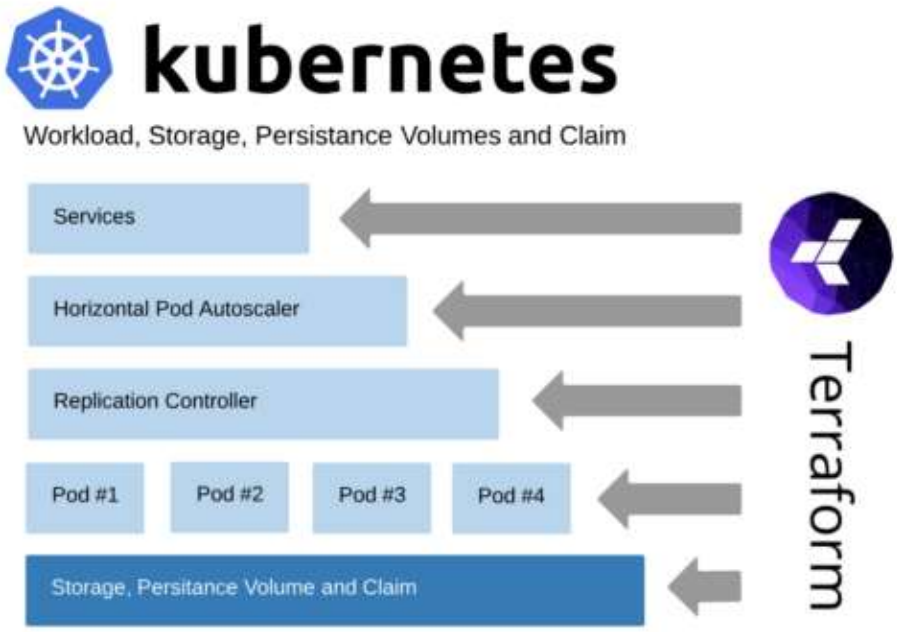
Christopher Quiles

Follow 67 Followers About

# Deploying Kubernetes Cluster Using Terraform

Christopher Quiles Jan 22 · 5 min read

Utilizing Kubernetes with Terraform, by scheduling and exposing a NGINX deployment on a Kubernetes cluster.




**Kubernetes** (K8S) is an open-source workload scheduler with focus on containerized applications. “In other words, you can cluster together groups of hosts running Linux containers, and Kubernetes helps you easily and efficiently manage those clusters.”

There are many advantages in using Terraform to provision Kubernetes Cluster:

- Allows maintaining Kubernetes Cluster definitions in Code.

- The biggest benefit when using Terraform to maintain Kubernetes resources is integration into the Terraform plan/apply life-cycle. So you can review planned changes before applying them. Also, using `kubectl`, purging of resources from the cluster is not trivial without manual intervention. Terraform does this reliably.

When setting up a Kubernetes workload, it is possible to use Terraform to directly schedule the pods. After Terraform provisions the pod, Kubernetes is responsible for managing the containers within.

## Getting Started:

If you don't have a Kubernetes cluster, you can use kind to provision a local Kubernetes cluster or provision one on a cloud provider.

Use the package manager `homebrew` to install kind.

```
$ brew install kind

$ curl https://raw.githubusercontent.com/hashicorp/learn-terraform-deploy-nginx-kubernetes-provider/master/kind-config.yaml --output kind-config.yaml
```

Once you've done this, download and save the kind configuration into a file named `kind-config.yaml`. This configuration adds extra port mappings, so you can access the NGINX service locally later.

```
1  kind: Cluster
2  apiVersion: kind.x-k8s.io/v1alpha4
3  nodes:
4  - role: control-plane
5    extraPortMappings:
6    - containerPort: 30201
7      hostPort: 30201
8      listenAddress: "0.0.0.0"
```

kind-config.yaml hosted with ♡ by GitHub view raw

vim kind-config.yaml

ERROR: problems with docker daemon.

**ERROR:** However for some reason my **docker daemon** wasn't running. Apparently after some research it is a complicated process to get Docker to run on MacOS. Then I found there is a way with Docker Machine but it couldn't seem to get Virtualbox installed, so long story short we'll skip that mess and hop into a Virtual Environment moving forward.

```
python3 -m pip install --user --upgrade pip

python3 -m pip install --user virtualenv

python3 -m venv env

source env/bin/activate
```

Virtual Env hosted with ♡ by GitHub view raw

process for installing and activating VENV for MacOs

Verify that your cluster exists by listing your kind clusters.

```
kind get clusters
```

Then, point `kubectl` to interact with this cluster. The context is `kind-` followed by the name of your cluster.

```
kubectl cluster-info --context kind-terraform-learn
```

```
(env) jerryc.quiles@Jerrys-MacBook-Pro ~ % kind get clusters
terraform-learn
(env) jerryc.quiles@Jerrys-MacBook-Pro ~ % kubectl cluster-info --context kind-terraform-learn
Kubernetes master is running at https://127.0.0.1:55521
KubeDNS is running at https://127.0.0.1:55521/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

## Configure the provider

**kubectl.** You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

```
$ mkdir learn-terraform-deploy-nginx-kubernetes
$ cd learn-terraform-deploy-nginx-kubernetes
$ vim kubernetes.tf
$ kubectl config cuurent-context
```

```
jerryc.quiles@Jerrys-MacBook-Pro projects % mkdir learn-terraform-deploy-nginx-kubernetes
jerryc.quiles@Jerrys-MacBook-Pro projects % cd learn-terraform-deploy-nginx-kubernetes
jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % vim kubernetes.tf
jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % kubectl config current-context
```

Then, create a new file named `kubernetes.tf` and add the following configuration to it.

```
terraform {
  required_providers {
    kubernetes = {
      source = "hashicorp/kubernetes"
    }
  }
}

provider "kubernetes" {}

resource "kubernetes_deployment" "nginx" {
  metadata {
    name = "scalable-nginx-example"
    labels = {
      App = "ScalableNginxExample"
    }
  }

  spec {
    replicas = 4
    selector {
      match_labels = {
        App = "ScalableNginxExample"
      }
    }
    template {
      metadata {
        labels = {
```

```
      }
      spec {
        container {
          image = "nginx:1.7.8"
          name  = "example"

          port {
            container_port = 80
          }

          resources {
            limits {
              cpu    = "0.5"
              memory = "512Mi"
            }
            requests {
              cpu    = "250m"
              memory = "50Mi"
            }
          }
        }
      }
    }
  }

  resource "kubernetes_service" "nginx" {
    metadata {
      name = "nginx-example"
    }
    spec {
      selector = {
        App = kubernetes_deployment.nginx.spec.0.template.0.metadata[0].labels.App
      }
      port {
        node_port   = 30201
        port        = 80
        target_port = 80
      }

      type = "NodePort"
    }
  }

```

Verify `kubectl`'s current-context is pointing to your Kubernetes cluster. If you're running kind, your current-context should be `kind-terraform-learn`.

```
(env) jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % kubectl config current-context
kind-terraform-learn
(env) jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % kubectl config use-context kind-terraform-learn
Switched to context "kind-terraform-learn".
```

if you don't see kind-terraform-learn switch by using $ kubectl config use-context kind-terraform-learn

run `terraform init` to download the latest version and initialize your Terraform workspace.

```
$ terraform init
```

```
(env) jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % terraform init

Initializing the backend...

Initializing provider plugins...
- Using previously-installed hashicorp/kubernetes v1.13.3

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, we recommend adding version constraints in a required_providers block
in your configuration, with the constraint strings suggested below.

* hashicorp/kubernetes: version = "~> 1.13.3"

Terraform has been successfully initialized!
```

## Schedule a deployment:

Add the following to your `kubernetes.tf` file. This Terraform configuration will schedule a NGINX deployment with two replicas on your Kubernetes cluster, internally exposing port 80 (HTTP).

```
terraform {
  required_providers {
    kubernetes = {
      source = "hashicorp/kubernetes"
    }
  }
}

provider "kubernetes" {}
```

```
    name = "scalable-nginx-example"
    labels = {
      App = "ScalableNginxExample"
    }
  }

  spec {
    replicas = 4
    selector {
      match_labels = {
        App = "ScalableNginxExample"
      }
    }
    template {
      metadata {
        labels = {
          App = "ScalableNginxExample"
        }
      }
      spec {
        container {
          image = "nginx:1.7.8"
          name  = "example"

          port {
            container_port = 80
          }

          resources {
            limits {
              cpu    = "0.5"
              memory = "512Mi"
            }
            requests {
              cpu    = "250m"
              memory = "50Mi"
            }
          }
        }
      }
    }
  }

  resource "kubernetes_service" "nginx" {
```

```
    }
    spec {
      selector = {
        App = kubernetes_deployment.nginx.spec.0.template.0.metadata[0].labels.App
      }
      port {
        node_port   = 30201
        port        = 80
        target_port = 80
      }

      type = "NodePort"
    }
  }

```

```
terraform {
  required_providers {
    kubernetes = {
      source = "hashicorp/kubernetes"
    }
  }
}

provider "kubernetes" {}

resource "kubernetes_deployment" "nginx" {
  metadata {
    name = "scalable-nginx-example"
    labels = {
      App = "ScalableNginxExample"
    }
  }

  spec {
    replicas = 4
    selector {
      match_labels = {
        App = "ScalableNginxExample"
      }
    }
    template {
      metadata {
        labels = {
          App = "ScalableNginxExample"
        }
      }
      spec {
        container {
          image = "nginx:1.7.8"
          name  = "example"

          port {
            container_port = 80
          }

          resources {
            limits {
              cpu    = "0.5"
              memory = "512Mi"
            }
            requests {
              cpu    = "250m"
              memory = "50Mi"
            }
          }
        }
      }
    }
  }
}
```

Apply the configuration to schedule the NGINX deployment.

```
$ terraform apply
```

```
kubernetes_deployment.nginx: Creating...
kubernetes_deployment.nginx: Still creating... [10s elapsed]
kubernetes_deployment.nginx: Creation complete after 16s [id=default/scalable-nginx-example]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

```
(env) jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % terraform apply
kubernetes_deployment.nginx: Refreshing state... [id=default/scalable-nginx-example]
kubernetes_service.nginx: Refreshing state... [id=default/nginx-example]

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:
```

```
$ kubectl get deployments
```

Once the apply is complete, verify the NGINX deployment is running.

```
(env) jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
scalable-nginx-example   2/2     2            2           2d16h
(env) jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes %
```

## Schedule a Service

Since our Kubernetes cluster is hosted locally on kind, we will expose the NGINX instance via **NodePort** to access the instance. This exposes the service on each node's IP at a static port, allowing you to access the service from outside the cluster at `<NodeIP>:<NodePort>`.

Add the following configuration to your `kubernetes.tf` file. This will expose the NGINX instance at the `node_port` — `30201`.

```
terraform {
  required_providers {
    kubernetes = {
      source = "hashicorp/kubernetes"
```

```
8
9   provider "kubernetes" {}
10
11  resource "kubernetes_deployment" "nginx" {
12    metadata {
13      name = "scalable-nginx-example"
14      labels = {
15        App = "ScalableNginxExample"
16      }
17    }
18
19    spec {
20      replicas = 4
21      selector {
22        match_labels = {
23          App = "ScalableNginxExample"
24        }
25      }
26      template {
27        metadata {
28          labels = {
29            App = "ScalableNginxExample"
30          }
31        }
32        spec {
33          container {
34            image = "nginx:1.7.8"
35            name  = "example"
36
37            port {
38              container_port = 80
39            }
40
41            resources {
42              limits {
43                cpu    = "0.5"
44                memory = "512Mi"
45              }
46              requests {
47                cpu    = "250m"
48                memory = "50Mi"
49              }
50            }
51          }
52        }
```

```
}

resource "kubernetes_service" "nginx" {
  metadata {
    name = "nginx-example"
  }
  spec {
    selector = {
      App = kubernetes_deployment.nginx.spec.0.template.0.metadata[0].labels.App
    }
    port {
      node_port   = 30201
      port        = 80
      target_port = 80
    }

    type = "NodePort"
  }
}

```

Here is the full kubernetes.tf configuration file.

```
terraform {
  required_providers {
    kubernetes = {
      source = "hashicorp/kubernetes"
    }
  }
}

provider "kubernetes" {}

resource "kubernetes_deployment" "nginx" {
  metadata {
    name = "scalable-nginx-example"
    labels = {
      App = "ScalableNginxExample"
    }
  }

  spec {
    replicas = 4
    selector {
```

```
    }
    template {
      metadata {
        labels = {
          App = "ScalableNginxExample"
        }
      }
      spec {
        container {
          image = "nginx:1.7.8"
          name  = "example"

          port {
            container_port = 80
          }

          resources {
            limits {
              cpu    = "0.5"
              memory = "512Mi"
            }
            requests {
              cpu    = "250m"
              memory = "50Mi"
            }
          }
        }
      }
    }
  }

  resource "kubernetes_service" "nginx" {
    metadata {
      name = "nginx-example"
    }
    spec {
      selector = {
        App = kubernetes_deployment.nginx.spec.0.template.0.metadata[0].labels.App
      }
      port {
        node_port   = 30201
        port        = 80
        target_port = 80
      }
```

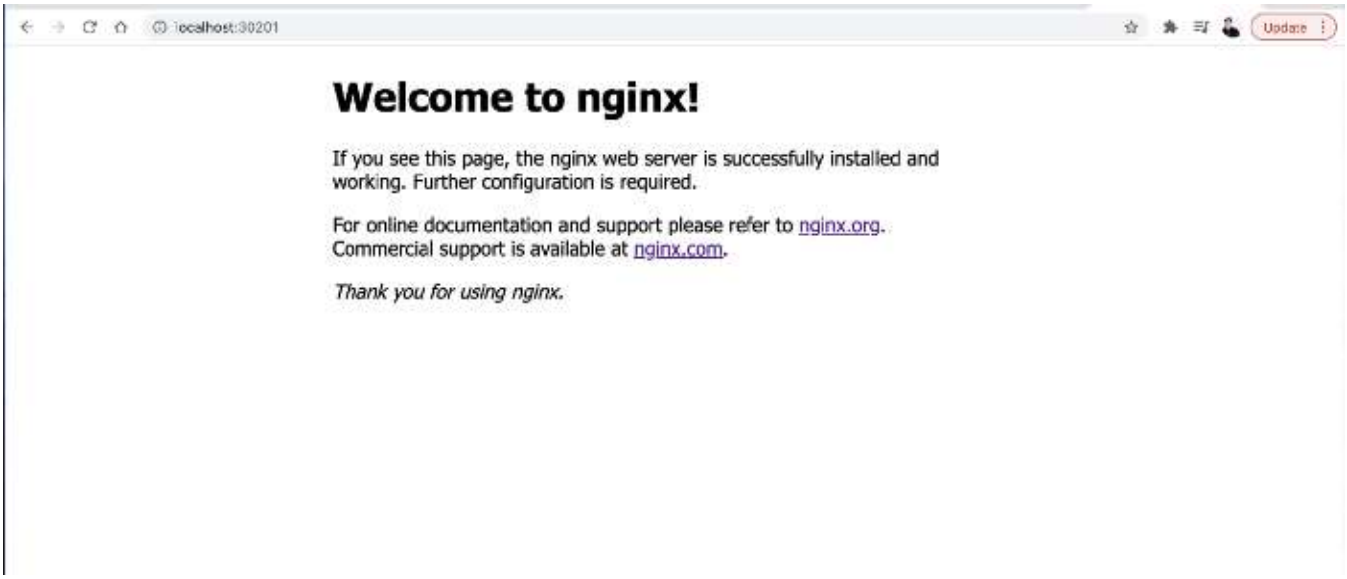
```
72    }
73  }
74
```

Once the apply is complete, verify the NGINX deployment is running.

```
$ kubectl get services
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        12m
nginx-example   NodePort    10.96.233.88    <none>        80:30201/TCP   17s
jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % vim kubernetes.tf
jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % terraform apply
kubernetes_deployment.nginx: Refreshing state... [id=default/scalable-nginx-example]
kubernetes_service.nginx: Refreshing state... [id=default/nginx-example]
```

You can access the NGINX instance by navigating to the NodePort at http://localhost:30201/ .




http://localhost:30201/

## Scale the deployment:

You can scale your deployment by increasing the `replicas` field in your configuration. Change the number of replicas in your Kubernetes deployment from `2` to `4` .

```
1  resource "kubernetes_deployment" "nginx" {
2    #
```

```
    replicas = 4

    # ...
  }

  # ...
}
```

kubernetes.tf hosted with ♡ by GitHub view raw

Apply the change to scale your deployment.

```
$ terraform apply
```

```
Plan: 0 to add, 1 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

kubernetes_deployment.nginx: Modifying... [id=default/scalable-nginx-example]
kubernetes_deployment.nginx: Modifications complete after 0s [id=default/scalable-nginx-example]
```

```
$ kubectl get deployments
```

```
Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes % kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
scalable-nginx-example   4/4     4            4           9m38s
jerryc.quiles@Jerrys-MacBook-Pro learn-terraform-deploy-nginx-kubernetes %
```

## Clean up your workspace

Running `terraform destroy` will de-provision the NGINX deployment

```
$ terraform destroy
```

Thanks for checking this out, hopefully this helped you!

### Contact me:

email → quileswest@gmail.com

Kubernetes Terraform Tech Computer Science DevOps