
terraform-aws-coreos-kubernetes



Opinionated Terraform module for creating a Highly Available Kubernetes cluster running on Container Linux by CoreOS (any channel) in an AWS Virtual Private Cloud VPC. With prerequisites installed `make all` will simply spin up a default cluster; and, since it is based on Terraform, customization is much easier than CloudFormation.

The default configuration includes Kubernetes add-ons: DNS, Dashboard and UI.

tl;dr

```
1 # prereqs
2 $ brew update && brew install awscli cfssl jq kubernetes-cli terraform
3
4 # build artifacts and deploy cluster
5 $ make all
6
7 # nodes
8 $ kubectl get nodes
9
10 # addons
11 $ kubectl get pods --namespace=kube-system
12
13 # verify dns - run after addons have fully loaded
14 $ kubectl exec busybox -- nslookup kubernetes
15
16 # open dashboard
17 $ make dashboard
18
19 # obliterate the cluster and all artifacts
20 $ make clean
```

Component and Tool Versions

component / tool	version
Container Linux by CoreOS	1409.7.0, 1465.3.0, 1492.1.0
kubernetes	1.7.4
flanneld	0.7.1
docker	1.12.6

component / tool	version
etcd	3.1.6
rkt	1.25.0
terraform	0.10.0
cfssl	1.2.0
aws-cli	aws-cli/1.11.129 Python/2.7.10 Darwin/16.7.0 botocore/1.5.92
jq	1.5

Features

- Cluster-internal Certificate Authority infrastructure for TLS certificate generation
- etcd3

AWS

- EC2 Key Pair creation
- AWS VPC Public and Private subnets
- IAM protected S3 bucket for asset distribution
- Bastion Host
- Multi-AZ Auto-Scaling Worker Nodes
- VPC NAT Gateway
- VPC Endpoint for simplified S3 access from EC2 instances

Container Linux by CoreOS

- etcd3 DNS Discovery Bootstrap
- kubelet runs under rkt (using Container Linux by CoreOS recommended Kubelet Wrapper Script)

Kubernetes

- Highly Available ApiServer Configuration
- Service accounts enabled

Terraform

- Container Linux by CoreOS AMI sourcing
- Terraform Pattern Modules

Prerequisites

- AWS Command Line Interface
- CFSSL: CloudFlare's PKI and TLS toolkit
- jq
- kubectl
- Terraform

Quick install prerequisites on Mac OS X with Homebrew:

```
1 $ brew update && brew install awscli cfssl jq kubernetes-cli terraform
```

Launch Cluster

`make all` will create: - AWS Key Pair (PEM file) - AWS VPC with private and public subnets - Route 53 internal zone for VPC - Bastion host - Certificate Authority server - etcd3 cluster bootstrapped from Route 53 - High Availability Kubernetes configuration (masters running on etcd nodes) - Autoscaling worker node group across subnets in selected region - kube-system namespace and addons: DNS, UI, Dashboard

```
1 make all
```

To open dashboard:

```
1 make dashboard
```

To display instance information:

```
1 make instances
```

To display status:

```
1 make status
```

To destroy, remove and generally undo everything:

```
1 make clean
```

`make all` and `make clean` should be idempotent - should an error occur simply try running the command again and things should recover from that point.

How Tack works

Tack Phases

Tack works in three phases:

1. Pre-Terraform
2. Terraform
3. Post-Terraform

Pre-Terraform The purpose of this phase is to prep the environment for Terraform execution. Some tasks are hard or messy to do in Terraform - a little prep work can go a long way here. Determining the Container Linux by CoreOS AMI for a given region, channel and VM Type for instance is easy enough to do with a simple shell script.

Terraform Terraform does the heavy lifting of resource creation and sequencing. Tack uses local modules to partition the work in a logical way. Although it is of course possible to do all of the Terraform work in a single `.tf` file or collection of `.tf` files, it becomes unwieldy quickly and impossible to debug. Breaking the work into local modules makes the flow much easier to follow and provides the basis for composing variable solutions down the track - for example converting the worker Auto Scaling Group to use spot instances.

Post-Terraform Once the infrastructure has been configured and instantiated it will take some time for it to settle. Waiting for the 'master' ELB to become healthy is an example of this.

Components

Like many great tools, *tack* has started out as a collection of scripts, makefiles and other tools. As *tack* matures and patterns crystalize it will evolve to a Terraform plugin and perhaps a Go-based cli tool for 'init-ing' new cluster configurations. The tooling will compose Terraform modules into a solution based on user preferences - think `npm init` or better yet yeoman.

TLS Certificates

- etcd3 coreos cloudint

```
1 curl --cacert /etc/kubernetes/ssl/ca.pem --cert /etc/kubernetes/ssl/k8s-  
  -etcd.pem --key /etc/kubernetes/ssl/k8s-etcd-key.pem https://etcd.  
  test.kz8s:2379/health  
2 openssl x509 -text -noout -in /etc/kubernetes/ssl/ca.pem  
3 openssl x509 -text -noout -in /etc/kubernetes/ssl/k8s-etcd.pem
```

ElasticSearch and Kibana To access Elasticseach and Kibana first start `kubectl proxy`.

```
1 $ kubectl proxy  
2 Starting to serve on localhost:8001
```

- `http://localhost:8001/api/v1/proxy/namespaces/kube-system/services/elasticsearch-logging`
- `http://localhost:8001/api/v1/proxy/namespaces/kube-system/services/kibana-logging/app/kibana`

FAQs

- Create an etcd cluster with more than 3 instances

Advanced Features and Configuration

Using an Existing VPC

If you have an existing VPC you'd like to deploy a cluster into, there is an option for this with *tack*.

Constraints

- You will need to allocate 3 static IPs for the etcd servers - Choose 3 unused IPs that fall within the IP range of the first subnet specified in `subnet-ids-private` under `vpc-existing.tfvars`
- Your VPC has to have private and public subnets (for now)
- You will need to know the following information:
 - VPC CIDR Range (e.g. 192.168.0.0/16)
 - VPC Id (e.g. vpc-abc123)
 - VPC Internet Gateway Id (e.g. igw-123bbd)
 - VPC Public Subnet Ids (e.g. subnet-xyz123,subnet-zyx123)
 - VPC Private Subnet Ids (e.g. subnet-lmn123,subnet-opq123)

Enabling Existing VPC Support

- Edit `vpc-existing.tfvars`
 - Uncomment the blocks with variables and fill in the missing information
- Edit `modules_override.tf` - This uses the overrides feature from Terraform
 - Uncomment the `vpc` module, this will override the reference to the regular VPC module and instead use the stub `vpc-existing` module which just pulls in the variables from `vpc-existing.tfvars`
- Edit the Makefile as necessary for `CIDR_PODS`, `CIDR_SERVICE_CLUSTER`, etc to match what you need (e.g. avoid collisions with existing IP ranges in your VPC or extended infrastructure)

Testing Existing VPC Support from Scratch In order to test existing VPC support, we need to generate a VPC and then try the overrides with it. After that we can clean it all up. These instructions are meant for someone wanting to ensure that the *tack* existing VPC code works properly. * Run `make all` to generate a VPC with Terraform * Edit `terraform.tfstate` * Search for the VPC block and cut it out and save it somewhere. Look for `"path": ["root","vpc"]` * Run `make clean` to remove everything but the VPC and associated networking (we preserved it in the previous step) * Edit as per instructions above * Run `make all` to test out using an existing VPC * Cleaning up: * Re-insert the VPC block into `terraform.tfstate` * Run `make clean` to clean up everything

Additional Configuration

- You should to tag your subnets for internal/external load balancers

Inspiration

- Code examples to create Container Linux by CoreOS cluster on AWS with Terraform by xuwang
- `kaws`: tool for deploying multiple Kubernetes clusters
- Kubernetes on Container Linux by CoreOS
- Terraform Infrastructure Design Patterns by Bart Spaans
- The infrastructure that runs Brandform
- AutoScaling your Kubernetes cluster on AWS
- Bash template substitution for manifests - from kayrus/elk-kubernetes

Other Terraform Projects

- bakins/kubernetes-coreos-terraform
- bobtfish/terraform-aws-coreos-kubernates-cluster
- chiefly/tf-aws-kubernetes
- cihangir/terraform-aws-kubernetes
- ericandrewlewis/kubernetes-via-terraform
- funkymonkeymonk/terraform-demo
- kelseyhightower/kubestack
- samsung-cnct/kraken
- wearemakery/kubestack-aws
- xuwang/aws-terraform

References

- CFSSL: CloudFlare's PKI and TLS toolkit
- Container Linux by CoreOS - Mounting Storage
- Deploying Container Linux by CoreOS cluster with etcd secured by TLS/SSL
- etcd dns discovery bootstrap
- Generate EC2 Key Pair
- Generate self-signed certificates
- kubectl Cheat Sheet
- Makefile [help](#) target
- Peeking under the hood of Kubernetes on AWS
- Self documenting Makefile
- Setting up etcd to run in production
- ssl artifact generation
- Cluster Autoscaler
- Persistent Storage - Kubernetes on AWS
- VPC endpoint Terraform example setup