# WHAT IS KUBERNETES (K8S)? A KUBERNETES BASICS TUTORIAL

In this post, we're going to explain **Kubernetes**, also known as K8s. In this introduction, we'll cover:

- What Kubernetes is
- What it can't do
- The problems it solves
- K8s architectural components
- Installation
- Alternative options

This article assumes you are new to Kubernetes and want to get a solid understanding of its concepts and building blocks.

*(This article is part of our Kubernetes Guide. Use the right-hand menu to navigate.)*

## What is Kubernetes?

To begin to understand the usefulness of Kubernetes, we have to first understand two concepts: immutable infrastructure and containers.

- **Immutable infrastructure** is a practice where servers, once deployed, are never modified. If something needs to be changed, you never do so directly on the server. Instead, you'll build a new server from a base image, that have all your needed changes baked in. This way we can simply replace the old server with the new one without any additional modification.

- **Containers** offer a way to package code, runtime, system tools, system libraries, and configs altogether. This shipment is a lightweight, standalone executable. This way, your application will behave the same every time no matter where it runs (e.g, Ubuntu, Windows, etc.). Containerization is not a new concept, but it has gained immense popularity with the rise of microservices and Docker.

Armed with those concepts, we can now define Kubernetes as a container or microservice platform that orchestrates computing, networking, and storage infrastructure workloads. Because it doesn't limit the types of apps you can deploy (any language works), Kubernetes extends how we scale containerized applications so that we can enjoy all the benefits of a truly immutable infrastructure. The general rule of thumb for K8S: if your app fits in a container, Kubernetes will deploy it.

By the way, if you're wondering where the name "Kubernetes" came from, it is a Greek word, meaning helmsman or pilot. The abbreviation K8s is derived by replacing the eight letters of "ubernete" with the digit 8.

The Kubernetes Project was open-sourced by Google in 2014 after using it to run production workloads at scale for more than a decade. Kubernetes provides the ability to run dynamically scaling, containerised applications, and utilising an API for management. Kubernetes is a vendor-agnostic container management tool, minifying cloud computing costs whilst simplifying the running of resilient and scalable applications.

Kubernetes has become the standard for running containerised applications in the cloud, with the main Cloud Providers (AWS, Azure, GCE, IBM and Oracle) now offering managed Kubernetes services.

# Kubernetes basic terms and definitions

To begin understanding how to use K8S, we must understand the objects in the API. Basic K8S objects and several higher-level abstractions are known as **controllers**. These are the building block of your application lifecycle.

Basic objects include:

- **Pod.** A group of one or more containers.
- **Service.** An abstraction that defines a logical set of pods as well as the policy for accessing them.
- **Volume.** An abstraction that lets us persist data. (This is necessary because containers are ephemeral—meaning data is deleted when the container is deleted.)
- **Namespace.** A segment of the cluster dedicated to a certain purpose, for example a certain project or team of devs.

Controllers, or higher-level abstractions, include:

- **ReplicaSet (RS).** Ensures the desired amount of pod is what's running.
- **Deployment.** Offers declarative updates for pods an RS.
- **StatefulSet.** A workload API object that manages stateful applications, such as databases.
- **DaemonSet.** Ensures that all or some worker nodes run a copy of a pod. This is useful for daemon applications like Fluentd.
- **Job.** Creates one or more pods, runs a certain task(s) to completion, then deletes the pod(s).

## Micro Service

A specific part of a previously monolithic application. A traditional micro-service based architecture would have multiple services making up one, or more, end products. Micro services are typically shared between applications and makes the task of Continuous Integration and Continuous Delivery easier to manage. Explore the difference between monolithic and microservices architecture.

## Images

Typically a docker container image – an executable image containing everything you need to run your application; application code, libraries, a runtime, environment variables and configuration files. At runtime, a container image becomes a container which runs everything that is packaged into that image.

## Pods

A single or group of containers that share storage and network with a Kubernetes configuration, telling those containers how to behave. Pods share IP and port address space and can communicate with each other over localhost networking. Each pod is assigned an IP address on which it can be accessed by other pods within a cluster. Applications within a pod have access to shared volumes – helpful for when you need data to persist beyond the lifetime of a pod. Learn more about Kubernetes Pods.

## Namespaces

Namespaces are a way to create multiple virtual Kubernetes clusters within a single cluster. Namespaces are normally used for wide scale deployments where there are many users, teams and projects.

## Replica Set

A Kubernetes replica set ensures that the specified number of pods in a replica set are running at all times. If one pod dies or crashes, the replica set configuration will ensure a new one is created in its place. You would normally use a Deployment to manage this in place of a Replica Set. Learn more about Kubernetes ReplicaSets.

## Deployments

A way to define the desired state of pods or a replica set. Deployments are used to define HA policies to your containers by defining policies around how many of each container must be running at any one time.

## Services

Coupling of a set of pods to a policy by which to access them. Services are used to expose containerised applications to origins from outside the cluster. Learn more about Kubernetes Services.

## Nodes

A (normally) Virtual host(s) on which containers/pods are run.

# Kubernetes architecture and components

A K8S cluster is made of a master node, which exposes the API, schedules deployments, and generally manages the cluster. Multiple worker nodes can be responsible for container runtime, like Docker or rkt, along with an agent that communicates with the master.

## Master components

These master components comprise a master node:

- **Kube-apiserver.** Exposes the API.
- **Etcd.** Key value stores all cluster data. (Can be run on the same server as a master node or on a dedicated cluster.)
- **Kube-scheduler.** Schedules new pods on worker nodes.
- **Kube-controller-manager.** Runs the controllers.
- **Cloud-controller-manager.** Talks to cloud providers.

## Node components

- **Kubelet.** Agent that ensures containers in a pod are running.
- **Kube-proxy.** Keeps network rules and perform forwarding.
- **Container runtime.** Runs containers.

# What benefits does Kubernetes offer?

Out of the box, K8S provides several key features that allow us to run immutable infrastructure. Containers can be killed, replaced, and self-heal automatically, and the new container gets access to those support **volumes**, **secrets**, **configurations**, etc., that make it function.

These key K8S features make your containerized application scale efficiently:

- **Horizontal scaling.**Scale your application as needed from command line or UI.
- **Automated rollouts and rollbacks.**Roll out changes that monitor the health of your application—ensuring all instances don't fail or go down simultaneously. If something goes wrong, K8S automatically rolls back the change.
- **Service discovery and load balancing.**Containers get their own IP so you can put a set of containers behind a single DNS name for load balancing.
- **Storage orchestration.**Automatically mount local or public cloud or a network storage.
- **Secret and configuration management.**Create and update secrets and configs without rebuilding your image.
- **Self-healing.**The platform heals many problems: restarting failed containers, replacing and rescheduling containers as nodes die, killing containers that don't respond to your user-defined health check, and waiting to advertise containers to clients until they're ready.
- **Batch execution.**Manage your batch and Continuous Integration workloads and replace failed containers.

- **Automatic binpacking.**Automatically schedules containers based on resource requirements and other constraints.

## What won't Kubernetes do?

Kubernetes can do a lot of cool, useful things. But it's just as important to consider what Kubernetes isn't capable of:

- It does not replace tools like Jenkins—so it will not build your application for you.
- It is not middleware—so it will not perform tasks that a middleware performs, such as message bus or caching, to name a few.
- It does not care which logging solution is used. Have your app log to **stdout**, then you can collect the logs with whatever you want.
- It does not care about your config language (e.g., JSON).

K8s is not opinionated with these things simply to allow us to build our app the way we want, expose any type of information and collect that information however we want.

## Kubernetes competitors

Of course, Kubernetes isn't the only tool on the market. There are a variety, including:

- Docker Compose—good for staging but not production-ready.
- Nomad—allows for cluster management and scheduling but it does not solve secret and config management, service discover, and monitoring needs.
- Titus—Netflix's open-source orchestration platform doesn't have enough people using it in production.

Overall, Kubernetes offers the best out-of-the-box features along with countless third-party add-ons to easily extend its functionality.

## Getting Started with Kubernetes

Typically, you would install Kubernetes on either on premise hardware or one of the major cloud providers. Many cloud providers and third parties are now offering Managed Kubernetes services however, for a testing/learning experience this is both costly and not required. The easiest and quickest way to get started with Kubernetes in an isolated development/test environment is minikube.

### How to install Kubernetes

Installing K8S locally is simple and straightforward. You need two things to get up and running: Kubectl and Minikube.

- Kubectl is a CLI tool that makes it possible to interact with the cluster.
- Minikube is a binary that deploys a cluster locally on your development machine.

With these, you can start deploying your containerized apps to a cluster locally within just a few minutes. For a production-grade cluster that is highly available, you can use tools such as:

- Kops
- EKS, which is an AWS managed service
- GKE, provided by Google

Minikube allows you to run a single-node cluster inside a Virtual Machine (typically running inside VirtaulBox). Follow the official Kubernetes documentation to install minikube on your machine. https://kubernetes.io/docs/setup/minikube/.

With minikube installed you are now ready to run a virtualised single-node cluster on your local machine. You can start your minikube cluster with;

```
$ minikube start
```

Interacting with Kubernetes clusters is mostly done via the kubectl CLI or the Kubernetes Dashboard. The kubectl CLI also supports bash autocompletion which saves a lot of typing (and memory). Install the kubectl CLI on your machine by using the official installation instructions https://kubernetes.io/docs/tasks/tools/install-kubectl/.

To interact with your Kubernetes clusters you will need to set your kubectl CLI context. A Kubernetes context is a group of access parameters that defines which users have access to namespaces within a cluster. When starting minikube the context is automatically switched to minikube by default. There are a number of kubectl CLI commands used to define which Kubernetes cluster the commands execute against.

```
$ kubectl config get-context
$ kubectl config set-context <context-name>
```

```
➜ / kubectl config current-context
minikube
➜ /
```

```
$ Kubectl config delete-context <context-name>
```

## Deploying your first containerised application to Minikube

So far you should have a local single-node Kubernetes cluster running on your local machine. The rest of this tutorial is going to outline the steps required to deploy a simple Hello World containerised application, inside a pod, with an exposed endpoint on the minikube node IP address. Create the Kubernetes deployment with;

```
$ kubectl run hello-minikube --image=k8s.gcr.io/
```

```
➜ / kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 -port=8080

deployment.apps/hello-minikube created
➜ /
```

```
echoserver:1.4 --port=8080
```

We can see that our deployment was successful so we can view the deployment with;

```
➜ / kubectl get deployments
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-minikube   1         1         1            1           10m
➜ /
```

```
$ kubectl get deployments
```

Our deployment should have created a Kubernetes Pod. We can view the pods running in our cluster with;

```
➜ / kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
hello-minikube-6c47c66d8-5v4vz    1/1       Running   0          10m
➜ /
```

```
$ kubectl get pods
```

Before we can hit our Hello World application with a HTTP request from an origin from outside our cluster (i.e. our development machine) we need to expose the pod as a Kubernetes service. By default, pods are only accessible on their internal IP address which has no access from outside the cluster.

```
$ kubectl expose deployment hello-minikube --
```

```
➜ / kubectl expose deployment hello-minikube --type=NodePort
service/hello-minikube exposed
```

```
type=NodePort
```

Exposing a deployment creates a Kubernetes service. We can view the service with:

```
➜ / kubectl get services
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-minikube   NodePort    10.111.43.194   <none>        8080:30912/TCP   54s
kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP          12m
➜ /
```

```
$ kubectl get services
```

When using a cloud provider you would normally set —type=loadbalancer to allocate the service with either a private or public IP address outside of the ClusterIP range. minikube doesn't support load balancers, being a local development/testing environment and therefore —type=NodePort uses the minikube host IP for the service endpoint. To find out the URL used to access your containerised application type;

```
➜ / minikube service hello-minikube --url
http://192.168.99.100:30912
➜ /
```

```
$ minikube service hello-minikube -—url
```

Curl the response from your terminal to test that our exposed service is reaching our pod.

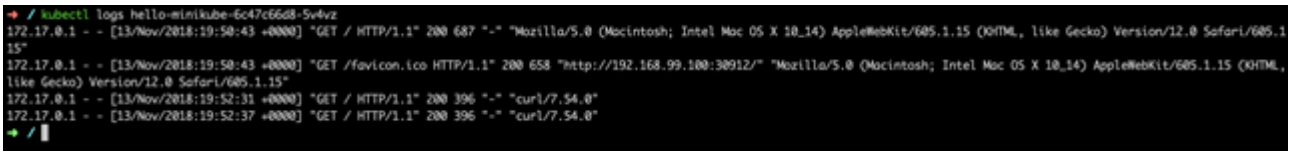
```
➜ / curl http://192.168.99.100:30912
CLIENT VALUES:
client_address=172.17.0.1
command=GET
real path=/
query=nil
request_version=1.1
request_uri=http://192.168.99.100:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
host=192.168.99.100:30912
user-agent=curl/7.54.0
BODY:
-no body in request-
➜ /
```

```
$ curl http://<minikube-ip>:<port>
```

Now we have made a HTTP request to our pod via the Kubernetes service, we can confirm that everything is working as expected. Checking the the pod logs we should see our HTTP request.




```
$ kubectl logs hello-minikube-c8b6b4fdc-sz67z
```

To conclude, we are now running a simple containerised application inside a single-node Kubernetes cluster, with an exposed endpoint via a Kubernetes service.

# Minikube for learning

Minikube is great for getting to grips with Kubernetes and learning the concepts of container orchestration at scale, but you wouldn't want to run your production workloads from your local machine. Following the above you should now have a functioning Kubernetes pod, service and deployment running a simple Hello World application.

From here, if you are looking to start using Kubernetes for your containerized applications, you would be best positioned looking into building a Kubernetes Cluster or comparing the many Managed Kubernetes offerings from the popular cloud providers.

# Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- The State of Kubernetes in 2020