Kubernetes Docs:

Grant updated the document and added ‘References/Additional Resources’

<https://gitscm.cisco.com/projects/EPS-KUBE/repos/cae-route-controller/browse/README_BASICS.md>

K8s Controller Deep Dive - <https://engineering.bitnami.com/articles/a-deep-dive-into-kubernetes-controllers.html>

Sample Controller - <https://github.com/kubernetes/sample-controller>

Understanding K8s Cache Package - <https://lairdnelson.wordpress.com/2018/01/07/understanding-kubernetes-tools-cache-package-part-0/>

first links are good to get started

Kubernetes:

* ETCD key store. ETCD is a distributed reliable key-value store used by kubernetes to store all data used to manage the cluster. Think of it this way, when you have multiple nodes and multiple masters in your cluster, etcd stores all that information on all the nodes in the cluster in a distributed manner. ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters.
* Scheduler:

The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to Nodes.

* Controller

The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases.

* Kubelet

kubelet is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.

Note:\*\*\* The master server has the kube-apiserverand that is what makes it a master.

The worker node (or minion) as it is also known, is were the containers are hosted.

the worker nodes have the kubelet agent that is responsible for interacting with the master to provide health information of the worker node and carry out actions requested by the master on the worker nodes.

All the information gathered are stored in a key-value store on the Master. The key value store is based on the popular etcd framework as we just discussed. The master also has the controller manager and the scheduler.

Kubernetes requires a special network between the master and worker nodes which is called as a POD network.

DEFINITION:

Kubernetes is an open-source platform for automating deployments, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

POD:

A set of one or more containers that are deployed onto a Node and share a unique IP and Volume (persistent Storage). Pods also define the security and runtime policy for each container

The shared context of a Pod is a set of Linux namespaces, cgroups, and potentially other facets of isolation - the same things that isolate a Docker container. Within a Pod’s context, the individual applications may have further sub-isolations applied.

*From <*[*https://kubernetes.io/docs/concepts/workloads/pods/pod/*](https://kubernetes.io/docs/concepts/workloads/pods/pod/)*>*

NOTE:

If that Pod is deleted for any reason, even if an identical replacement is created, the related thing (e.g. volume) is also destroyed and created anew.

*From <*[*https://kubernetes.io/docs/concepts/workloads/pods/pod/*](https://kubernetes.io/docs/concepts/workloads/pods/pod/)*>*

Machine generated alternative text:
Kubernetes Pod 
Group of one or more containers thatkare always co-located, 
co-scheduled, and run in a shared context 
Containers in the same pod have the same hostname 
Each pod is isolated by 
o Process ID (PID) namespace 
o Network namespace 
o Interprocess Communication (l PC) namespace 
o Unix Time Sharing (UTS) namespace 
Alternative to a VM with multiple processes 

Uses of pods

Pods can be used to host vertically integrated application stacks (e.g. LAMP), but their primary motivation is to support co-located, co-managed helper programs, such as:

* + content management systems, file and data loaders, local cache managers, etc.
  + log and checkpoint backup, compression, rotation, snapshotting, etc.
  + data change watchers, log tailers, logging and monitoring adapters, event publishers, etc.
  + proxies, bridges, and adapters
  + controllers, managers, configurators, and updaters

*From <*[*https://kubernetes.io/docs/concepts/workloads/pods/pod/*](https://kubernetes.io/docs/concepts/workloads/pods/pod/)*>*

Controllers provide self-healing with a cluster scope, as well as replication and rollout management. Controllers like [StatefulSet](https://kubernetes.io/docs/concepts/workloads/controllers/statefulset.md) can also provide support to stateful Pods.

*From <*[*https://kubernetes.io/docs/concepts/workloads/pods/pod/*](https://kubernetes.io/docs/concepts/workloads/pods/pod/)*>*

SERVICES:

Machine generated alternative text:
Services 
An abstraction to define a logical set of Pods bound by a policy by to 
access them 
Services are exposed through internal and external endpoints 
Services can also point to non-Kubernetes endpoints through a 
Virtual-IP-Bridge 
Supports TCP and UDP 
Interfaces with kube-proxy to manipulate iptables 
Service can be exposed internal or external to the cluster 

Exposing the service:

Machine generated alternative text:
Exposing Services 
Client 
Node 
Port 
Cluster 
DB Pod 
Web 
Pod 
Web 
Pod 
Web 
Pod 

Replication Controller:

Machine generated alternative text:
Replication Controller 
Ensures that a Pod or homogeneous set of Pods are always up and 
available 
Always maintains desired number of Pods 
o If there are excess Pods, they get killed 
o New pods are launched when they fail, get deleted, or terminated 
Creating a replication controller with a count of 1 ensures that a Pod is 
always available 
Replication Controller and Pods are associated through Labels 

**Controller pattern**

In applications of robotics and automation, a control loop is a non-terminating loop that regulates the state of the system. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API server and makes changes attempting to move the current state towards the desired state.

\* Examples of controllers that ship with Kubernetes today are the replication controller, endpoints controller, namespace controller, and service accounts controller.

To reduce complexity, all controllers are packaged and shipped in a single daemon named kube-controller-manager. The simplest implementation of a controller is a loop:

**for {  
 desired := getDesiredState()  
 current := getCurrentState()  
 makeChanges(desired, current)  
}**

**Controller components**

There are **two main components of a controller: Informer/SharedInformer and Workqueue.** Informer/SharedInformer watches for changes on the current state of Kubernetes objects and sends events to Workqueue where events are then popped up by worker(s) to process.

**Informer**

The vital role of a Kubernetes controller is to watch objects for the desired state and the actual state, then send instructions to make the actual state be more like the desired state.**In order to retrieve an object's information, the controller sends a request to Kubernetes API server.**