****Deploying Laravel on Kubernetes****

****Prerequisites****

The following software needs to be installed on your local system:

1) [Docker](https://docs.docker.com/install/" \t "https://medium.com/hackernoon/_blank)

2) [Kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/" \t "https://medium.com/hackernoon/_blank)

3) [Minikube](https://github.com/kubernetes/minikube/releases" \t "https://medium.com/hackernoon/_blank)

****Docker image****

Kubernetes deploys containerised applications, and therefore as a first step, you will need to build a Docker image of the demo application. Since this will be run locally on Minikube, you can just build a local Docker Image from the **Dockerfile** included in the example code.

**FROM composer:1.6.5 as build   
WORKDIR /app   
COPY . /app   
RUN composer install**

**FROM php:7.1.8-apache   
EXPOSE 80   
COPY --from=build /app /app   
COPY vhost.conf /etc/apache2/sites-available/000-default.conf   
RUN chown -R www-data:www-data /app \   
 && a2enmod rewrite**

This DockerFile is made of two parts:

1. The first part extends a PHP composer image so that you can install the application’s dependencies.
2. The second part creates a final Docker image with an Apache web server to serve the application.

Before you can test the Docker image, you will need to build it:

**cd /to/your/project/directory   
docker build -t yourname/laravel-kubernetes-demo .**

Then run the application with:

**docker run -ti \   
 -p 8080:80 \   
 -e APP\_KEY=base64:cUPmwHx4LXa4Z25HhzFiWCf7TlQmSqnt98pnuiHmzgY= \   
 laravel-kubernetes-demo**

And the application should be available on **[http://localhost:8080.](http://localhost:8080./" \t "https://medium.com/hackernoon/_blank)**

With this setup, the container is generic and the **APP\_KEY** is not hardcoded or shared.

****Building the image within Minikube****

**cd /to/your/project/directory   
eval $(minikube docker-env)   
docker build -t yourname/laravel-kubernetes-demo .**

Don’t forget to execute the eval. Building the image within the virtual machine is necessary. You should run the command only once in the current terminal.

****Deploying the image****

Now that the application’s image is built and available in Minikube you can go ahead with deploying it.

Always start by making sure that kubectl is in the correct context. In this case, the context is Minikube. You can quickly switch context as follows:

kubectl config use-context minikube

then you can deploy the container image:

kubectl run laravel-kubernetes-demo \   
 --image=yourname/laravel-kubernetes-demo \   
 --port=80 \   
 --image-pull-policy=IfNotPresent \   
 --env=APP\_KEY=base64:cUPmwHx4LXa4Z25HhzFiWCf7TlQmSqnt98pnuiHmzgY=

The above command tells kubectl to run our demo application from the Docker image. The first parameter of the command simply asks kubectl to not pull the image from a registry such as Docker Hub if it exists locally which in this case it does. Do note that you still need to be logged on to Docker’s so that kubectl can check if the image is up to date.

You can check that a Pod is created for the application by running:

kubectl get pods

which should return a similar output to:

NAME READY STATUS RESTARTS AGE   
laravel-kubernetes-demo-7dbb9d6b48-q54wp 1/1 Running 0 18m

You can also use the Minikube GUI dashboard to monitor the cluster.

To view the dashboard, just run the following:

minikube dashboard

or to acquire the dashboard’s URL address:

minikube dashboard --url=true

****Exposing a Service****

So far you have created a deployment which is running the application’s container. A Pod running in the cluster has a dynamic IP. If you route the traffic directly to it using the IP, you may still need to update the routing table every time you restart the Pod. In fact, on every deployment or container restart, a new IP is assigned to the Pod. To avoid managing IP addresses manually, you need to use a Service. The Service acts as a load balancer for a set of Pods. So even if the IP address of a Pod changes, the service is always pointing to it. And since the Service always has the same IP, you won’t need to update anything manually.

You can create a service with:

kubectl expose deployment laravel-kubernetes-demo --type=NodePort --port=80

and provided all went well, you will see a confirmation similar to:

service "laravel-kubernetes-demo" exposed

Running the following command:

kubectl get services

will show you a list of running services. You can also view the running service under the “Services” navigation menu within the dashboard. To verify this deployment and the service exposure is obviously seeing the running application in the browser

To obtain the URL of the application (service), you can use the following command:

minikube service --url=true laravel-kubernetes-demo

which will output the IP address and port number similar to:

[http://192.168.99.101:31399](http://192.168.99.101:31399/" \t "https://medium.com/hackernoon/_blank)

or, launch the application directly in the browser:

minikube service laravel-kubernetes-demo

****Scaling****

And that is it. You have successfully deployed the application in Kubernetes.

Well, you only have one deployment with a single Pod running, provisioned to a Node with the exposed web service. Let’s scale this deployment to two more instances of the application.

So that you understand where you are at this moment, run the following command to get a list of desired and available Pods:

kubectl get deploymentNAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE   
laravel-kubernetes-demo 1 1 1 1 57m

The output will be “1” for each. You want to have three available Pods so let’s scale this up:

kubectl scale --replicas=3 deployment/laravel-kubernetes-demo   
deployment "laravel-kubernetes-demo" scaled

Done. You have replicated the first Pod to another two, giving you three Pods running this service. Running the get deployment command will verify this.

kubectl get deploymentNAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE   
laravel-kubernetes-demo 3 3 3 3 59m

You can also see this in the Dashboard under Pods or in the Service detail screen.

Now you’re running three instances of the applications using three Pods.

In Kubernetes you can scale to multiple instances in a snap:

kubectl scale --replicas=10 deployment/laravel-kubernetes-demo   
deployment "laravel-kubernetes-demo" scaled

You can see how convenient it is to use Kubernetes to scale your website.

****Ingress****

You deployed the application and scaled the deployment. You have already seen the running application in the browser when pointed to the cluster’s (Minikube) IP address and node’s port number. Now, you will see how to access the application through an assigned URL as you would do when deploying to the cloud.

To use a URL in Kubernetes, you need an Ingress. An Ingress is a set of rules to allow inbound connections to reach a Kubernetes cluster. The Ingress is necessary because, in Kubernetes, resources such as Pods only have IP addresses which are routable by and within the cluster. Meaning that they are not accessible or reachable to and from the world outside.

I have included an ingress.yaml file with the source code of this demo application with the following contents:

apiVersion: extensions/v1beta1   
kind: Ingress   
metadata:   
 name: laravel-kubernetes-demo-ingress   
 annotations: ingress.kubernetes.io/rewrite-target: /   
spec:   
 backend:   
 serviceName: default-http-server   
 servicePort: 80   
 rules:   
 - host: laravel-kubernetes.demo   
 - http:   
 paths:   
 - path: /   
 backend:   
 serviceName: laravel-kubernetes-demo   
 servicePort: 80

Among the basic content you would expect from a Kubernetes resource file, this file defines a set of rules to follow when routing inbound traffic. The laravel-kubernetes.demo URL will point to the Service where the application is running, as previously labelled laravel-kubernetes-demo on port 8181.

The Ingress resource is useless without an Ingress controller so you will need to create a new controller or use an existing one. Let’s use the Nginx Ingress controller for routing the traffic. Minikube (v0.14 and above) comes with the Nginx setup as an addon which you will need to enable manually:

minikube addons enable ingress

Please note that it may take few minutes for Minikube to download and install Nginx as an Ingress.

Once you have enabled the Ingress addon, you can create the Ingress in this way:

kubectl create -f path-to-your-ingress-file.yaml

You can verify and obtain the Ingress’ information by running the following command:

kubectl describe ing laravel-kubernetes-demo-ingress

which outputs something similar to:

**Name: laravel-kubernetes-demo-ingress   
Namespace: default   
Address: 192.168.99.101   
Default backend: default-http-server:80 (<none>)   
Rules:   
 Host Path Backends   
 ---- ---- --------   
 \*   
 / laravel-kubernetes-demo:8181 (172.17.0.6:8181)   
Annotations:   
 rewrite-target: /   
Events:   
Type Reason Age From Message   
---- ------ ---- ---- -------   
Normal CREATE 39s nginx-ingress-controller Ingress default/laravel-kubernetes-demo-ingressNormal UPDATE 20s nginx-ingress-controller Ingress default/laravel-kubernetes-demo-ingress**

You can now access the application through the minikube IP address as shown above. To access the application through the URL [https://laravel-kubernetes.demo,](https://laravel-kubernetes.demo,/" \t "https://medium.com/hackernoon/_blank) you will need to add an entry in your hosts file.