Terraform files explanation

Terraform files and explanation

The first three files have been pre-created from the gen-backend.sh script in the tf-setup stage.

In this example we use a local state files for the Load Balancer creation as an alternative example, but you could setup a remote state file on S3 as we have done in the infrastructure build stages.

aws.tf

This specifies the location of the backend Terraform state file which in this case is locally stored in the current directory.

```
terraform {
 required_version = "~> 0.15.3"
 required_providers {
  aws = {
  source = "hashicorp/aws"
  #  Allow any 3.22+  version of the AWS provider
  version = "~> 3.22"
  }
  null = {
  source = "hashicorp/null"
  version = "~> 3.0"
  }
  external = {
  source = "hashicorp/external"
  version = "~> 2.0"
  }

 }
}
```

```
provider "aws" {
 region                = var.region
 shared_credentials_file = "~/.aws/credentials"
 profile               = var.profile
}
provider "null" {}
provider "external" {}
```

vars-main.tf

This file defines some variables with default values for the five dynamoDB tables, the region and default profile name

```
# TF_VAR_region
variable "region" {
 description = "The name of the AWS Region"
 type       = string
 default    = "eu-west-1"
}

variable "profile" {
 description = "The name of the AWS profile in the credentials file"
 type       = string
 default    = "default"
}

variable "cluster-name" {
 description = "The name of the EKS Cluster"
 type       = string
 default    = "mycluster1"
}

variable "stages" {
type=list(string)
default=["net","iam","c9net","cluster","nodeg","cicd","eks-cidr"]
}
```

```
variable "stagecount" {
type=number
default=7
}
```

remote-cluster.tf

Remote access to the Terraform state file for the EKS cluster build

```
data terraform_remote_state "cluster" {
backend = "s3"
config = {
bucket = "terraform-state-f8ffc212119c-1604689183n"
region = "eu-west-1"
key = "terraform/at-terraform-eks-workshop1-cluster.tfstate"
}
}
```

data-eks-cluster.tf

Populate the EKS cluster data, note as we've seen before it uses the remote output from the cluster build **name = data.terraform_remote_state.cluster.outputs.cluster-name**

```
data "aws_eks_cluster" "eks_cluster" {
  name = data.terraform_remote_state.cluster.outputs.cluster-name
}

output "endpoint" {
  value = data.aws_eks_cluster.eks_cluster.endpoint
}

output "ca" {
  value = data.aws_eks_cluster.eks_cluster.certificate_authority[0].data
}
```

```
# Only available on Kubernetes version 1.13 and 1.14 clusters created or upgraded on or after September 3, 2019.
output "identity-oidc-issuer" {
  value = data.aws_eks_cluster.eks_cluster.identity[0].oidc[0].issuer
}

output "cluster-name" {
  value = data.aws_eks_cluster.eks_cluster.name
}
```

null_policy.tf

This null policy simply pulls down a local copy the required policy file for the AWS Load Balancer policy definition.

```
resource "null_resource" "policy" {
triggers = {
   always_run = timestamp()
}
provisioner "local-exec" {
   on_failure  = fail
   when = create
   interpreter = ["/bin/bash", "-c"]
   command     = <<EOT
        curl -o iam-policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/main/docs/install/iam_policy.json
    EOT
}
}
```

aws_iam_policy-lb2.tf

Note this **depends_on** the previous null_resource "policy" - which downloads the iam-policy.json file.

```
resource "aws_iam_policy" "load-balancer-policy" {
  depends_on  = [null_resource.policy]
  name        = "AWSLoadBalancerControllerIAMPolicy"
  path        = "/"
  description = "AWS LoadBalancer Controller IAM Policy"

  policy = file("iam-policy.json")

}
```

null_post_policy.tf

This null provisioner starts the **post-policy.sh** script after the load balancer policy has been created **depends_on=[aws_iam_policy.load-balancer-policy]**

```
resource "null_resource" "post-policy" {
depends_on=[aws_iam_policy.load-balancer-policy]
triggers = {
   always_run = timestamp()
}
provisioner "local-exec" {
   on_failure  = fail
   interpreter = ["/bin/bash", "-c"]
   when = create
   command     = <<EOT
      reg=$(echo ${data.aws_eks_cluster.eks_cluster.arn} | cut -f4 -d':')
      acc=$(echo ${data.aws_eks_cluster.eks_cluster.arn} | cut -f5 -d':')
      cn=$(echo ${data.aws_eks_cluster.eks_cluster.name})
      echo "$reg $cn $acc"
      ./post-policy.sh $reg $cn $acc
      echo "done"
    EOT
}
}
```

post-policy.sh

This script:

- Installs the Load Balancer controllers Custom Resource Definition (CRD).
- Installs helm chart for the aws-load-balancer-controller.

Using helm makes this a lot simpler as the chart does several necessary steps for us:

- As this controller is being invoked in a private VPC we have to specify access to the dependant image via a local VPC endpoint: **--set image.repository=602401143452.dkr.ecr.$1.amazonaws.com/amazon/aws-load-balancer-controller** The regional aspect of the endpoint is passed via a variable **$1**.
- The helm chart also creates a necessary service account **--set serviceAccount.name=aws-load-balancer-controller**.
- The chart also deals with TLS certificate prerequisites transparently for us.

```
test -n "$1" && echo REGION is "$1" || "echo REGION is not set && exit"
test -n "$2" && echo CLUSTER is "$2" || "echo CLUSTER is not set && exit"
test -n "$3" && echo ACCOUNT is "$3" || "echo ACCOUNT is not set && exit"
test -n "$LBC_VERSION" && echo LBC_VERSION is "$LBC_VERSION" || "export LBC_VERSION=2.1.0"
helm repo add eks https://aws.github.io/eks-charts
rm -f crds.yaml*
wget https://raw.githubusercontent.com/aws/eks-charts/master/stable/aws-load-balancer-controller/crds/crds.yaml
# create the custom resource definition for the load balancer
kubectl apply -f crds.yaml
# install the load balancer controller using the helm chart
helm upgrade -i aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=$2 --set serviceAccount.name=aws-load-balancer-controller --set image.repository=602401143452.dkr.ecr.$1.amazonaws.com/amazon/aws-load-balancer-controller --set image.tag="v2.1.0"
```
