1. Terraform creates everything related to VPC
2. Kops requests the resources created by Terrafrom
3. Then Kops creates k8s cluster and other k8s related resources on top of existing infra

Let’s define the task. You have AWS account and you need to create AWS infrastructure (VPC) and K8s cluster on top of it.

So let’s do it.

Install required tools

[aws-cli](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html), [Terraform](https://www.terraform.io/downloads.html), [Kops](https://github.com/kubernetes/kops/blob/master/docs/install.md), [jq](https://stedolan.github.io/jq/download/).

Configure AWS cli

If you don’t have configured AWS credentials see [how to get Access Keys](https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html)

And then execute:

*aws* configure --profile my-profile

It will do a small quiz:

AWS Access Key ID [None]: AKIAIOS5ILYU233RPKEA  
AWS Secret Access Key [None]: HIDDEN  
Default region name [None]: us-east-1  
Default output format [None]:

Then in order to select profile “my-profile” execute:

export AWS\_PROFILE=my-profile

## Domain and project name

export DOMAIN=your.domainexport PROJECT\_NAME=dev

In this case you can use some fake-domain.com as your domain. We are not going to use real domain name but Kops requires route53 hozted zone.

## The state files

Both Terraform and Kops need a back-end to store their state. I will use s3 buckets for both.

aws s3 mb s3://terraform-state.${DOMAIN}  
aws s3 mb s3://kops-state.${DOMAIN}

The bucket names need to be unique across whole existing AWS accounts. So I accomplished it by adding my domain as a suffix. You can name them whatever currently available.

## Simple Terraform config example

Create a directory that will contain terraform configuration. The directory may be later stored in a git repository.

mkdir terraform-kops-example **&&** cd terraform-kops-example

Create file main.tf with the content bellow:

**terraform** {  
 **backend** “s3” {}  
  
**provider** “aws” {}  
**variable** “vpc\_cidr\_block” {default = “10.0.0.0/16”}  
**variable** “project\_name” {}  
**variable** “domain” {}  
**variable** “networks” {  
 type = map(object({  
 cidr\_block = string  
 availability\_zone = string  
 }))  
 default = {  
 n0 = {  
 cidr\_block = “10.0.0.0/24”  
 availability\_zone = “us-east-1a”  
 }  
 n1 = {  
 cidr\_block = “10.0.1.0/24”  
 availability\_zone = “us-east-1b”  
 }  
 n2 = {  
 cidr\_block = “10.0.2.0/24”  
 availability\_zone = “us-east-1c”  
 }  
 }  
}  
  
resource “aws\_vpc” “vpc” {  
 cidr\_block = var.vpc\_cidr\_block  
 tags = “${  
 map(  
 “Name”, “${var.project\_name}”  
 )  
 }”  
}#vpc set deliberately as we don’t have real domain  
#in order to create public zone - remove vpc section  
resource “aws\_route53\_zone” “domain” {  
 name = “${var.project\_name}.${var.domain}”  
 vpc {  
 vpc\_id = “${aws\_vpc.vpc.id}”  
 }  
}  
  
data “aws\_availability\_zones” “available” {}  
  
resource “aws\_subnet” “subnets” {  
 count = “${length(var.networks)}”  
 availability\_zone = “${var.networks[“n${count.index}”].availability\_zone}”  
 cidr\_block = “${var.networks[“n${count.index}”].cidr\_block}”  
 vpc\_id = “${aws\_vpc.vpc.id}”  
 tags = “${  
 map(  
 “Name”, “${var.project\_name}”  
 )  
 }”  
}  
  
resource “aws\_internet\_gateway” “internet\_gateway” {  
 vpc\_id = “${aws\_vpc.vpc.id}”  
}  
  
resource “aws\_route\_table” “0route\_table” {  
 vpc\_id = “${aws\_vpc.vpc.id}”  
 tags = {  
 Name = “${var.project\_name}”  
 }  
 route {  
 cidr\_block = “0.0.0.0/0”  
 gateway\_id = “${aws\_internet\_gateway.internet\_gateway.id}”  
 }  
}  
  
resource “aws\_route\_table\_association” “route\_table\_association” {  
 count = “${length(var.networks)}”  
 subnet\_id = “${aws\_subnet.subnets.\*.id[count.index]}”  
 route\_table\_id = “${aws\_route\_table.route\_table.id}”  
}  
  
output “vpc\_id” {  
 value = “${aws\_vpc.vpc.id}”  
}  
  
data “aws\_subnet\_ids” “subnet\_ids” {  
 depends\_on = [  
 aws\_subnet.subnets  
 ]  
 vpc\_id = aws\_vpc.vpc.id  
}  
  
output “subnet\_ids” {  
 value = data.aws\_subnet\_ids.subnet\_ids.ids.\*  
}  
  
output “networks” {  
 value = var.networks  
}

## Terraform init

Init Terraform with specifying where it should store the state:

terraform init \  
-backend-config “bucket=terraform-state.${DOMAIN}” \  
-backend-config “key=file.state”

Create new workspace where state of the configuration will be stored:

terraform workspace new ${PROJECT\_NAME}

Or select workspace if already created:

terraform workspace **select** ${PROJECT\_NAME}

We might need this workspaces functionality as it would allow you to keep multiple states of multiple project or environments in the same bucket.

Your Terraform is ready for applying, so let’s do it

## Terraform apply

terraform apply -var “project\_name=${PROJECT\_NAME}” -var “domain=${DOMAIN}”

Your VPC has been created and ready to go. Let’s create k8s cluster on top of it with Kops.

## Configure Kops state

export KOPS\_STATE\_STORE=s3://kops-state.${DOMAIN}

## Kops create cluster

This cmd will only prepare configuration of the cluster and store it in the s3 bucket we specified via KOPS\_STATE\_STORE env variable.

kops create cluster \  
--vpc=**$**(terraform output vpc\_id) \  
--master-zones=**$**(terraform output -json networks | jq -r '.[].availability\_zone' | paste -sd, -) \  
--zones=**$**(terraform output -json networks | jq -r '.[].availability\_zone' | paste -sd, -) \  
--subnets=**$**(terraform output -json subnet\_ids | jq -r 'join(“,”)') \  
--networking=calico \  
--node-count=3 \  
--master-size=t2.medium \  
--node-size=t2.medium \  
--dns-zone=${PROJECT\_NAME}.${DOMAIN} \  
--dns=private \  
--name=${PROJECT\_NAME}.${DOMAIN}

“dns=private” is set in order to disable DNS resolution verification as we don’t have real doamin name.

As you can see vpc, zones, master-zones, subnets comes from terraform output because VPC and Subnets already exist.

Also we don’t need to set “target=terraform” flag that doesn’t make sense as it would create additional Terraform configuration that would require additional “Terraform apply” and state and so forth.

## Review Kops clsuter configuration (optional)

kops edit ckuster --name ${PROJECT\_NAME}.${DOMAIN}

## Apply Kops configuration

This will deploy k8s cluster:

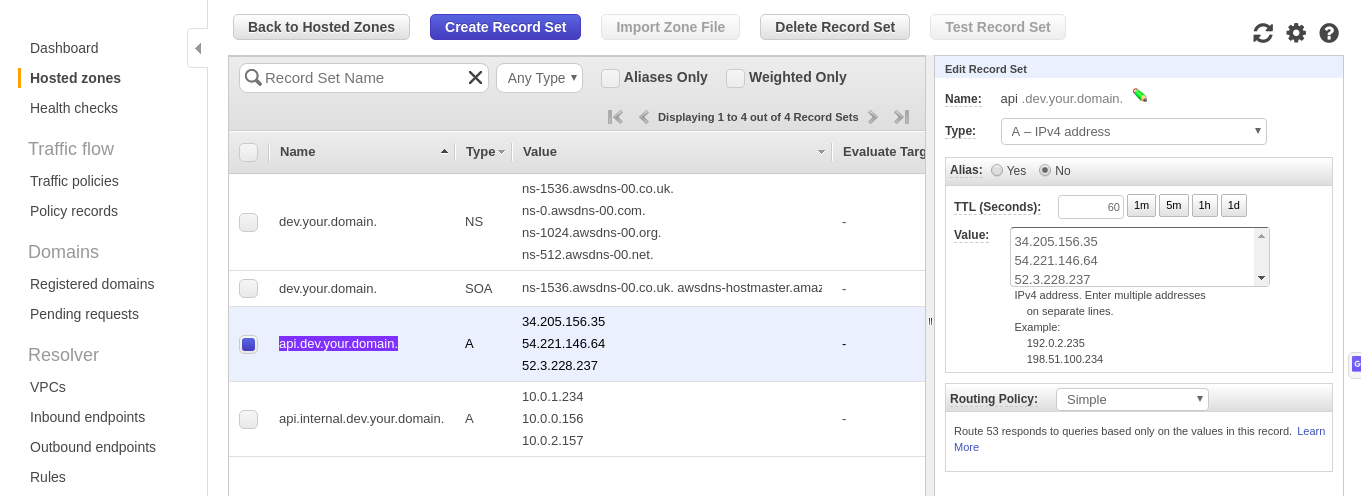
kops update cluster --name ${PROJECT\_NAME}.${DOMAIN} --yes

Check the cluster (will fail):

kubectl cluster-infoTo further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.  
Unable to connect to the server: dial tcp: lookup api.dev.your.domain on 8.8.8.8:53: no such host

The error arose because the hosted zone that contains “api.dev.your.domain” DNS record is private and the domain name cannot be resolved.

In order to fix it get api endpoint ip from route53 in AWS console:



Fix it via hosts file:

Sudo bash -c ‘echo “34.205.156.35 api.dev.your.domain” >> /etc/hosts’

Check cluster again:

kubectl cluster-info  
Kubernetes master is running at [https://api.dev.your.domain](https://api.dev.your.domain/)  
KubeDNS is running at <https://api.dev.your.domain/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy>To further debug and diagnose cluster problems, use ‘kubectl cluster-info dump’.

Check the nodes:

kubectl get nodes  
NAME STATUS ROLES AGE VERSION  
ip-10-0-0-156.ec2.internal Ready master 14m v1.12.8  
ip-10-0-1-234.ec2.internal Ready master 14m v1.12.8  
ip-10-0-2-157.ec2.internal Ready master 14m v1.12.8

Done!

## Cleanup

kops delete cluster --name ${PROJECT\_NAME}.${DOMAIN} --yes  
terraform destroy -var “project\_name=${PROJECT\_NAME}” -var “domain=${DOMAIN}”  
aws s3 rb s3://terraform-state.${DOMAIN} --force  
aws s3 rb s3://kops-state.${DOMAIN} --force

[AWS](https://medium.com/tag/aws)

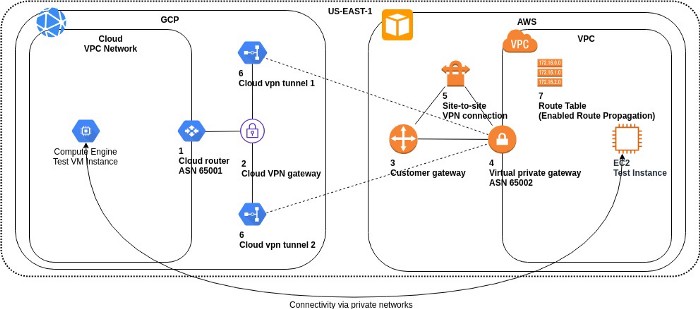
* [Kops](https://medium.com/tag/kops)
* [Terraform](https://medium.com/tag/terraform)
* [Kubernetes](https://medium.com/tag/kubernetes)
* [K8s](https://medium.com/tag/k8s)

A walk-through for configuring secure redundant connectivity between AWS VPCs and GCP Cloud VPC Networks with dynamic BGP routing.

It doesn’t cover creating custom VPCs as there is a lot of guides in the Internet.

# Use-cases:

1. Secured and united infrastructure distributed between 2 cloud providers
2. Secured migration from one cloud provider to another
3. Simulating on-premise VPN connections
4. Getting rid of manual routes configuration
5. etc…



Schema

The numbers represent the order in which AWS and GCP services need to be created.

Let’s assume you have default GCP Project and AWS Account with default VPC.

Let’s also describe the task: 2 virtual machines, one in GCP and one in AWS should be able to say hello to each other without using public IP addresses.