I just launched WordPress and MySQL on Kubernetes in 11 minutes

WordPress is arguably the most popular CMS for websites and blogs. By running WordPress on Kubernetes, we can build a reliable and scalable content management platform. It is available as a docker image with over 10 million pulls on DockerHub.

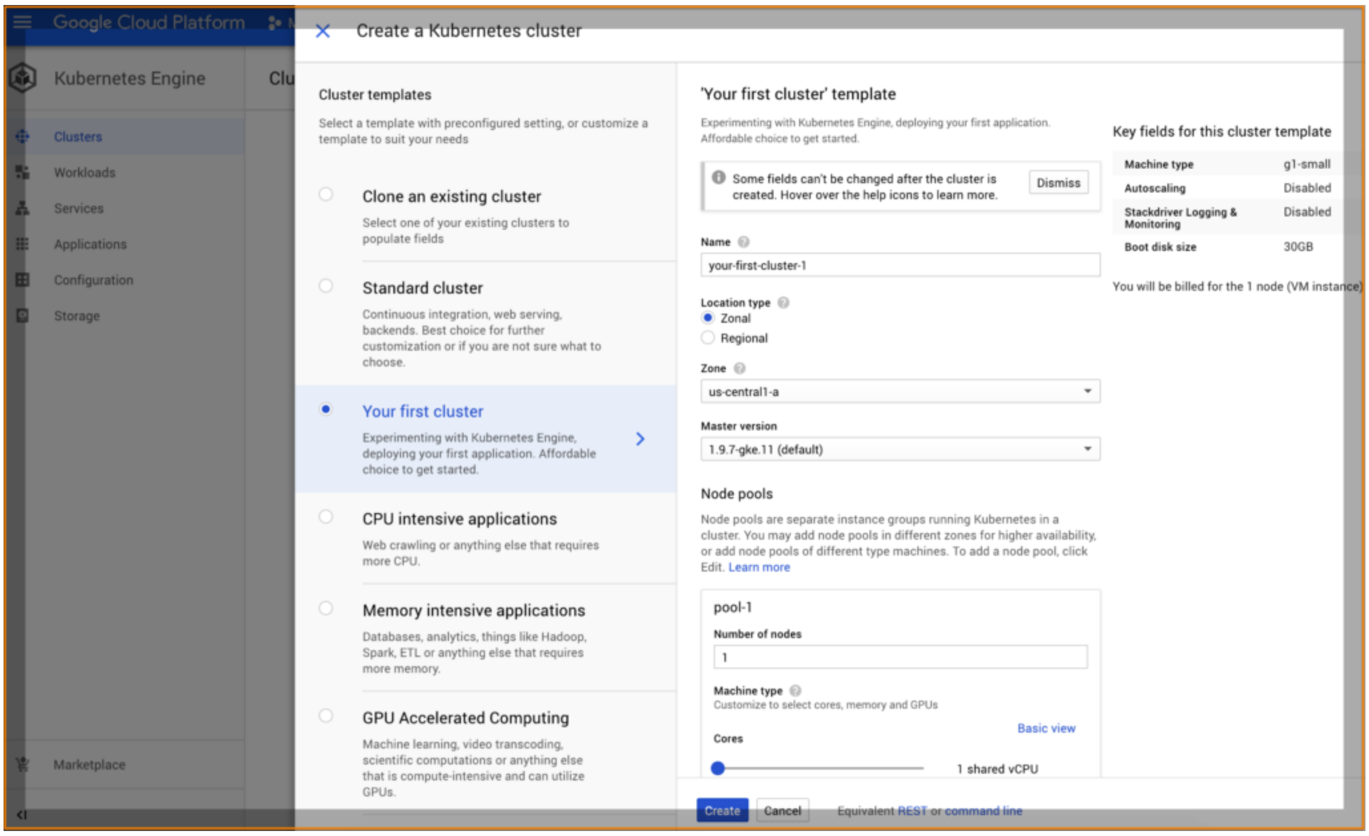
It’s believed that there are nearly two billion websites online today that means more or less one website for every four people in the world.

The fact that we have nearly two billion websites online today is thanks CMS, And according to BuiltWith Trends, WordPress has 59.9% market share making it top website platform and popular CMS.

So, If you are someone who cares about SEO, Speed, and Efficiency, Kubernetes is an answer because it allows large numbers of containers to work together in harmony, reducing operational burden.

> *If you want to upgrade your skills, i’ve got you covered in this piece about the Best Kubernetes Training Courses*

With Kubernetes we can run containers across many different machines. Scaling up or down by adding or removing containers when demand changes.


Google Cloud Platform

Kubernetes Engine

Clusters
Workloads
Services
Applications
Configuration
Storage
Marketplace

Create a Kubernetes cluster

Cluster templates

Select a template with preconfigured setting, or customize a template to suit your needs

Clone an existing cluster
Select one of your existing clusters to populate fields

Standard cluster
Continuous integration, web serving, backends. Best choice for further customization or if you are not sure what to choose.

Your first cluster
Experimenting with Kubernetes Engine, deploying your first application. Affordable choice to get started.

CPU intensive applications
Web crawling or anything else that requires more CPU.

Memory intensive applications
Databases, analytics, things like Hadoop, Spark, ETL or anything else that requires more memory.

GPU Accelerated Computing
Machine learning, video transcoding, scientific computations or anything else that is compute-intensive and can utilize GPUs.

'Your first cluster' template

Experimenting with Kubernetes Engine, deploying your first application. Affordable choice to get started.

Some fields can't be changed after the cluster is created. Hover over the help icons to learn more. Dismiss

Name: your-first-cluster-1

Location type: [✓ Zonal] Regional

Zone: us-central1-a

Master version: 1.9.7-gke.11 (default)

Node pools

Node pools are separate instance groups running Kubernetes in a cluster. You may add node pools in different zones for higher availability, or add node pools of different type machines. To add a node pool, click Edit. Learn more

pool-1

Number of nodes: 1

Machine type
Customize to select cores, memory and GPUs

Basic view

Cores: 1 shared vCPU

Create Cancel Equivalent REST or command line

Key fields for this cluster template

Machine type	g1-small
Autoscaling	Disabled
Stackdriver Logging & Monitoring	Disabled
Boot disk size	30GB

You will be billed for the 1 node (VM instance)

Source: Google Cloud

Alright, we heard that "Kubernetes is awesome" but now it about time to walk the walk. In the next 30 minutes, I'll take you through the entire process of creating your first WordPress running on Kubernetes and you better believe it's going to be awesome.

Deploy WordPress and MySQL on Kubernetes in 8 Easy Steps

Without further ado let's start with what you'll need. First off you'll need to sign up for Google Cloud and you also claim $300 Credit from Google Cloud valid up to 12 months. There are a few steps you'll have to take in order to be squared away but the entire process should take less than 5 minutes.

NFS Server Setup

We can use the NFS server to store WordPress and MySQL data. So, Let’s prepare it first.

Remember, you will need to change “`172.31.32.0/24`” to your private cluster subnet.

Create Backup Directory

We will need to create a backup directory for MySQL & WordPress files for volumes:

Create a Secret for MySQL Password

After MySQL setup, we need to use MySQL secrets engine. The MySQL Secrets engine Vault generates database credentials dynamically based on configured roles.

A Secret for MySQL is an object that stores a piece of sensitive data like a password or key.

Each item in a secret must be base64 encoded. So, let’s create a secret for admin use and encode the password for admin:

```
echo -n 'admin' | base64
```

You will get the encoded password: YRWtaW4=

Or we can also do the following according to the documentation,

Create the Secret object from the following command. You will need to replace YOUR_PASSWORD with the password you want to use.

> *Note: To protect the Secret from exposure, neither* `get` *nor* `describe` *show its contents.*

Create a secret.yml file for MySQL so that it will be mapped with WordPress as an Environment Variable:

This means that services that need to access a database no longer need to hard code credentials: they can request them from Vault, and use Vault’s leasing mechanism to more easily roll keys.

So, don’t forget to add your encoded password to your secret.yml above.

Once done, run the following command:

```
kubectl create -f secret.yml
```

Deploy Persistent Volume for WordPress & MySQL

Now, let’s go ahead with creating PV files and change the IP address of the NFS server you are using.

On Kubernetes, both WordPress and MySQL use PersistentVolumes (PV) and PersistentVolumeClaims (PVC) to store data.

Here’s a link to Kubernetes documentation, if you wish to learn more about PV and PVS.

PVC is a request for storage that can at some point become available, bound to some actual PV.

Let’s create a PVC for WordPress first with the following content:

And then execute the following command to Deploy the MySQL from the mysql-deployment.yaml file:

```
kubectl create -f pvc-wordpress.yml
```

Then let’s verify that a PersistentVolume got dynamically provisioned.

Note that it can take up to a few minutes for the PVs to be provisioned and bound.

Deploy PersistentVolumeClaim(PVC)

PVC is a request for storage that can at some point become available, bound to some actual PV.

Create a PVC for WordPress with the following content:

And then run:

```
kubectl create -f pvc-wordpress.yml
```

Do the same for MySQL:

Run:

```
kubectl create -f pvc-mysql.yml
```

Verify that a PersistentVolume got dynamically provisioned:

```
kubectl get pvc
```

> *Note: It can take up to a few minutes for the PVs to be provisioned and bound.*

The response should be like this:

Verify that the Service is running by running the following command:

```
kubectl get services wordpress
```

The response should be like this:

> *Note: Minikube can only expose Services through NodePort. The EXTERNAL-IP is always pending.*

Great, we’ve made it to the deployment step.

Deploy MySQL

Here’s the mysql-deploy.yml for MySQL service and deployment:

The file consists of two separate configurations:

The Service part maps MySQL’s port 3306 and makes it available for all containers with the labels app:wordpress and tier:mysql.

The Deployment part declares the strategy for creating and specs of our MySQL container:

- It’s an image from the Docker Hub: `mysql:5.6`

- It has `app:wordpress` and `tier:frontend` labels (used in Service)
- It also contains an environment variable called `MYSQL_ROOT_PASSWORD` which holds the value from our secret password
- Port `3306` is now open
- Volume is claimed and mounted in `/var/lib/mysql`.

Now we can create the deployment and service by executing the following command:

```
kubectl create -f mysql-deploy.yml
```

Deploy WordPress

The process above is pretty much the same for WordPress. Here's the wordpress-deploy.yml:

As you might have noticed that this file also consists of two configurations:

Service maps port `80` of the container to the node's `external IP:Port` for all containers with the labels `app:wordpress` and `tier:frontend`

Deployment declares the creation spec of our WordPress container:

- It's an image from the Docker Hub: `wordpress:4.8-apache`
- It has `app:wordpress` and `tier:frontend` labels (used in Service)
- It contains environment variables `WORDPRESS_DB_HOST`, which is the internal hostname of the MySQL instance, and `WORDPRESS_DB_PASSWORD`, which

holds the value from our secret password
Port `80` is open
it has a volume claim mounted in `/var/www/html` from which the WordPress sources are served.

Here’s what we need to execute next:

```
kubectl create -f wordpress-deploy.yml
```

Verify that a PersistentVolume got dynamically provisioned:

```
  kubectl get pvc
```

Note: It can take up to a few minutes for the PVs to be provisioned and bound.

The response should be like this:

Verify that the Service is running by running the following command:

```
kubectl get services wordpress
```

The response should be like this:

Note: Minikube can only expose Services through NodePort.The EXTERNAL-IP is always pending.

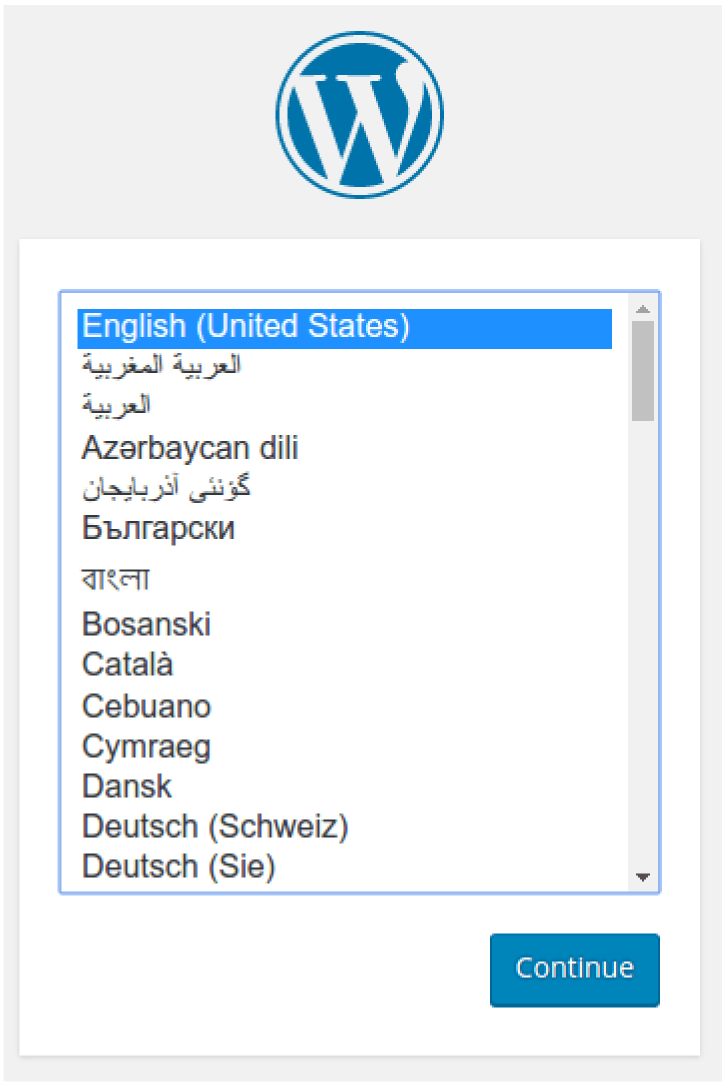
Run the following command to get the IP Address for the WordPress Service:

```
minikube service wordpress --url
```

Launch WordPress

To access WordPress, Copy the IP address, and load the page in your browser to view your site.

You should see the WordPress set up the page similar to the following screenshot.



Warning: Do not leave your WordPress installation on this page. If another user finds it, they can set up a website on your instance and use it to serve malicious content.

Either install WordPress by creating a username and password or delete your instance.

Done! Now you can create your own website from the WordPress Dashboard.

Troubleshooting

You can check if everything works fine by using the following commands:

On NFS Server

Check NFS server and you will see your data under /mysql and /html

```
ls /mysql
```

```
ls /html
```

After installing WordPress try to delete one pod and it will mount data from /html automatically:

```
kubeclt delete pods pod-name
```

Cleaning up

Run the following command to delete your Secret:

```
kubectl delete secret mysql-pass
```

Run the following commands to delete all Deployments and Services:

```
kubectl delete deployment -l app=wordpress
kubectl delete service -l app=wordpress
```

Run the following commands to delete the PersistentVolumeClaims. The dynamically provisioned PersistentVolumes will be automatically deleted.

```
kubectl delete pvc -l app=wordpress
```

Thanks for making it to the end

You have now deployed WordPress with MySQL, Persistent volumes, and NFS using Kubernetes engine.

go ahead and boldly Say "Hello World"

If you liked this article, I've got a few very practical reads for you. One about the Kubernetes Courses from the World-Class Educators and one about Google Cloud Certification Specializations.

I've also got this data science newsletter that you might be into. I send a tiny email once every fortnight with some useful and cool stuff I've found/made.

Don't worry, I hate spam as much as you. 👇