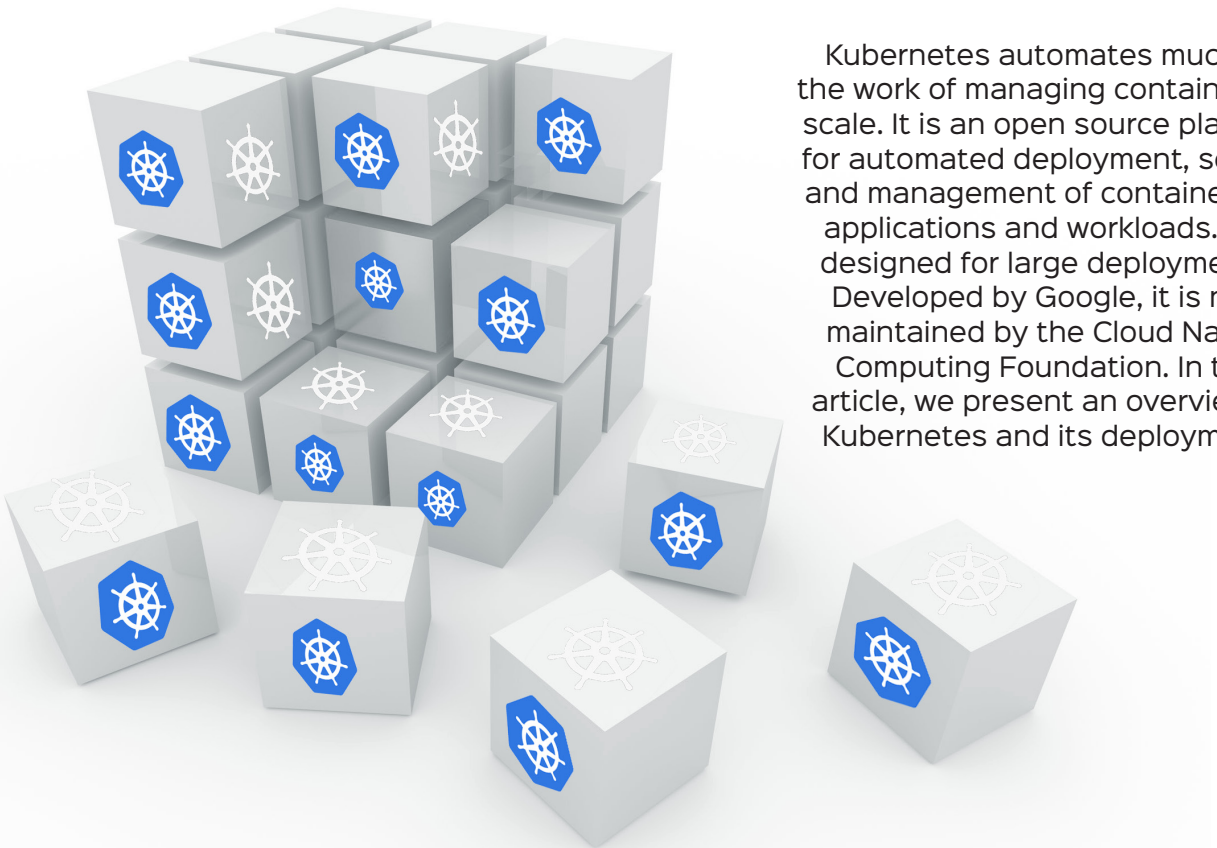
An Introduction to Cluster Creation and Deployment in Kubernetes



Kubernetes automates much of the work of managing containers at scale. It is an open source platform for automated deployment, scaling and management of containerised applications and workloads. It is designed for large deployments. Developed by Google, it is now maintained by the Cloud Native Computing Foundation. In this article, we present an overview of Kubernetes and its deployment.

As technologies like Docker and containerisation have become indispensable parts of developer and operations toolkits and have gained traction in organisations of all sizes, there is a need for better management tools and deployment environments. Kubernetes is one such tool.

What is Kubernetes?

Kubernetes (K8s) is an open source system for automating deployment, scaling, and management of containerised applications. It is the most popular and feature-rich container orchestration system in the world, and offers a wide range of features like:

- Automated rollouts and rollbacks
- Service discovery and load balancing
- Secret and configuration management
- Storage orchestration
- Automatic bin packing
- Batch execution
- Horizontal scaling

Kubernetes is designed for extensibility.

How does Kubernetes work?

Kubernetes operates on a three phase loop of checks and balances.

- **Observe**

  During the observe phase, Kubernetes aggregates a snapshot of the current state of the cluster. Kubelets collect state information about their respective nodes and feed this state back to the main process, giving the main a holistic view of the clusters' current state.

- **Check differences**
  The state snapshot from the observe phase is then compared to the expected static cluster parameters specified in the Kubernetes configuration. Any discrepancies between the current state and the expected cluster state are identified and slated for action.

- **Take action**
  The main then issues commands to bring the cluster back up to the expected state. This can mean removing or creating pods, horizontal or vertical scaling across nodes, and more. The above three phases are general descriptions of the Kubernetes loop. However, a Kubernetes cluster may actually have many different loops. These loops are executed by controllers. The ReplicationController or RC follows the loop phases. During the take action phase, RC is solely responsible for culling or creating new pods managed by the replica set.

Creation of Kubernetes clusters

A Kubernetes cluster is a set of nodes that runs containerised applications. Containerisation of applications packages an app with its dependencies and some necessary services. Kubernetes clusters allow containers to run across multiple machines and environments — virtual, physical, cloud based, and on-premises. They are not restricted to a specific OS, and are able to share the OS run anywhere.

A Kubernetes cluster consists of two types of resources:

- The control plane coordinates the cluster (it is responsible for managing the cluster).
- Nodes are the workers that run applications (a node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster).

Figure 1 explains the Kubernetes basic model, which includes creating Kubernetes, deploying an app, exploring the app, exposing the app publicly, as well as scaling up and updating the app.

App installation

After running the Kubernetes cluster one can deploy containerised applications on top of it. To create and manage a deployment by using the Kubernetes command line interface, *Kubectl*, the Kubernetes API, is used to interact with the cluster.

Inspecting the application and deployment

In the deployment phase of Kubernetes, it is necessary to know about pods and nodes. A pod is a group of one or more application containers (such as Docker) and includes shared storage (volumes), the IP address, and information about how to run them.

The most common operations can be done with the following *kubectl* commands.

- *kubectl get:* List resources
- *kubectl describe:* Show detailed information about a resource
- *kubectl logs:* Print the logs from a container in a pod
- *kubectl exec:* Execute a command on a container in a pod

Scaling and updating the application

Scaling is accomplished by changing the number of replicas in a deployment. It increases the number of pods to the new desired state. Kubernetes also supports auto-scaling of pods.

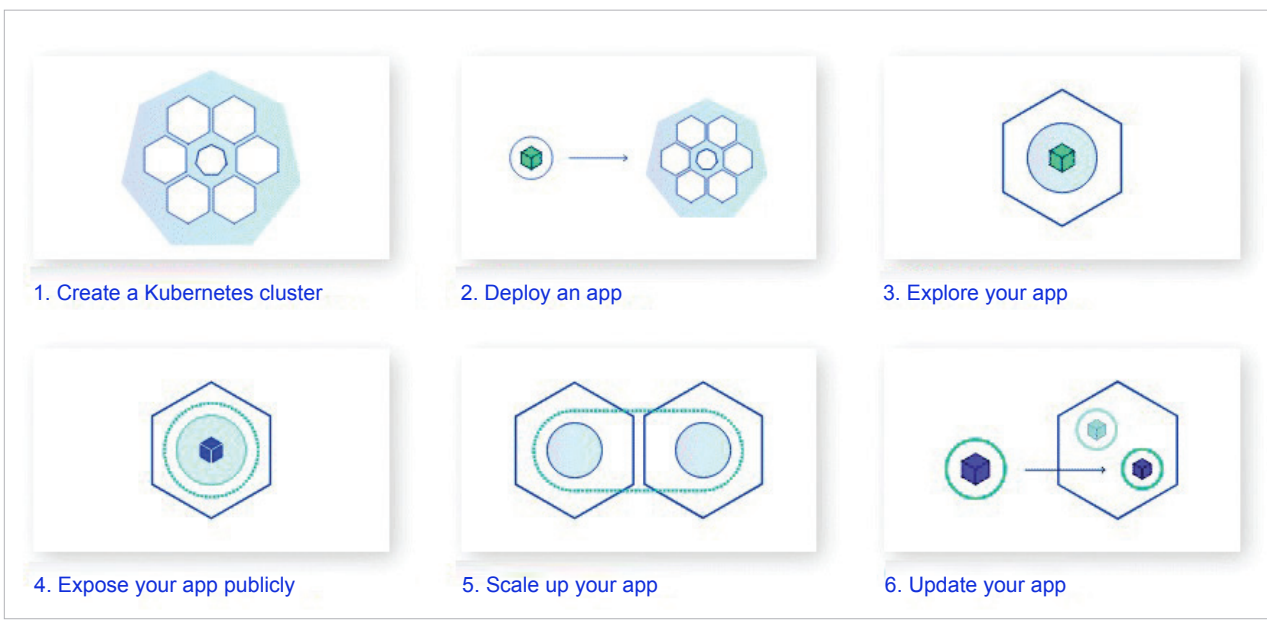


Figure 1: Basic modules of Kubernetes

Services have an integrated load balancer that distributes network traffic to all pods of an exposed deployment. Services continuously monitor the running pods using endpoints, to ensure the traffic is sent only to available pods.

Rolling updates allow updates of deployments to take place with zero downtime by incrementally updating pods instances with new ones. These allow the following actions:

- Promote an application from one environment to another (via container image updates)
- Rollback to previous versions
- Continuous integration and continuous delivery of applications with zero downtime

Managing Kubernetes applications

Kubernetes application management includes deployment and operations. Given below is a list of operational challenges:

- Hardening and compliance
- Managing configurations of all workloads deployed on the cluster
- Managing multi-tenant clusters
- Balancing security and agility

Deployment on a hosted Kubernetes

A pod is the basic execution unit of a Kubernetes application. Each pod represents a part of a workload that is running on a cluster. Type *kubectl* in a terminal to see its usage, and this performs the specified action (like create, describe) on the specified resource (like node, container).

```
$ kubectl
```

To view the nodes in the cluster, run the *kubectl get nodes* command. You can see the available nodes (available node name: *minikude*). Kubernetes will choose where to deploy our application based on available node resources:

```
$ kubectl get nodes
NAME       STATUS   ROLES                  AGE   VERSION
minikube   Ready    control-plane,master   19m   v1.20.2
```

To deploy the first app on Kubernetes with the *kubectl create deployment* command, it is necessary to provide the deployment name and app image location (include the full repository URL for images hosted outside the Docker hub):

```
$ kubectl create deployment kubernetes-bootcamp --image=gcr
.io/google-samples/kubernetes-bootcamp:v1
```

To list your deployments, use the *kubectl get deployments* command. We can see there is only one deployment running a single instance of the app. The instance is running inside a Docker container on a node.

```
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           23m
```

There are other options for exposing your application outside the Kubernetes cluster. To open a second terminal window to run the proxy, type *kubectl proxy* in a terminal. The proxy enables direct access to the API from these terminals:

```
Terminal   Terminal 2   +
$ echo -e "\n\n\n\e[92mStarting Proxy.
After starting it wie first Terminal Tab\n";  Please click the

Starting Proxy. After starting it will not output a response.
Please click the first Terminal Tab

$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

We can query the version directly through the API using the *curl* command.

Note: Check the top of the terminal. The proxy runs in a new tab (Terminal 2), and the recent commands were executed in the original tab (Terminal). The proxy still runs in the second tab, and this allows your *curl* command to work using *localhost:800*. If port 8001 is not accessible, ensure that the *kubectl* proxy started above is running.

```
Terminal   Terminal 2   +
Kubernetes Bootcamp Terminal

$ sleep 1; launch.sh
Starting Kubernetes. This is expected to take less than a m
inute.....
Kubernetes Started
$
$
$ curl http://localhost:8001/version
{
  "major": "1",
  "minor": "20",
  "gitVersion": "v1.20.2",
  "gitCommit": "faecb196815e248d3ecfb03c680a4507229c2a56",
  "gitTreeState": "clean",
  "buildDate": "2021-01-13T13:20:00Z",
  "goVersion": "go1.15.5",
  "compiler": "gc",
  "platform": "linux/amd64"
```

The road ahead

Since Kubernetes is open source, it allows the freedom to take advantage of on-premises, hybrid, or public cloud infrastructures. It lets you effortlessly move workloads to where it matters to you. Kubernetes can be combined with modern pipeline tools like Jenkins X to build highly agile and lean CI/CD (continuous integration and continuous delivery/ continuous deployment) systems. Modern, high-performance DevOps teams should make good use of Kubernetes. END

By: Dr S. Balakrishnan and Arun R.

Dr S. Balakrishnan is the professor and head, Department of Computer Science and Business System, Sri Krishna College of Engineering and Technology, Coimbatore.

Arun R. is an assistant professor for CSE, Hindusthan Institute of Technology, Coimbatore.