Kubernetes Processes

There is a core set of *Kubernetes* processes that form the cluster runtime. The cluster runtime is further divded into master components, aka the control plane, and node components. There is also a set of cluster add-ons.

Master Components

kube-apiserver

The kube-apiserver exposes the *Kubernetes* API. It is where the *Kubernetes Resources* are published into.

kube-controller-manager

The kube-controller-manager runs a number of controllers that are responsible for watching for particular *Kubernetes Resources* in the API. The controllers bring the cluster into alignment with the declared state in the *Resources*.

kube-scheduler

The kube-scheduler assigns *Pods* to nodes based on a number of criteria. Such as resource requests and resource availability, workload placement policies, etc.

cloud-controller-manager

The cloud-controller-manager runs controllers that interact with a specific cloud provider. An example would be *AWS* or *Digital Ocean*.

Etcd

Etcd is a distributed key / value store. It is used to store the *Kubernetes* cluster state.

Node Components

kubelet

The kubelet runs on each node and is responsible for managing the *Pods* on that node.

kube-proxy

The kube-proxy configures the network

rules so that abstractions like the *Service Resource* work.*Services* are described in some detail further on in the document.

Kubernetes Concepts
Before describing *Kubernetes Resources* there is some important conceptual information to cover.

Labels and Selectors
Labels is one of the fundamental *Kubernetes* concepts. A label is a key / value pair that is attached to a *Kubernetes Resource*. And each *Kubernetes Resource* can have many labels. Labels serve as a means to identify a particular *Resource* or set of *Resources*.

Selectors are also key / value pairs. They are used to select *Resources* that have a particular label or set of labels. By this mechanism it is possible for one *Resource* type to select one or more *Resources* of another type. Labels can also be used with kubectl to query just particular *Resources*.
The following diagram depicts this selection mechanism with a *Service* selecting a set of *Pods*.

More information about labels and selectors can be found here.
Kubernetes Resources
Resources are high level abstractions that contain the declarative state for an infrastructure component.*Resources* are published into the *Kubernetes* API server and it is the responsibility of *Kubernetes*controllers to bring the cluster into agreement with the declared state.
There are many different types of *Resource* available. The full list of *Resources* can be found in the API documentation. The following sections describe just a few of the most important *Resources*.
Note
This is really just a high level introduction. More detailed information about some of these *Resources* will be provided as the course unfolds.
Workloads
Pods
As described previously, a *Pod* is the fundamental unit of execution in *Kubernetes*. A *Pod* has one or more containers. The following diagram depicts a *Pod* and the three types of containers that it can have.

Container Type	Description
Application	This is the core container for an application. You must have one of

these with the most common pattern being that a *Pod* only has the application container.

Sidecar A *Pod* can also have other containers that perform some useful work in support of the application container. A container that ships application logs is a good example. There can be any number of these within a *Pod*.

Init Sometimes it is necessary to do some initialisation before starting the application container. For instance, to create a database. There can be any number of these *Init*containers within a *Pod* and they are run one at a time in sequential order. See herefor more information.

As with other *Resource* types, a *Pod* can have labels. And these labels are used by other *Resources* (see next section) to enable management of the *Pods*.

Deployments

A *Deployment* allows you to manage the lifecycle of *Pods* and an associated *Resource* called a *ReplicaSet*.

A *Deployment* contains a specification for a *Pod* and also additional information, such as the number of *Pods* to run.

The *ReplicaSet* is created when a *Deployment* is created or updated and it is actually the *ReplicaSet* that is used as the definition for creating the *Pods*.

Each time the *Deployment* is updated a new *ReplicaSet* is created. This makes it possible to roll back a *Deployment* by using the previous *ReplicaSet*.

The following diagram depicts this.

More informations about *Deployments* can be found here.

DaemonSet

A *Daemonset* ensures that a *Pod* is run on all, or a subset, of the available *Kubernetes Nodes*.

A use case for this might be where you want to have a metric collector run on all the *Nodes* so that kernel performance statistics can be gathered.

The following diagram depicts this.

More information about *DaemonSets* can be found here.

StatefulSet

A *StatefulSet* manages sets of *Pods* in a way that provides a guaranteed *Pod* name and a specific storage volume for each. The order in which the *Pods* are created and destroyed is also guaranteed.

A use case for this is where a distributed database cluster relies upon its members having a specific name with associated data and that the cluster must be created with an initial bootstrap member.*ElasticSearch* is a good example of this.

The following diagram depicts this.

Job

A *Job* creates one of more *Pods* and ensures that at least a certain amount of them run to completion.
A use case for this is a batch data load.
The following diagram depicts this.

CronJob
A *CronJob* runs *Jobs* on a schedule. The format for the schedule is the Cron format.
A use case for this is a scheduled backup.
The following diagram depicts this.

More information about *CronJobs* can be found here.
Discovery and Load Balancing
Service
A *Service* is used to group together one or more *Pods* using a selector.
The *Service* then provides a *ClusterIP* and a DNS name for the *Pods*.
The *Kube Proxy* process uses the information contained within a *Service* to configure appropriate iptables rules on each of the nodes so that clients of the *Service* are able to be routed to an appropriate *Pod*.
There is a lot more detail to *Services* which can be found in the link below the diagram.

Ingress
An *Ingress* is used to configure an *Ingress Controller* so that a *Service* can be made accessible externally of the cluster.
An *Ingress* is a layer 7 HTTP construct and can configure both HTTP and HTTPS access.
An *Ingress Controller* is a reverse proxy that watches for *Ingress Rosources* in the *Kubernetes* API server and configures itself with appropriate rules as per the *Ingress*.

Config and Storage
ConfigMap
A *ConfigMap* allows configuration files to be stored independantly of a container image. The files stored within the *ConfigMap* can then be mounted into a container within a *Pod* at runtime.
The following diagram depicts this.

Secret
A *Secret* allows secret files and values to be stored independant of a container image. Much like a *ConfigMap* but also with the ability to store the value of a key in an environment variable.

The following diagram depicts this.

More information about *Secrets* can be found here.

PersistentVolume and PersistentVolumeClaim

A *PersistentVolume* represents some form of distributed storage.
A *PersistenVolumeClaim* allows a *PersistentVolume* to be claimed for a *Pod*.
The following diagram depicts this.

NameSpace

This is where we put it all together. A *NameSpace* can be thought of as like an environment. Collections of *Resources* can be deployed into the *NameSpace* to form a grouping of related components.
Examples of *NameSpace* use might be to create a DEV environment, or to share a set of services that are used by other *NameSpaces*, i.e. an *Ingress Controller*.
The following diagram depicts this.