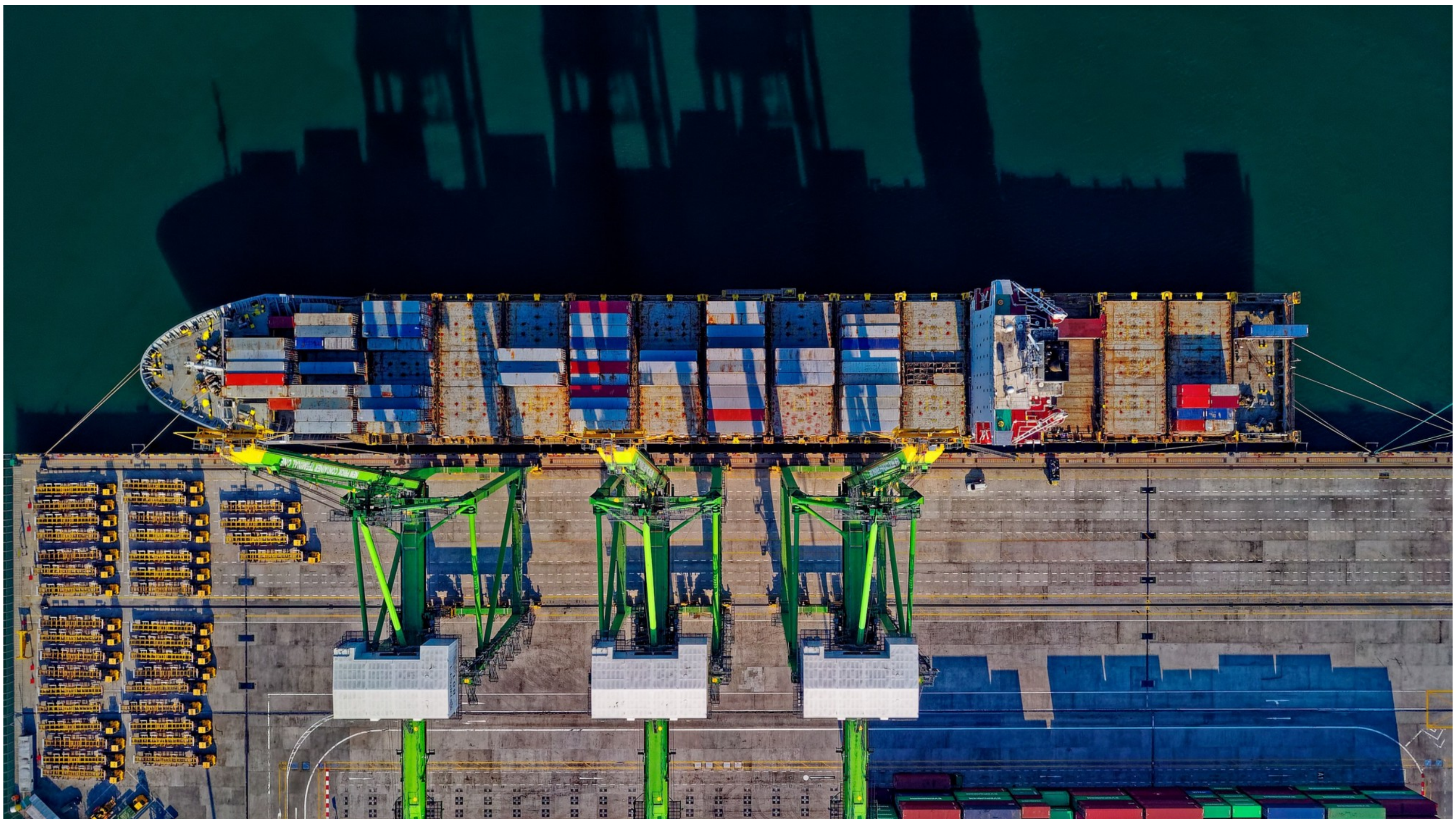
Why Kubernetes Is Awesome: A Beginner’s Guide.

Jan-David Stärk Follow
Dec 18 · 5 min read



What is Kubernetes?

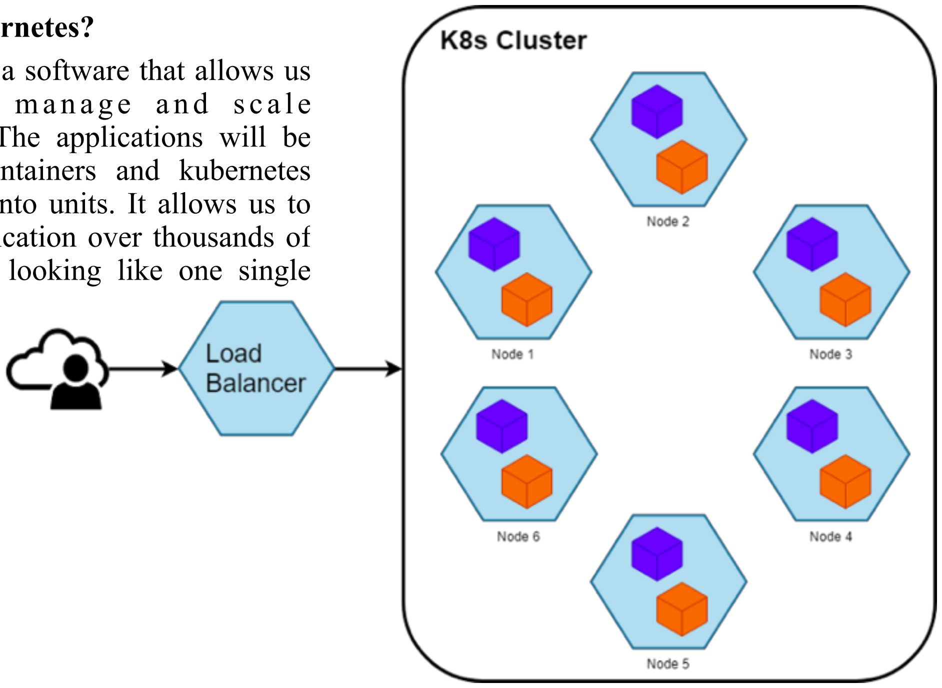
Kubernetes is a software that allows us to deploy, manage and scale applications. The applications will be packed in containers and kubernetes groups them into units. It allows us to span our application over thousands of servers while looking like one single unit.

How does Kubernetes work?

Okay, let’s have a quick look at what we’re going to build:

What is Kubernetes?

Kubernetes is a software that allows us to deploy, manage and scale applications. The applications will be packed in containers and kubernetes groups them into units. It allows us to span our application over thousands of servers while looking like one single unit.



Kubernetes overview

We've got a user who connects to a load balancer. The load balancer redirects the request to a node inside of the kubernetes cluster. The node then processes the request, delivering a response back.

> *Note: A node is a machine (virtual/physical) inside the cluster. Each node can contain multiple pods and a pod can contain one or more container. The concept of pods has been introduced to bundle container which is useful if they have to share resources like the filesystem.*

Working with Kubernetes (K8s) consists of five main steps:

1. **Develop** an application.
2. **Containerize** your application.
3. Create a kubernetes **cluster**.
4. **Deploy** your container to the cluster.
5. **Expose** and **scale** the cluster.

It's easy, isn't it? But let's jump straight into an example. Sometimes it's better to drive a car before learning how each component of it works 😉

Kubernetes: Let's go!

1. Develop an application.

Let's create a simple Node.js application—a simple HTTP server that returns a string with the OS' hostname and platform:

```javascript
const os = require('os');
const http = require('http');

const requestHandler = (req, res) => {
    console.log('Request incoming from ' + req.connect
    res.writeHead(200);
    res.end('Success! You\'ve hit ' + os.hostname() +
};
```

Our Node.js HTTP server.

We can fire it up by typing `node index.js` into our terminal. Verify that it is up 'n' running by opening `localhost:3000` in your local web browser. Once you've done that, go on!

2. Containerize your application.

Alright, we've got our Node.js application. Let's put it into a Docker container—the defacto standard for containerizing applications. Create a file called `DOCKERFILE` in the same directory as you Node.js application:

```dockerfile
FROM node:lts
ADD index.js /index.js
CMD node index.js
```

Our DOCKERFILE containing build information.

What this does is it basically just grabs the node LTS (long-term supported) version from the Docker Hub, adding our previously created index.js to the container. After that's done, the command `node index.js` will be executed (like we did earlier).

Alright, let's create the container now!

> *Note: You'll need a Docker account and a working Docker installation.*

Alright, let’s build and tag our container. The tag name is just `simple-http` and serves as some sort of an identifier:

```
docker build -t simple-http .
```

You can view your current images (containers) by executing `docker images` in your terminal. Try to play with your container, maybe even starting it. After you’ve done that, go on and re-tag your image like the following:

```
docker tag simple-http <<your_docker_username>>/simple-http
```

… and push it to Docker hub:

```
docker push <<your_docker_username>>/simple-http
```

After that, verify that it is working by trying to start your first Docker container. The last option, `-d`, tells Docker to run the image in the background. `-p` is an option to map ports: local port `3000` maps to the container’s internal port `3000`:

```
docker run --name server-container -p 3000:3000 -d
<<your_docker_username>>/simple-http
```

You can stop the container by executing `docker stop server-container`.

3. Create a kubernetes cluster.

Alright, let’s go all in and use GCP: Google Cloud Platform. It’s quite easy once you get started. Just sign up for it; you’ll get some sweet $$$ to test things out!

Once you’ve also installed the cloud SDK, add the kubectl tool:

```
gcloud components install kubectl
```

This installs `kubectl`, a command line tool (CLI) to manage kubernetes. It can take a while to install, so you can grab a cup of coffee in the meantime. ☕☕

After that, try to create a new project:

```
gcloud projects create simple-http-project — set-as-default
```

… and create a new kubernetes cluster called simple-http-kubes with 2 nodes:

```
gcloud container clusters create simple-http-kubes --num-nodes 2
--machine-type f1-micro --region us-east1
```

You'll probably receive an error asking you to enable the Kubernetes Engine API. Just follow the link and activate it ;) After that's done, reissue the command. 👌

> *Note: Due to GCP account restrictions, while you're just getting started it may not work if you're using more than 2 nodes.*

After that, you'll receive a response like that:

```
NAME               LOCATION  MASTER_VERSION  MASTER_IP
simple-http-kubes  us-east1  1.10.9-gke.5    35.229.48.
```

And now you're up and running! Open the MASTER_IP shown above in your browser and you will see … nothing. Why? Well, we've never told our little kube cluster which container it should be executing. Let's do that now!

4. Deploy your container to the cluster.

To deploy the application to your cluster, run the following command in your terminal:

```
kubectl run simple-http-kubes --image=
<<your_docker_username>>/simple-http --port=3000 --
generator=run/v1
```

That's it. Easy, isn't it?

5. Expose and scale the cluster.

But we also have to **expose** the cluster by using a load balancer:

```
kubectl expose rc simple-http-kubes --type=LoadBalancer --name
simple-http
```

Wait until your LoadBalance has got an external IP (you can check that with `kubectl get services`) and load it (and the port ofc 😉).

> Note: Use *curl* as a shortcut: *curl external-ip:3000*

So, we've got a few kubernetes nodes so far but just one replication. Let's change that. I mean, we wanna **scale** it, don't we?

```
kubectl scale rc simple-http-kubes --replicas=10
```

And we're done!

Wait until everything is up and running (check it by using *kubectl get pods*) and *boom:* if you reissue the curl command a few times, you should be able to see a different host sending the response! We did it! 🔥

```
Success! You've hit simple-http-kubes-s6l7w on linux!
```

And that's it! There's a lot to learn about kubernetes and it is definitely worth it! I recommend reading the book *Kubernetes in Action* by Marko Luksa.

Have a good day! 😊 👋

. . .

Kubernetes in Action (Amazon Ref Link)