DevOps With Brian - Linode Terraform

Release 0.1.0

Brian Hopkins

Dec 11, 2022

DOCUMENTATION

This documentation goes over the initial setup of everything required to setup a Linode LKE cluster as code via Terraform. This currently sets up the following after complete:

Linode LKE - Kubernetes Cluster with 3 shared nodes

Linode node_balancer - Using nginx ingress it will use this to allow for public resources if wanted from kubernetes.

Linode Domain - Sets up your own domain in linode to be used for dns resolution in kubernetes as well as for a githubpages if desired.

This also sets up cert-manager in kubernetes and allows for auto cert generation using Let's Encrypt using annocations on our deployments.

CHAPTER

ONE

INITIAL SETUP

This documentation section goes over the initial setup of everything required to setup a Linode LKE cluster as code via Terraform. This currently sets up the following after complete:

```
Linode LKE - Kubernetes Cluster with 3 shared nodes
Linode node_balancer - Using nginx ingress it will use this to allow for public␣
↪resources if wanted from kubernetes.
Linode Domain - Custom domain in linode to be used for dns resolution in kubernetes as␣
↪well as for a githubpages if desired.
```

This also sets up cert-manager in kubernetes and allows for auto cert generation using Let's Encrypt using annocations on our deployments.

There are a few prerequisites if you decide to use this whole project. The following sections go over some of those and how to set them up.

1.1 Commitizen & Conventional Commits

This repo uses Conventional Commits along with Commitizen to allow for auto versioning and such.

Ensure you are using conventional commits for your commit messages and ensure you have installed Commitizen as well from the link provided above.

1.2 Custom Domain

For the entire project the way I am using it you will need your own domain, otherwise you can skip the ingress ssl and domain parts and only use the LKE Terraform.

Once you setup your own domain you are going to want to point it to the Linode nameservers:

```
ns1.linode.com
ns2.linode.com
ns3.linode.com
ns4.linode.com
ns5.linode.com
```

1.3 Terraform

You will also want to ensure you have Terraform Downloads installed.

Also this repo uses remote state located in Terraform Cloud, more information can be found at https://www.hashicorp.com/products/terraform/pricing. Currently using the Free tier and setup an account for free. After setting up a free account you will want to generate an API token as discussed in https://developer.hashicorp.com/terraform/tutorials/cloud-get-started/cloud-login.

1.4 Linode Account

You will also want to sign up for a Linode account if you don't already have one.

This setup if used completely will setup as previously stated above the LKE, Node_Balancer, and Domain setup in Linode.

For more information on Terraform on Linode for LKE please see Deploy LKE Cluster Using Terraform

CHAPTER

TWO

KUBERNETES SETUP

This section outlines how to setup the Linode LKE cluster via Terraform code. All of the steps below will be ran from the `lke` directory.

2.1 Linode API Token

We will need a Linode API token in order for this to work and be able to setup the resources. You can find a how to located at https://www.linode.com/docs/guides/getting-started-with-the-linode-api/

The scopes you need to give it access to are:

```
Domains - Read/Write
Kubernetes - Read/Write
IPs - Read/Write
Linodes - Read/Write
```

This API token will become the `TF_VAR_token` mentioned in the next section.

2.2 Terraform Variables

The first thing we need to do is setup some Terraform variables that we are going to be using.

2.2.1 Sensitive Variables

There are some variables throughout this setup that are sensitive and you don't want to store in your `terraform.tfvars` file, so for these you will do a `export` command to set the variables on your own shell.

As mentioned already in the previous step you will want to from your shell run `export TF_VAR_token=XXX` which is the Linode API token you already setup in the previous section.

This will be the only required variable that is a secret for the `lke` setup, when we setup the domain section next it will require more variables.

The other variables can be found in the `lke/terraform.tfvars` file. These variables below control the cluster label and size of cluster, etc.

2.2.2 Other Variables

There are a few variables in the `lke/terraform.tfvars` file that need to be set in order to ensure your cluster is setup how you want. This section will outline those variables.

Modify these for your needs.

1. The `label` controls what the cluster label will be named.
2. The `k8s_version` tells it what version of kubernetes to use.
3. The `region` tells what location to build the cluster in.
4. The `pools` is a list variable that tells what type of nodes to use and how many. Currently it is setup to use shared nodes, All node types can be found Here

---

Note: Currently the pool setting defined configures the following:

A shared 2GB Linode with 1 vcpu and costs about $10 per node per month for a total cost of $30, plus $10 a month for the load balancer.

2.3 Terraform Cloud Login For Remote State

Before peforming the next steps you will need to login via the cli to the terraform cloud and generate a token to use following the guide at https://developer.hashicorp.com/terraform/tutorials/cloud-get-started/cloud-login

2.4 Terraform Init & Plan

Now that we have our variables all setup and should have Terraform installed now, we can initialize our project and run a plan to verify what it will do.

Make sure you are in the `lke` folder and run the following:

`terraform init` - This will initialize everything needed for the project to run and install modules.

Once this is complete you can now run the plan command to validate you have all your vars setup and it can generate everything properly before applying it:

`terraform plan -var-file="terraform.tfvars"` which will give a output of the information that it will deploy, validate this looks right before continuing.

2.5 Deploy LKE Terraform

As long as we didn't have any issues with the previous Init & Plan step we can now deploy our cluster.

Ensuring we still have our `TF_VAR_token` exported on our shell then we can run:

`terraform apply -var-file="terraform.tfvars"` which should ask for a yes prompt and then will deploy the cluster and will generate your kubeconfig to connect to it.

Note: This step can take a few to complete since it has to spin up nodes and set it all up, so be patient.

2.6 Connecting To New Kubernetes Cluster

After running the terraform apply command to deploy the cluster we should now have a `kube-config` file in our directory, so from this dir we can now run `kubectl` commands to interact with the cluster, export this as seen below.

Add this to the `$KUBECONFIG` env var:

```
export KUBECONFIG=$(pwd)/kube-config
```

If for some reason you ever delete your kube-config or lose it, you can regenerate it via:

```
export KUBE_VAR=`terraform output kubeconfig` && echo $KUBE_VAR | base64 -di > kube-
↪config
export KUBECONFIG=$(pwd)/kube-config
```

This will recreate it and set the path of `KUBECONFIG` to it to be used.

Now we should be able to run `kubectl cluster-info` to get the info from the cluster which confirms we can access it:

```
Kubernetes control plane is running at https://XXXX.us-east-2.linodelke.net:443
KubeDNS is running at https://XXXX.us-east-2.linodelke.net:443/api/v1/namespaces/kube-
↪system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Go ahead and run the following command to get the node_balancer external-ip address which you will need for the next dns steps:

```
kubectl -n default get services -o wide ingress-ingress-nginx-controller
```

This should give us something like:

```
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)
↪                 AGE     SELECTOR
my-ingress-nginx-controller   LoadBalancer   10.128.169.60   192.0.2.0   80:32401/TCP,
↪443:30830/TCP   7h51m   app.kubernetes.io/instance=cingress-nginx,app.kubernetes.io/
↪name=ingress-nginx
```

Let's move on to the `dns` folder and steps.

CHAPTER

THREE

LINODE DNS/DOMAIN SETUP

This section is used to setup the Linode DNS options if you want to connect a custom domain and be able to do SSL for public host Kubernetes apps.

There is also configuration to setup the required A records for my domain which is a var in the TF code to the Github Pages addresses. By doing this I can host my other github pages repo on my custom domain.

Ensure you are in the `dns` folder for these steps.

3.1 Linode API Token

Just as we setup before for lk3, we will need a Linode API token in order for this to work and be able to setup the resources. You can find a how to located at https://www.linode.com/docs/guides/getting-started-with-the-linode-api/

The scopes you need to give it access to are:

```
Domains - Read/Write
Kubernetes - Read/Write
IPs - Read/Write
Linodes - Read/Write
```

This API token will become the `TF_VAR_token` mentioned in the next section.

3.2 Terraform Variables

The first thing we need to do is setup some Terraform variables that we are going to be using.

3.2.1 Sensitive Variables

There are some variables throughout this setup that are sensitive and you don't want to store in your `terraform.tfvars` file, so for these you will do a `export` command to set the variables on your own shell.

1. As mentioned already in the previous step you will want to from your shell run `export TF_VAR_token=XXX` which is the Linode API token you already setup in the previous section.
2. The `export TF_VAR_soa_email=xxx@xxx.com` needs to be ran to export the email that is associated with the domain when registered.
3. The `export TF_VAR_nodebalancer_ip=X.X.X.X` will be the IP of the nodebalancer that was setup in the previous kubernetes step.

3.2.2 Other Variables

There are a few variables in the `dns/terraform.tfvars` file that need to be set in order to ensure your dns/domain is setup how you want. This section will outline those variables.

Modify these for your needs.

1. The `domain_name` variable is the domain name you will be setting up in linode.
2. The `ghost_alias` is used for ghost cname to our custom domain and sets up the required A records for that.

3.3 Terraform Cloud Login For Remote State

Before peforming the next steps you will need to login via the cli to the terraform cloud and generate a token to use following the guide at https://developer.hashicorp.com/terraform/tutorials/cloud-get-started/cloud-login

3.4 Terraform Init & Plan

Now that we have our variables all setup and should have Terraform installed now, we can initialize our project and run a plan to verify what it will do.

Make sure you are in the `dns` folder and run the following:

`terraform init` - This will initialize everything needed for the project to run and install modules.

Once this is complete you can now run the plan command to validate you have all your vars setup and it can generate everything properly before applying it:

`terraform plan -var-file="terraform.tfvars"` which will give a output of the information that it will deploy, validate this looks right before continuing.

3.5 Deploy DNS/Domain Terraform

As long as we didn't have any issues with the previous Init & Plan step we can now deploy our dns & domain changes..

Ensuring we still have our `TF_VAR_token, TF_VAR_soa_email, and TF_VAR_nodebalancer_ip` exported on our shell then we can run:

`terraform apply -var-file="terraform.tfvars"` which should ask for a yes prompt and then will deploy the cluster and will generate your kubeconfig to connect to it.

Now you should see your changes reflected in the Linode UI under domains.

Let us move on to to the `cert-manager` folder and steps for auto SSL certs.

CHAPTER

FOUR

CERT MANAGER SETUP

This section will go over how to get cert manager setup on your kubernetes cluster to allow for automated SSL certificates from Let's Encrypt.

More information can be found at Linode TLS Encryption Guide Kubernetes or at Cert Manager Install

Ensure you are in the `cert-manager` directory for all of these steps.

Note: Be sure before starting any of the below steps you already have your dns nameservers from your custom domain pointing to the linode servers and resolving, this should have been done in the previous dns terraform steps.

4.1 Install Cert Manager CRDs

Ensure you have your *kube-config* file from the previous kubernetes step exported or you can copy to this folder if needed, and then you will want to run the following to install the cert manager CRDs:

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.8.0/
↪cert-manager.crds.yaml
```

We should see something like this:

```
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
```

4.2 Install Cert Manager

Now we can add our cert-manager helm repo and update it then install it:

```
helm repo add cert-manager https://charts.jetstack.io

helm repo update

helm install my-cert-manager cert-manager/cert-manager --namespace cert-manager --
↪version v1.8.0
```

If successful we should see something like this:

```
NAME: my-cert-manager
LAST DEPLOYED: Mon Nov 21 06:39:07 2022
NAMESPACE: cert-manager
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
cert-manager v1.8.0 has been deployed successfully!

In order to begin issuing certificates, you will need to set up a ClusterIssuer
or Issuer resource (for example, by creating a 'letsencrypt-staging' issuer).

More information on the different types of issuers and how to configure them
can be found in our documentation:

https://cert-manager.io/docs/configuration/

For information on how to configure cert-manager to automatically provision
Certificates for Ingress resources, take a look at the `ingress-shim`
documentation:

https://cert-manager.io/docs/usage/ingress/
```

Now verify you see the corresponding pods coming up and running:

```
kubectl get pods --namespace cert-manager
```

You should see something like:

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-579d48dff8-84nw9              1/1     Running   3          1m
cert-manager-cainjector-789955d9b7-jfskr   1/1     Running   3          1m
cert-manager-webhook-64869c4997-hnx6n      1/1     Running   0          1m
```

Note: Before continuing to the next steps ensure these cert-manager pods are running and ready.

4.3 Setting Up ClusterIssuer Resource

Next we will be creating a cluster issuer resource that will be in charge of helping with the automated ssl certs.

This manifest we are about to install will register an account on an ACME server used by Let's Encrypt for the certificates.

To secure the email we have set this up as a export instead of an actual terraform tfvar. So we need to export our email address we want to use with Let's Encrypt to issue the certificates automatically.

Once you know what this email should be run the following:

```
export EMAIL=xxx@xxx.com
envsubst < acme-issuer-prod.yaml | kubectl apply -f -
```

This will update the email section of that file for you automatically and apply it.

We should now have everything we need setup in order to deploy our test application. In this setup we are using a rasa chatbot atm for a demo example. Please proceed to the next Rasa Demo Setup section to see how this works.

CHAPTER

FIVE

RASA DEMO SETUP

Now in order to test everything we need a demo app to deploy, you can use whatever you like but for our setup we are using a Rasa chatbot on a github pages setup.

Everything being performed in this step will be done in the `rasa` directory.

5.1 Deploy Rasa On Kubernetes

So now we want to deploy our previous chatbot model from our last video we made, so in order to do that we setup a helm chart values file to use.

First thing we need to do is add our helm repos:

```
helm repo add rasa https://helm.rasa.com
helm repo update
```

Now we can actually install our Rasa chatbot using the helm install with our values file `rasa.values.yaml`.

There are a few custom things that need to be set in this file however:

```
hostname - This needs to be set to whatever your a record you setup in dns was with your␣
↪custom domain.
hosts - The hosts section under secret needs to be set to the same name as the hostname.
initialModel - This should be pointing to a non authenticated location where your rasa␣
↪model is,
               we are using the model from a previous video we made with a chatbot.
```

Now that we have set our values we can install this into kubernetes:

```
helm install -f rasa-values.yaml rasa rasa/rasa
```

This might take a few mins to come up, but once the pod shows ready you can see the status via going to https://subdomain.yourdomain.com/status

You can check the pod status by running:

```
kubectl get pods
```

And you should see these:

```
NAME                                    READY   STATUS              RESTARTS␣
↪   AGE
rasa-6fb894b7c-vr85l                    0/1     PodInitializing     0          ␣
```

(continues on next page)

(continued from previous page)

```
↪  36s
rasa-postgresql-0                                  0/1     ContainerCreating   0          ␣
↪  35s
```

Once these show running you should be able to hit the resource at the https://subdomain.yourdomain.com/status route.

CHAPTER

SIX

CHANGELOG

6.1 0.1.0 (2022-12-11)

6.1.1 Feat

- **init**: initial files

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search