. what is Kubernetes? K8S

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.

 K8s as an abbreviation results from counting the eight letters between the "K" and the "s".

what is the need of K8S?

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

Kubernetes provides you with:

* **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
* **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
* **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
* **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
* **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
* **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

How does Kubernetes work in GCP platform?

Kubernetes automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, monitoring your applications,

and more—making it easier to manage applications.

What is GKE?

Google Kubernetes Engine (GKE) provides a managed environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multiple machines (specifically, [Compute Engine](https://cloud.google.com/compute) instances) grouped together to form a [cluster](https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture).

what are the steps to deploy a spring boot appln onto a Kubernetes container in GKE?

## [Setup and requirements](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#1)

## [Get source code](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#2)

## [Locally run the app](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#3)

## [Package the Java app as a Docker container](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#4)

## [Create your cluster](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#5)

## [Deploy your app to Kubernetes](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#6)

## [Allow external traffic](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#7)

## [Scale your service](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#8)

## [Roll out an upgrade to your service](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#9)

## [Roll back](https://codelabs.developers.google.com/codelabs/cloud-springboot-kubernetes#10)

what is a Kubernetes Cluster?

A **Kubernetes cluster** is a set of nodes that run containerized applications. Containerizing applications packages an app with its dependences and some necessary services. They are more lightweight and flexible than virtual machines. In this way, Kubernetes clusters allow for applications to be more easily developed, moved and managed.

Kubernetes clusters allow containers to run across multiple machines and environments: virtual, physical, cloud-based, and on-premises. Kubernetes containers are not restricted to a specific operating system, unlike virtual machines. Instead, they are able to share operating systems and run anywhere

Kubernetes clusters are comprised of one master node and a number of worker nodes. These nodes can either be physical computers or virtual machines, depending on the cluster.

The **master node** controls the state of the cluster; for example, which applications are running and their corresponding container images. The master node is the origin for all task assignments. It coordinates processes such as:

* **Scheduling** and scaling applications
* **Maintaining** a cluster’s state
* **Implementing** updates

The **worker nodes** are the components that run these applications. Worker nodes perform tasks assigned by the master node. They can either be [virtual machines](https://www.vmware.com/topics/glossary/content/virtual-machine) or physical computers, all operating as part of one system.

There must be a minimum of one master node and one worker node for a Kubernetes cluster to be operational. For production and staging, the cluster is distributed across multiple worker nodes. For testing, the components can all run on the same physical or virtual node.

A **namespace** is a way for a Kubernetes user to organize many different clusters within just one physical cluster. Namespaces enable users to divide cluster resources within the physical cluster among different teams via resource quotas. For this reason, they are ideal in situations involving complex projects or multiple teams.

A Kubernetes cluster contains six main components:

1. **API server:** Exposes a REST interface to all Kubernetes resources. Serves as the front end of the Kubernetes control plane.
2. **Scheduler**: Places containers according to resource requirements and metrics. Makes note of Pods with no assigned node, and selects nodes for them to run on.
3. **Controller manager**: Runs controller processes and reconciles the cluster’s actual state with its desired specifications. Manages controllers such as node controllers, endpoints controllers and replication controllers.
4. **Kubelet**: Ensures that containers are running in a Pod by interacting with the Docker engine , the default program for creating and managing containers. Takes a set of provided PodSpecs and ensures that their corresponding containers are fully operational.
5. **Kube-proxy**: Manages network connectivity and maintains network rules across nodes. Implements the Kubernetes Service concept across every node in a given cluster.
6. **Etcd**: Stores all cluster data. Consistent and highly available Kubernetes backing store.

What is a Pod?

A pod is the smallest execution unit in Kubernetes. A pod encapsulates one or more applications. Pods are ephemeral by nature, if a pod (or the node it executes on) fails, Kubernetes can automatically create a new replica of that pod to continue operations. Pods include one or more [containers](https://www.vmware.com/topics/glossary/content/containers) (such as Docker containers).

Pods also provide environmental dependencies, including persistent storage volumes (storage that is permanent and available to all pods in the cluster) and configuration data needed to run the container(s) within the pod.

Pods represent the processes running on a cluster. By limiting pods to a single process, Kubernetes can report on the health of each process running in the cluster. Pods have:

* a unique IP address (which allows them to communicate with each other)
* persistent storage volumes (as required)
* configuration information that determine how a container should run.

Although most pods contain a single container, many will have a few containers that work closely together to execute a desired function.

what is a Node in Kubernetes?

 A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods across the Nodes in the cluster. The control plane's automatic scheduling takes into account the available resources on each Node.

A Pod always runs on a **Node**.

what is Kubelet

An agent that runs on each [node](https://kubernetes.io/docs/concepts/architecture/nodes/) in the cluster. It makes sure that [containers](https://kubernetes.io/docs/concepts/containers/) are running in a [Pod](https://kubernetes.io/docs/concepts/workloads/pods/).

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes

Kubelet, a process responsible for communication between the Kubernetes control plane and the Node; it manages the Pods and the containers running on a machine.

What is Kubectl?

The Kubernetes command-line tool, kubectl, **allows you to run commands against Kubernetes clusters**. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

what is a Docker? Explain?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

* Develop your application and its supporting components using containers.
* The container becomes the unit for distributing and testing your application.
* When you’re ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

why should we create a docker image? what does it contain?

A [Docker](https://searchitoperations.techtarget.com/definition/Docker) image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker [container](https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization), like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.

Docker is used to create, run and deploy applications in containers. A Docker image contains application code, [libraries](https://searchsqlserver.techtarget.com/definition/library), tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.

Docker images have multiple layers, each one originates from the previous layer but is different from it. The layers [speed up Docker builds](https://searchitoperations.techtarget.com/tip/Optimize-Docker-images-for-improved-efficiency-and-security) while increasing reusability and decreasing disk use. Image layers are also read-only files. Once a container is created, a writable layer is added on top of the unchangeable images, allowing a user to make changes.

what is the need of the container platform?

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

Kubernetes provides you with:

* **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
* **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
* **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
* **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
* **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
* **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

what are the benefits of deploying spring boot on a container platform?

Containers are a streamlined way to build, test, deploy, and redeploy applications on multiple environments from a developer’s local laptop to an on-premises data center and even the cloud. Benefits of containers include:

* **Less overhead**  
  Containers require less system resources than traditional or hardware virtual machine environments because they don’t include operating system images.
* **Increased portability**  
  Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms.
* **More consistent operation**  
  DevOps teams know applications in containers will run the same, regardless of where they are deployed.
* **Greater efficiency**  
  Containers allow applications to be more rapidly deployed, patched, or scaled.
* **Better application development**  
  Containers support agile and DevOps efforts to accelerate development, test, and production cycles.

If you are running your application from a container, you can use an executable jar, but it is also often an advantage to explode it and run it in a different way. Certain PaaS implementations may also choose to unpack archives before they run.

This is actually slightly faster on startup (depending on the size of the jar) than running from an unexploded archive. At runtime you shouldn’t expect any differences.

Once you have unpacked the jar file, you can also get an extra boost to startup time by running the app with its "natural" main method instead of the JarLauncher.