Kubernetes Basics

This tutorial provides a walkthrough of the basics of the Kubernetes cluster orchestration system. Each module contains some background information on major Kubernetes features and concepts and includes an interactive online tutorial. These interactive tutorials let you manage a simple cluster and its containerized applications for yourself.

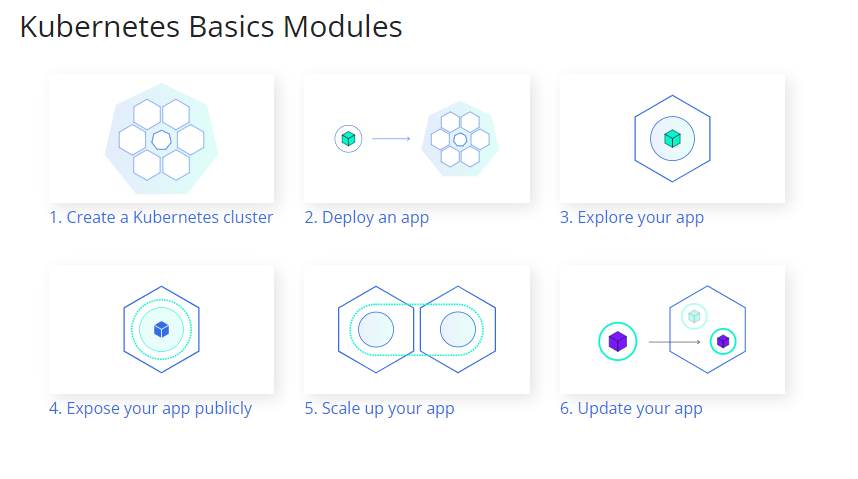
Using the interactive tutorials, you can learn to:

* Deploy a containerized application on a cluster.
* Scale the deployment.
* Update the containerized application with a new software version.
* Debug the containerized application.

The tutorials use Katacoda to run a virtual terminal in your web browser that runs Minikube, a small-scale local deployment of Kubernetes that can run anywhere. There's no need to install any software or configure anything; each interactive tutorial runs directly out of your web browser itself.

**What can Kubernetes do for you?**

With modern web services, users expect applications to be available 24/7, and developers expect to deploy new versions of those applications several times a day. Containerization helps package software to serve these goals, enabling applications to be released and updated without downtime. Kubernetes helps you make sure those containerized applications run where and when you want, and helps them find the resources and tools they need to work. Kubernetes is a production-ready, open source platform designed with Google's accumulated experience in container orchestration, combined with best-of-breed ideas from the community.



1. Create Cluster

**Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.**

**Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.**

A Kubernetes cluster consists of two types of resources:

* The **Control Plane** coordinates the cluster
* **Nodes** are the workers that run applications

Diagram

Description automatically generated

**The Control Plane is responsible for managing the cluster.** The Control Plane coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

**A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.** Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes control plane.

**Module 1 - Create a Kubernetes cluster**

**Step 1 of 3**

Cluster up and running

We already installed minikube for you. Check that it is properly installed, by running the minikube version command:

minikube version

OK, we can see that minikube is in place.

Start the cluster, by running the *minikube start* command:

minikube start

Great! You now have a running Kubernetes cluster in your online terminal. Minikube started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

**Step 2 of 3**

#### Cluster version

To interact with Kubernetes during this bootcamp we’ll use the command line interface, kubectl. We’ll explain kubectl in detail in the next modules, but for now, we’re just going to look at some cluster information. To check if kubectl is installed you can run the kubectl version command:

kubectl version

OK, kubectl is configured and we can see both the version of the client and as well as the server. The client version is the kubectl version; the server version is the Kubernetes version installed on the master. You can also see details about the build.

**Step 3 of 3**

#### Cluster details

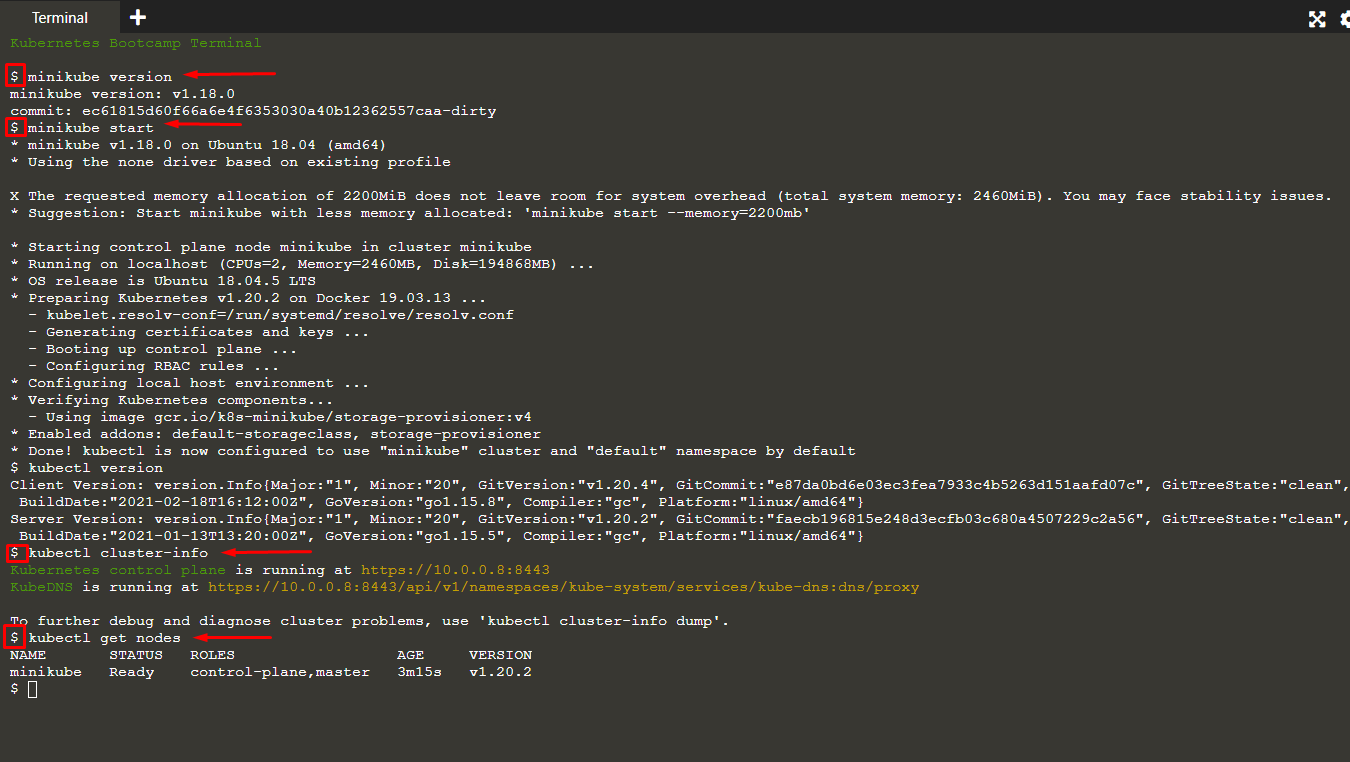
Let’s view the cluster details. We’ll do that by running kubectl cluster-info:

kubectl cluster-info

During this tutorial, we’ll be focusing on the command line for deploying and exploring our application. To view the nodes in the cluster, run the kubectl get nodes command:

kubectl get nodes

This command shows all nodes that can be used to host our applications. Now we have only one node, and we can see that its status is ready (it is ready to accept applications for deployment).



Interactive Tutorial - Deploying an App

**Step 1 of 3**

#### kubectl basics

Like minikube, kubectl comes installed in the online terminal. Type kubectl in the terminal to see its usage. The common format of a kubectl command is: kubectl action resource. This performs the specified action (like create, describe) on the specified resource (like node, container). You can use --help after the command to get additional info about possible parameters (kubectl get nodes --help).

Check that kubectl is configured to talk to your cluster, by running the kubectl version command:

kubectl version

OK, kubectl is installed and you can see both the client and the server versions.

To view the nodes in the cluster, run the kubectl get nodes command:

kubectl get nodes

Here we see the available nodes (1 in our case). Kubernetes will choose where to deploy our application based on Node available resources.

**Step 2 of 3**

#### Deploy our app

Let’s deploy our first app on Kubernetes with the kubectl create deployment command. We need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker hub).

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

Great! You just deployed your first application by creating a deployment. This performed a few things for you:

* searched for a suitable node where an instance of the application could be run (we have only 1 available node)
* scheduled the application to run on that Node
* configured the cluster to reschedule the instance on a new Node when needed

To list your deployments use the get deployments command:

kubectl get deployments

We see that there is 1 deployment running a single instance of your app. The instance is running inside a Docker container on your node.

**Step 3 of 3**

#### View our app

Pods that are running inside Kubernetes are running on a private, isolated network. By default they are visible from other pods and services within the same kubernetes cluster, but not outside that network. When we use kubectl, we're interacting through an API endpoint to communicate with our application.

We will cover other options on how to expose your application outside the kubernetes cluster in Module 4.

The kubectl command can create a proxy that will forward communications into the cluster-wide, private network. The proxy can be terminated by pressing control-C and won't show any output while its running.

We will open a second terminal window to run the proxy.

echo -e "\n\n\n\e[92mStarting Proxy. After starting it will not output a response. Please click the first Terminal Tab\n";

kubectl proxy

We now have a connection between our host (the online terminal) and the Kubernetes cluster. The proxy enables direct access to the API from these terminals.

You can see all those APIs hosted through the proxy endpoint. For example, we can query the version directly through the API using the curl command:

curl http://localhost:8001/version

Note: Check the top of the terminal. The proxy was run in a new tab (Terminal 2), and the recent commands were executed the original tab (Terminal 1). The proxy still runs in the second tab, and this allowed our curl command to work using *localhost:8001*.

**If Port 8001 is not accessible, ensure that the kubectl proxy started above is running.**

The API server will automatically create an endpoint for each pod, based on the pod name, that is also accessible through the proxy.

First we need to get the Pod name, and we'll store in the environment variable POD\_NAME:

export POD\_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}') echo Name of the Pod: $POD\_NAME

You can access the Pod through the API by running:

curl http://localhost:8001/api/v1/namespaces/default/pods/$POD\_NAME/

In order for the new deployment to be accessible without using the Proxy, a Service is required which will be explained in the next modules.