How to build a serverless app platform on Kubernetes

Learn how to build a Heroku-like platform using open source Cloud Native technologies.

By Peter Mbanugo

How to build a serverless app platform on Kubernetes

By Peter Mbanugo

Chapter 1: Introduction And Set Up

Kubernetes provides a set of primitives to run resilient, distributed applications. It takes care of the scaling and automatic failover for your application. It also provides deployment patterns and APIs that allow you to automate resource management and provision new workloads.

One of the main challenges that developers face is how to focus more on the details of the code rather than the infrastructure where that code runs. For that, serverless is one of the leading architectural paradigms to address this challenge. There are various platforms such as AWS Lambda, AWS Fargate, and Azure Functions, that allow you to run serverless applications either deployed as single functions or running inside containers. These managed platforms come with certain drawbacks like:

- Vendor lock-in
- Constraint in the size of the application binary/ artefacts
- Cold start performance

You could be in a situation where you're only allowed to run applications within a private data centre, or you may be using Kubernetes but you would like to harness the features of serverless architecture. There are different open-source platforms and tools, such as Knative and OpenFaaS, that use Kubernetes to abstract the infrastructure from the developer, allowing you to deploy and manage your applications using serverless architecture and patterns. Using any of those platforms eliminates the problems mentioned in the previous paragraph.

This book focuses on showing you how to build a platform to deploy and manage serverless applications on Kubernetes. It will be a platform that works similar to Heroku or Google Cloud Run, but with minimal features.

You will learn how to install Knative and deploy serverless applications on it. Afterwards, you will configure a continuous integration and deployment pipeline that will;

- take source code from Git,
- auto-detect the application and build a secure, OCI-compliant image,
- push the image to a container registry,
- and deploy the application on Knative.

In the end, you will have built a web application (using Next.js) that allows developers to connect to their GitHub repositories, and deploy code to Knative using an automated continuous delivery process.

The Serverless Landscape

Serverless computing is a deployment model that allows you to build and run applications without having to manage servers. It describes a model where a provider handles the routine work of provisioning, maintaining, and scaling the server infrastructure, while the developers can simply package and upload their code for deployment. Serverless apps can automatically scale up and down as needed, without any extra configuration by the developer.

As stated in a white paper by the CNCF serverless working group, there are two primary serverless personas:

1. **Developer**: Writes code for and benefits from the serverless platform that provides them with the point of view that there are no servers and that their code is always running.
2. **Provider**: Deploys the serverless platform for an external or internal customer.

The provider needs to manage servers (or containers) and will have some cost for running the platform, even when idle. A self-hosted system can still be considered serverless; Typically, one team acts as the provider and another as the developer.

There are various ways to run serverless apps. As seen in **Figure 1**, there are various tools, frameworks, and platforms to build and run serverless applications. It can be through managed serverless platforms like IBM Cloud Code and Google Cloud Run, or open-source alternatives that you can self-host, such as OpenFaaS and Knative.

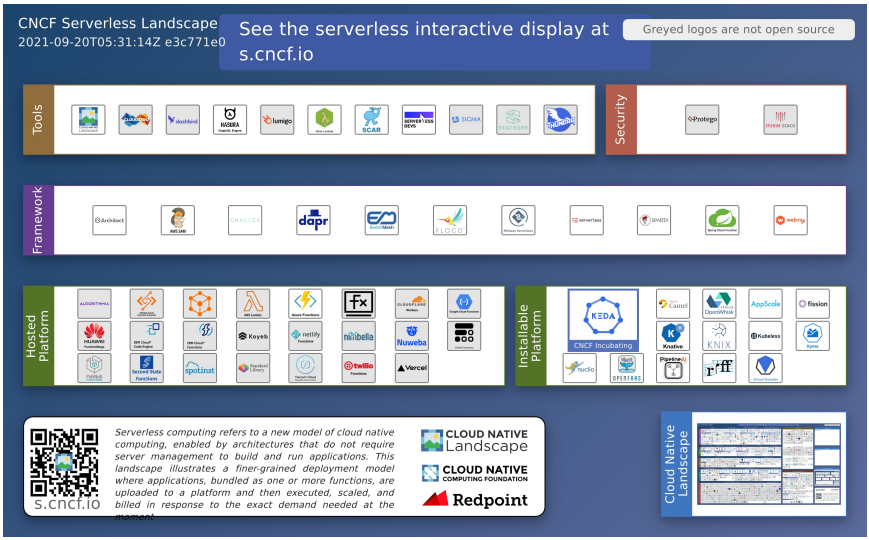

Figure 1: Cloud Native Computing Foundation (CNCF) Serverless Landscape

You should see Knative listed in **Figure 1**, on the lower right section labelled Installable Platforms. I chose the Knative project amongst all the others in that section because they have a nice supportive community on Slack, continuously improving the docs, and most importantly, any image deployed on Knative, can be deployed on any platform that can run OCI (Open Container Initiative) image. That means that I'm not locked to using only Knative for my app, and I can deploy it to other platforms without modifying the code or image. Contrast that to running containers on AWS Lambda, which requires a proprietary runtime that only works within their platform.

Another important reason for choosing Knative is because it is backed by big companies like Google, IBM, and

VMware (just to name a few), and it powers products like Google Cloud Run and IBM Cloud Code Engine.

The Open Container Initiative (OCI) is a Linux Foundation project, started in June 2015 by Docker, to design open standards for container image format and runtime. The OCI currently contains two specifications: the Runtime Specification and the Image Specification. To learn more about it, visit opencontainers.org, or visit red.ht/3hPJAck to learn more about some container-related terminology.

Cluster and Tools Set-Up

To follow the instructions in this book, you will need a Kubernetes cluster. It doesn't matter if it's a local cluster (e.g minikube or kind), or a remote cluster. The instructions in this book will focus on using a managed Kubernetes cluster from DigitalOcean and show alternative instructions for Docker Desktop when necessary. I implore you to use the managed Kubernetes service from DigitalOcean for this book because that's where I run the examples for this book.

Let's get started!

Install kubectl

The Kubernetes command-line tool, *kubectl*, allows you to run commands against Kubernetes clusters. Docker

Desktop installs kubectl for you, so if you have installed Docker Desktop, you should have kubectl installed already and you can skip this section. If you don't have kubectl installed, follow the instruction below to install it.

If you're on Linux or macOS, you can install kubectl using Homebrew by running the command ***brew install kubectl***. If you're on Windows, run the command ***curl -LO https://dl.k8s.io/release/v1.21.0/bin/windows/amd64/kubectl.exe*** to install kubectl, and then add the binary to your PATH.

Ensure that the version you installed is up to date by running the command ***kubectl version –client***.

I have version 1.21.4 when I wrote the examples for this book.

Create a Cluster with Docker Desktop

Docker Desktop includes a standalone Kubernetes server and client. This is a single-node cluster that runs within a Docker container on your local system and should be used only for local testing. To install Docker Desktop, go to the URL docs.docker.com/get-docker and download the appropriate binary for your OS.

To enable Kubernetes support and install a standalone instance of Kubernetes running as a Docker container, go to **Preferences** > **Kubernetes** and then click **Enable Kubernetes**.

Click **Apply & Restart** to save the settings and then click **Install** to confirm, as shown in **Figure 2**. This instantiates the images required to run the Kubernetes server as containers.

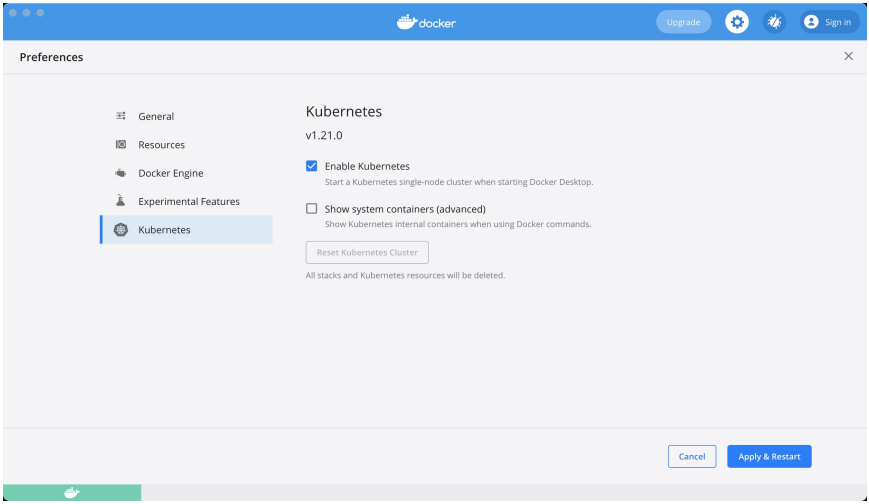

Figure 2: Enable Kubernetes on Docker Desktop

The status of Kubernetes shows in the Docker menu and the context points to **docker-desktop**, as shown in **Figure 3**.

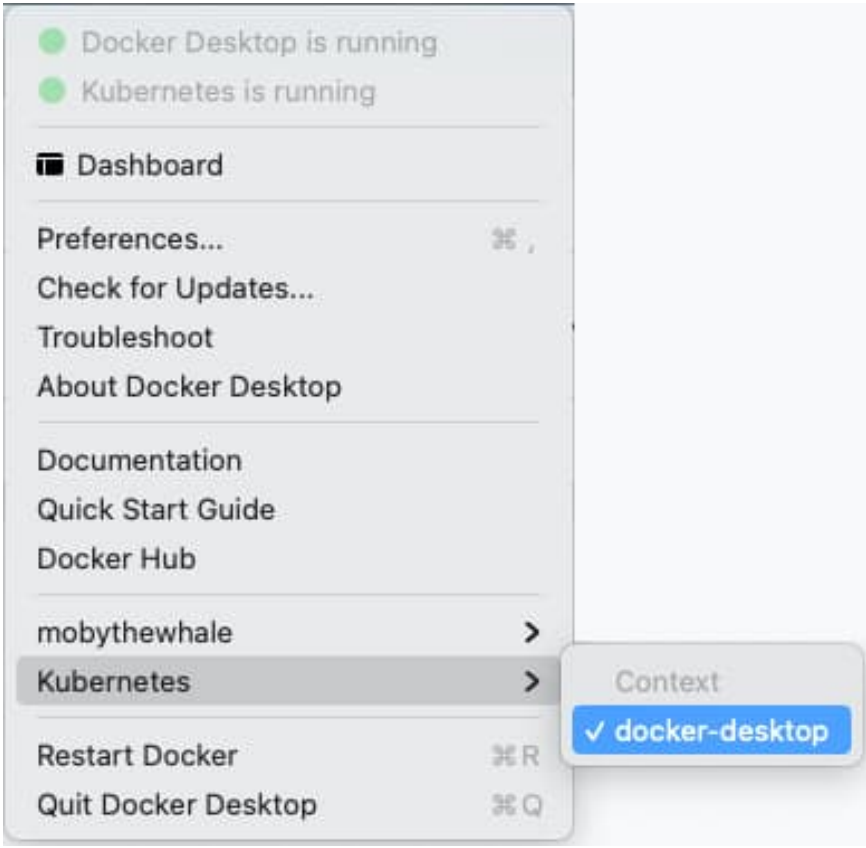

Figure 3: kube context

Create a Cluster on DigitalOcean

In order to use DigitalOcean Kubernetes Service, you need a DigitalOcean account. If you don't have an account, you can create one using my referral link (https://m.do.co/c/257c8259d8ef), which gives you $100 credit to try out different things on DigitalOcean.

You will create a cluster using *doctl*, the official command-line interface for the DigitalOcean API.

After you have created a DigitalOcean account, follow the instructions on docs.digitalocean.com/reference/doctl/how-to/install to install and configure *doctl*.

After you have installed and configured doctl, open your command line application and run the command below in order to create your cluster on DigitalOcean.

```
doctl kubernetes cluster create serverless-app-platform \
--region fra1 --size s-2vcpu-4gb --count 1
```

Wait for a few minutes for your cluster to be ready. When it is done, you should have a single-node cluster with the name serverless-app-platform, in Frankfurt. The size of the node is a machine with 2 vCPUs, and 4GB RAM. Also, the command you just executed will set the current kubectl context to that of the new cluster.

You can modify the values passed to the `doctl kubernetes cluster create` command. The *--region* flag indicates the cluster region. Run the command `doctl kubernetes options regions` to see possible values that can be used. The machine size to use when creating nodes is specified using the *--size* flag. Run the command `doctl kubernetes options sizes` for a list of possible values. The *--count* flag specifies the number of nodes to create. For prototyping purposes, you created a single-node cluster with 2 vCPUs and 4GB RAM.

Check that you can connect to your cluster by using kubectl to see the nodes.

Run the command kubectl get nodes. You should see one node in the list, and the **STATUS** should be in the **READY** state.

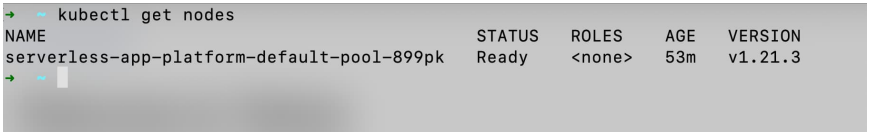

Figure 4: kubectl get nodes

Now that your cluster is ready, let's move on to the next chapter where you will learn about Knative and how to use it to run serverless applications.