of modifications to the watched objects. Every time an object is updated, the server sends the new version of the object to all connected clients watching the object.

***Understanding the Scheduler***

All the Scheduler does is update the pod definition through the API server. The API server then notifies the Kubelet (again, through the watch mechanism described previously) that the pod has been scheduled. As soon as the Kubelet on the target node sees the pod has been scheduled to its node, it creates and runs the

pod’s containers.

Pods belonging to the same Service or ReplicaSet are spread across multiple nodes by default.

Instead of running a single Scheduler in the cluster, you can run multiple Schedulers. Then, for each pod, you specify the Scheduler that should schedule this particular pod by setting the schedulerName property in the pod spec.

***The controllers running in the Controller Manager***

the API server doesn’t do anything except store resources in etcd and notify clients about the change. The Scheduler only assigns a node to the pod, so you need other active components to make sure the actual state of the system converges toward the desired state, as specified in the resources deployed through the

API server. **This work is done by controllers running inside the Controller Manager.**

The list of these controllers includes the :

 Replication Manager (a controller for ReplicationController resources)

 ReplicaSet, DaemonSet, and Job controllers

 Deployment controller

 StatefulSet controller

 Node controller

 Service controller

 Endpoints controller

 Namespace controller

 PersistentVolume controller

 Others

What each of these controllers does should be evident from its name. From the list, you can tell there’s a controller for almost every resource you can create. Resources are descriptions of what should be running in the cluster, whereas the controllers are the active Kubernetes components that perform actual work as a result of the deployed resources.

Controllers do many different things, but they all watch the API server for changes to resources (Deployments, Services, and so on) and perform operations for each change.

Most of the time, these operations include creating other resources or updating the watched resources themselves (to update the object’s status, for example).

In general, controllers run a reconciliation loop, which reconciles the actual state with the desired state (specified in the resource’s spec section) and writes the new actual state to the resource’s status section. Controllers use the watch mechanism to be notified of changes, but because using watches doesn’t guarantee the controller won’t miss an event, they also perform a re-list operation periodically to make sure they haven’t missed anything.

Controllers never talk to each other directly. They don’t even know any other controllers exist. Each controller connects to the API server and, through the watch mechanism described in section 11.1.3, asks to be notified when a change occurs in the list of resources of any type the controller is responsible for.

Again, all these controllers operate on the API objects through the API server. They don’t communicate with the Kubelets directly or issue any kind of instructions to them. In fact, they don’t even know Kubelets exist.

After a controller updates a resource in the API server, the Kubelets and Kubernetes Service Proxies,

also oblivious of the controllers’ existence, perform their work, such as spinning up a pod’s containers and attaching network storage to them, or in the case of services, setting up the actual load balancing across pods.

***The Kubelet***

In a nutshell, the Kubelet is the component responsible for everything running on a worker node. Its initial job is to register the node it’s running on by creating a Node resource in the API server. Then it needs to continuously monitor the API server for Pods that have been scheduled to the node, and start the pod’s containers. It does this by telling the configured container runtime (which is Docker, CoreOS’ rkt, or something else) to run a container from a specific container image. The Kubelet then constantly monitors running containers and reports their status, events, and resource consumption to the API server. The Kubelet is also the component that runs the container liveness probes, restarting containers when the probes fail. Lastly, it terminates containers when their Pod is deleted from the API server and notifies the server that the pod has terminated.