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, The S3 bucket and DynamoDB tables were also pre-created in the tf-setup stage.

backend-c9net.tf

This specifies the location of the backend Terraform state file on S3 and the dynamoDB table used for the state file locking. Also the Terraform version requirements.

```
terraform {
required_version = "~> 0.15.3"
required_providers {
 aws = {
  source = "hashicorp/aws"
#  Allow any 3.1x version of the AWS provider
  version = "~> 3.22"
 }
}
backend "s3" {
bucket = "tf-state-ip-172-31-3-99-16105348315920224 27"
key = "terraform/terraform_locks_c9net.tfstate"
region = "eu-west-1"
dynamodb_table = "terraform_locks_c9net"
encrypt = "true"
}
}
provider "aws" {
region = var.region
shared_credentials_file = "~/.aws/credentials"
profile = var.profile
}
```

vars-dynamodb.tf & vars-main.tf

As described in previous sections.

---

remote-net.tf

This defines the location of a remote state file for the network (net) stage of the build. Crucially allows us to read any defined output variables that were created as part of this isolated Terraform build activity.

You'll see these output variables used in this sections Terraform files as vales that begin with: **data.terraform_remote_state.net.outputs.**

Using remote state allows separation of duties when building infrastructure as resources can be referenced in a "read only" fashion by using remote_state variables and Terraform data resources.

```
data terraform_remote_state "net" {
backend = "s3"
config = {
bucket = "terraform-state-ip-172-31-2-146"
region = "eu-west-1"
key = "terraform/terraform_locks_net.tfstate"
}
}
```

Next we gather some existing resources using Terraform data resources.

Find the default & cicd VPC's and make it available (read only) as a Terraform *data* resource

data-defvpc.tf & data-cicdvpc.tf

```
data "aws_vpc" "vpc-default" {
```

```
 default = true
}
data "aws_vpc" "vpc-cicd" {
 default = false
 filter {
  name   = "tag:workshop"
  values = ["eks-cicd"]
 }
}
```

---

data-sg-c9-instance.tf

Find the Cloud9 Instance using the tag pair we created it with - "workshop" and value "eks-terraform".

The instance is then used to output the Cloud9 IDE's security group and the role associated with the instance profile as output variables.

If you have difficulty in following this - have a look at the command:

```
terraform state show data.aws_instance.c9inst
```

And look at the data structure that Terraform populates for this resource.

```
variable "c9label" {
description="Cloud9 IDE Name Label"
type=string
default="eks-terraform"
}

output c9lab {
  value = "%{ if var.c9label != "" } true %{else} false %{endif}"
}


data "aws_instance" "c9inst" {
 filter {
```

```
   name   = "tag:workshop"
   values = ["*${var.c9label}*"]
  }
}

data "aws_security_group" "c9sg" {
  name = sort(data.aws_instance.c9inst.security_groups)[0]
}


data "aws_iam_instance_profile" "c9ip" {
  name = data.aws_instance.c9inst.iam_instance_profile
}

output c9role {
value=data.aws_iam_instance_profile.c9ip.role_arn
}
```

Locate the CICD security group using it's VPC id and specific tag key/value pair.

data-sg-cicd.tf

```
data "aws_security_group" "cicd-sg" {
  vpc_id=data.aws_vpc.vpc-cicd.id
  filter {
    name   = "tag:workshop"
    values = ["eks-cicd"]
  }
}
```

Locate the cicd route table using it's name tag

data-rtb-cicd.tf

```
data "aws_route_table" "cicd-rtb" {
```

```
 vpc_id=data.aws_vpc.vpc-cicd.id
 filter {
  name   = "tag:Name"
  values = ["rtb-eks-cicd-priv1"]
 }
}
```

Next add to the default VPC's route table a route to the EKS private CIDR block using the peering connection id as the destination.

def-route-add.tf

```
resource "aws_route" "rt-def" {
 route_table_id         = data.aws_vpc.vpc-default.main_route_table_id
 destination_cidr_block    = data.terraform_remote_state.net.outputs.eks-cidr
 vpc_peering_connection_id = aws_vpc_peering_connection.def-peer.id
}
```

Next add to the EKS's route tables a route to the Default VPC's CIDR block using the peering connection id as the destination. Note within the use of the previously obtained data values: **data.aws_vpc.vpc-default.cidr_block** and the remote_state value **data.terraform_remote_state.net.outputs.rtb-priv1** for the EKS VPC's routing table.

eks-route-add.tf

```
resource "aws_route" "rt-eks1" {
 route_table_id         = data.terraform_remote_state.net.outputs.rtb-priv1
 destination_cidr_block    = data.aws_vpc.vpc-default.cidr_block
 vpc_peering_connection_id = aws_vpc_peering_connection.def-peer.id
}

resource "aws_route" "rt-eks2" {
 route_table_id         = data.terraform_remote_state.net.outputs.rtb-priv2
 destination_cidr_block    = data.aws_vpc.vpc-default.cidr_block
 vpc_peering_connection_id = aws_vpc_peering_connection.def-peer.id
```

```
}

resource "aws_route" "rt-eks3" {
 route_table_id          = data.terraform_remote_state.net.outputs.rtb-priv3
 destination_cidr_block    = data.aws_vpc.vpc-default.cidr_block
 vpc_peering_connection_id = aws_vpc_peering_connection.def-peer.id
}
```

The file **cicd-route-add.tf** similarly adds routes to the cicd router table

---

For the EKS cluster security group: Add rules to allow traffic from the Cloud9 IDE's CIDR range for ports 443. We also specific a default egress rule.

And for the all nodes security groups: Add rules to allow traffic from the Cloud9 IDE's CIDR range for ports 22. We also specific a default egress rule.

sg-rule-eks.tf

```
resource "aws_security_group_rule" "eks-all" {
 type            = "ingress"
 from_port       = 443
 to_port         = 443
 protocol        = "tcp"
 cidr_blocks     = [data.aws_vpc.vpc-default.cidr_block]
 security_group_id = data.terraform_remote_state.net.outputs.cluster-sg
}

resource "aws_security_group_rule" "eks-all-egress" {
 type            = "egress"
 from_port       = 0
 to_port         = 0
 protocol        = "-1"
 security_group_id = data.terraform_remote_state.net.outputs.cluster-sg
 cidr_blocks     = ["0.0.0.0/0"]
```

```
}


resource "aws_security_group_rule" "eks-node" {
  type            = "ingress"
  from_port       = 22
  to_port         = 22
  protocol        = "tcp"
  cidr_blocks     = [data.aws_vpc.vpc-default.cidr_block]
  security_group_id = data.terraform_remote_state.net.outputs.allnodes-sg
}


resource "aws_security_group_rule" "eks-node-egress" {
  type            = "egress"
  from_port       = 0
  to_port         = 0
  protocol        = "-1"
  security_group_id = data.terraform_remote_state.net.outputs.allnodes-sg
  cidr_blocks     = ["0.0.0.0/0"]
}
```

Next we add some rules to the Cloud9 IDE Security Group that allows incoming traffic from the EKS worker nodes CIDR blocks for port 22 and 443.

sg-rule-def.tf

```
resource "aws_security_group_rule" "sg-def-22" {
  type            = "ingress"
  from_port       = 443
  to_port         = 443
  protocol        = "tcp"
  cidr_blocks     = [data.terraform_remote_state.net.outputs.eks-cidr]
  security_group_id = data.aws_security_group.c9sg.id
}
```

```
resource "aws_security_group_rule" "sg-def-eks-all" {
  type            = "ingress"
  from_port       = 22
  to_port         = 22
  protocol        = "tcp"
  cidr_blocks     = [data.terraform_remote_state.net.outputs.eks-cidr]
  security_group_id = data.aws_security_group.c9sg.id
}
```

There are other files in this sub directory that perform similar operations to the above. They should be easy to understand having studied the examples above.