# Kubernetes Resources Part 1

## Summary

Up to this stage, we have explored how to package an application using Docker and how to bootstrap a cluster using k3s. In the next phase, we need to deploy the packaged application to a Kubernetes cluster.

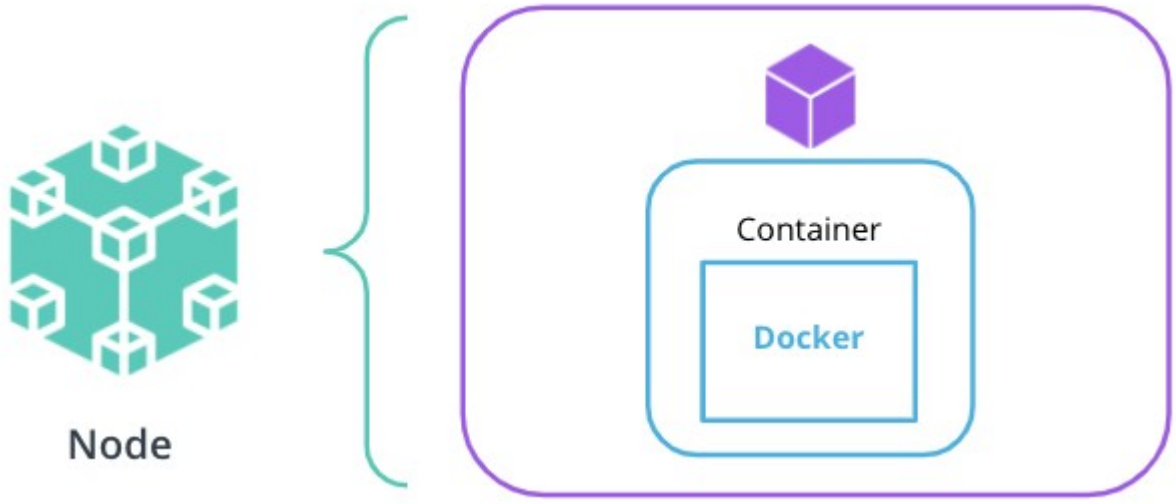
Kubernetes provides a rich collection of resources that are used to deploy, configure, and manage an application. Some of the widely used resources are:

- Pods - the atomic element within a cluster to manage an application
- Deployments & ReplicaSets - oversees a set of pods for the same application
- Services & Ingress - ensures connectivity and reachability to pods
- Configmaps & Secrets - pass configuration to pods
- Namespaces - provides a logical separation between multiple applications and their resources
- Custom Resource Definition (CRD) - extends Kubernetes API to support custom resources

## Application Deployment

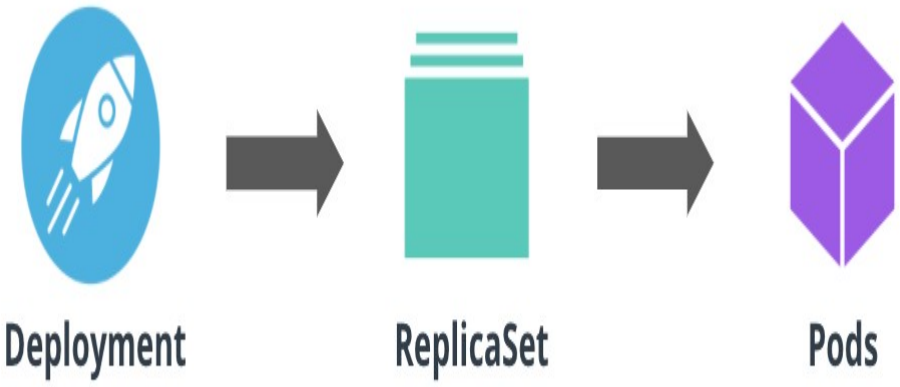
## Summary

A **pod** is the anatomic element within a cluster that provides the execution environment for an application. Pods are the smallest manageable units in a Kubernetes cluster. Every pod has a container within it, that executes an application from a Docker image (or any OCI-compliant image). There are use cases where 2-3 containers run within the same pod, however, it is highly recommended to keep the 1:1 ratio between your pods and containers.
All the pods are placed on the cluster nodes. A note can host multiple pods for different applications.




Pod architecture, showcasing a container running a Docker image

## Deployments and ReplicaSets




Application management using a Deployment and ReplicaSet

To deploy an application to a Kubernetes cluster, a **Deployment** resource is necessary. A Deployment contains the specifications that describe the desired state of the application. Also, the Deployment resource manages pods by using a **ReplicaSet**. A ReplicaSet resource ensures that the desired amount of replicas for an application are up and running at all times.

To create a deployment, use the `kubectl create deployment` command, with the following syntax:

```
# create a Deployment resource
# NAME - required; set the name of the deployment
# IMAGE - required;  specify the Docker image to be executed
```

```
# FLAGS - optional; provide extra configuration parameters for the resource
# COMMAND and args - optional; instruct the container to run specific commands when it starts
kubectl create deploy NAME --image=image [FLAGS] -- [COMMAND] [args]

# Some of the widely used FLAGS are:
-r, --replicas - set the number of replicas
-n, --namespace - set the namespace to run
--port - expose the container port
```

For example, to create a Deployment for the Go hello-world application, the following command can be used:

```
# create a go-helloworld Deployment in namespace `test`
kubectl create deploy go-helloworld --image=pixelpotato/go-helloworld:v1.0.0 -n test
```

It is possible to create headless pods or pods that are not managed through a ReplicaSet and Deployment. Though it is not recommended to create standalone pods, these are useful when creating a testing pod.

To create a headless pod, the `kubectl run` command is handy, with the following syntax:

```
# create a headless pod
# NAME - required; set the name of the pod
# IMAGE - required;  specify the Docker image to be executed
# FLAGS - optional; provide extra configuration parameters for the resource
# COMMAND and args - optional; instruct the container to run specific commands when it starts
kubectl run NAME --image=image [FLAGS] -- [COMMAND] [args...]

# Some of the widely used FLAGS are:
--restart - set the restart policy. Options [Always, OnFailure, Never]
--dry-run - dry run the command. Options [none, client, server]
-it - open an interactive shell to the container
```

For example, to create a busybox pod, the following command can be used:

```
# example: create a busybox pod, with an interactive shell and a
restart policy set to Never
kubectl run -it busybox-test --image=busybox --restart=Never
```

## Rolling Out Strategy

The Deployment resource comes with a very powerful rolling out strategy, which ensures that no downtime is encountered when a new version of the application is released. Currently, there are 2 rolling out strategies:

- RollingUpdate - updates the pods in a rolling out fashion (e.g. 1-by-1)
- Recreate - kills all existing pods before new ones are created

  For example, in this case, we upgrade a Go hello-world application from version 1.0.0 to version 2.0.0:


Rolling update of an application, between different versions

Where:

1. The Go hello-world application is running version v1.0.0 in a pod managed by a ReplicaSet
2. The version of Go hello-world application is set to v2.0.0
3. A new ReplicaSet is created that controls a new pod with the application running in version v2.0.0
4. The traffic is directed to the pod running v2.0.0 and the pod with the old configuration (v1.0.0) is removed

## Application Development Demo

## Summary

This demo showcases how an application can be deployed, configured, and managed within a Kubernetes cluster using Deployment, ReplicaSet, and pod resources.

The instructor uses the Go hello-world application in version v1.0.0 and v2.0.0. The difference between these 2 versions is the exposed port by the application. Below are the code snippets for both application versions (you can also refer to the go-hellowolrd application from the course repository):

```
# Application version: v1.0.0
# port exposed: 6111
package main

import (
    "fmt"
    "net/http"
)

func helloWorld(w http.ResponseWriter, r *http.Request){
    fmt.Fprintf(w, "Hello World")
}

func main() {
    http.HandleFunc("/", helloWorld)
```

```
    http.ListenAndServe(":6111", nil)
}
# Application version: v2.0.0
# port exposed: 6112
package main

import (
    "fmt"
    "net/http"
)

func helloWorld(w http.ResponseWriter, r *http.Request){
    fmt.Fprintf(w, "Hello World")
}

func main() {
    http.HandleFunc("/", helloWorld)
    http.ListenAndServe(":6112", nil)
}
```

## New terms

- **Pod** - smallest manageable uint within a cluster that provides the execution environment for an application
- **ReplicaSet** - a mechanism to ensure a number of pod replicas are up and running at all times
- **Deployment** - describe the desired state of the application to be deployed

## Further reading

Explore the Kubernetes resources in more detail:

- Kubernetes Pods
- Kubernetes Deployments
- Kubernetes ReplicaSets
- Kubernetes RollingOut Strategies