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 and have been explained in previous sections.

backend-iam.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, the AWS region and profile from variables, and the AWS credentials from a local file if present.

```
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 = "terraform-state-ip-172-31-2-146"
key = "terraform/terraform_locks_iam.tfstate"
region = "eu-west-1"
dynamodb_table = "terraform_locks_iam"
encrypt = "true"
}
}
provider "aws" {
region = var.region
shared_credentials_file = "~/.aws/credentials"
profile = var.profile
}
```

vars-dynamodb.tf

This file defines some variables with default values for the 7x dynamoDB tables, the region and default profile name

```
variable "table_name_net" {
  description = "The name of the DynamoDB table. Must be unique in this AWS account."
  type        = string
  default     = "at-terraform-eks-workshop1-net"
}
** OUTPUT TRUNCATED FOR BREVITY **
```

vars-main.tf

Same as previous sections

EKS requires a few roles to be pre-defined to operate correctly

The cluster service role, an output is defined **cluster_service_role_arn** which is used in the EKS cluster creation in a later stage.

aws_iam_role__cluster-ServiceRole.tf

```
# File generated by aws2tf see https://github.com/aws-samples/aws2tf

resource "aws_iam_role" "eks-cluster-ServiceRole-HUIGIC7K7HNJ" {
  assume_role_policy = jsonencode(
    {
      Statement = [
        {
```

```
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
         Service = [
          "eks-fargate-pods.amazonaws.com",
          "eks.amazonaws.com",
         ]
        }
       },
      ]
      Version = "2012-10-17"
    }
  )
  force_detach_policies = false
  max_session_duration  = 3600
  name              = "eks-cluster-ServiceRole-HUIGIC7K7HNJ"
  path              = "/"
  tags = {
   "Name"  = "eks-cluster/ServiceRole"

  }
}

output "cluster_service_role_arn" {
  value = aws_iam_role.eks-cluster-ServiceRole-HUIGIC7K7HNJ.arn
}
```

And a managed nodegroup role, an output is defined **nodegroup_role_arn** which is also used in the EKS cluster creation in a later stage.

aws_iam_role__nodegroup-NodeInstanceRole.tf

```
# File generated by aws2tf see https://github.com/aws-samples/aws2tf
# aws_iam_role.eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO:
resource "aws_iam_role" "eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO" {
  assume_role_policy = jsonencode(
  {
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      },
    ]
    Version = "2012-10-17"
  }
 )
 force_detach_policies = false
 max_session_duration  = 3600
 name                 = "eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO"
 path                = "/"
 tags = {
   "Name"                              = "eks-nodegroup-ng-maneksami2/NodeInstanceRole"
 }
}

output "nodegroup_role_arn" {
 value = aws_iam_role.eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO.arn
}
```

To these two roles, various policies are then defined and later on attached to the Role.

One example of a policy that is defined is shown here:

aws_iam_role_policy__cluster-ServiceRole__PolicyCloudWatchMetrics.tf

```
# File generated by aws2tf see https://github.com/aws-samples/aws2tf
resource "aws_iam_role_policy" "eks-cluster-ServiceRole-HUIGIC7K7HNJ__eks-cluster-
PolicyCloudWatchMetrics" {
 name = "eks-cluster-PolicyCloudWatchMetrics"
 policy = jsonencode(
  {
   Statement = [
    {
     Action = [
      "cloudwatch:PutMetricData",
     ]
     Effect   = "Allow"
     Resource = "*"
    },
   ]
   Version = "2012-10-17"
  }
 )
 role = aws_iam_role.eks-cluster-ServiceRole-HUIGIC7K7HNJ.id
}
```

And one example of a policy for the nodegroup role:

aws_iam_role_policy__nodegroup-NodeInstanceRole-PolicyAutoScaling.tf

One example is shown here:

```
resource "aws_iam_role_policy" "eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO__eks-nodegroup-ng-maneksami2-PolicyAutoScaling" {
 name = "eks-nodegroup-ng-maneksami2-PolicyAutoScaling"
 policy = jsonencode(
  {
   Statement = [
    {
     Action = [
      "autoscaling:DescribeAutoScalingGroups",
      "autoscaling:DescribeAutoScalingInstances",
      "autoscaling:DescribeLaunchConfigurations",
      "autoscaling:DescribeTags",
          "autoscaling:SetDesiredCapacity",
          "autoscaling:TerminateInstanceInAutoScalingGroup",
          "ec2:DescribeLaunchTemplateVersions",
        ]
        Effect   = "Allow"
        Resource = "*"
      },
    ]
    Version = "2012-10-17"
  }
 )
 role = aws_iam_role.eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO.id
}
```

And finally Policy attachments are specified, one example of each are shown below:

aws_iam_role_policy_attachment__cluster-ServiceRole-AmazonEKSClusterPolicy.tf

```
# File generated by aws2tf see https://github.com/aws-samples/aws2tf
resource "aws_iam_role_policy_attachment" "eks-cluster-ServiceRole-HUIGIC7K7HNJ__AmazonEKSClusterPolicy" {
 policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
 role       = aws_iam_role.eks-cluster-ServiceRole-HUIGIC7K7HNJ.id
}
```

---

aws_iam_role_policy_attachment__nodegroup-NodeInstanceRole-AmazonEC2ContainerRegistryReadOnly.tf

```
# File generated by aws2tf see https://github.com/aws-samples/aws2tf
resource "aws_iam_role_policy_attachment" "eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO__AmazonEC2ContainerRegistryReadOnly" {
 policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
 role       = aws_iam_role.eks-nodegroup-ng-ma-NodeInstanceRole-1GFKA1037E1XO.id
}
```