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-cicd.tf & vars-main.tf

As described in previous sections

Data resources - Read only references we need to build the infrastructure

data-cicdvpc.tf

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

data_subnet_cicd.tf

```
data "aws_subnet" "cicd" {

 filter {
  name   = "tag:workshop"
  values = ["cicd-private1"]
 }
}
```

data-sg-cicd.tf

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

data_kms_alias_s3.tf

```
data "aws_kms_alias" "s3" {
 name = "alias/aws/s3"
}
```

CodeBuild and CodePipeline Roles Policies and Policy Attachments

These Terraform files define the required Policies, Roles and Policy attachments to the roles that CodeBuild and CodePipeline require:

Policies:

- aws_iam_policy__AWSCodePipelineServiceRole-eu-west-1-pipe-eksworkshop-app.tf
- aws_iam_policy__CodeBuildBasePolicy-eks-cicd-build-app-eu-west-1.tf
- aws_iam_policy__CodeBuildVpcPolicy-eks-cicd-build-app-eu-west-1.tf

Roles:

- aws_iam_role__AWSCodePipelineServiceRole-eu-west-1-pipe-eksworkshop-app.tf
- aws_iam_role__codebuild-eks-cicd-build-app-service-role.tf

Role Policy attachments:

- aws_iam_role_policy_attachment__AWSCodePipelineServiceRole-eu-west-1-pipe-eksworkshop-app.tf
- aws_iam_role_policy_attachment__codebuild-eks-cicd-build-app-service-role__AdministratorAccess.tf
- aws_iam_role_policy_attachment__codebuild-eks-cicd-build-app-service-role__CodeBuildBasePolicy-eks-cicd-build-app-eu-west-1.tf
- aws_iam_role_policy_attachment__codebuild-eks-cicd-build-app-service-role__CodeBuildVpcPolicy-eks-cicd-build-app-eu-west-1.tf

aws_s3_bucket__codepipeline-bucket.tf

This uses an external data provider to run a script: **get-bucket-name.sh** which returns a unique name for the bucket based on you hostname and a fine-grained timestamp.

```
data "external" "bucket_name" {
  program = ["bash", "get-bucket-name.sh"]
}

output "Name" {
  value = data.external.bucket_name.result.Name
}

resource "aws_s3_bucket" "codepipeline-bucket" {
  bucket = data.external.bucket_name.result.Name
  request_payer  = "BucketOwner"
  tags         = {}

  versioning {
    enabled    = false
    mfa_delete = false
  }
  force_destroy = false
  acl         = "private"
}
```

CodeCommit, CodeBuild and CodePipeline recourses

aws_codecommit_repository__eksworkshop-app.tf

Create a CodeCommit repository for our sample application

```
resource "aws_codecommit_repository" "eksworkshop-app" {
 repository_name = "eksworkshop-app"
 description     = "This is the Sample App Repository"
}
```

aws_codebuild_project__eks-cicd-build-app.tf

This creates a CodeBuild project , this will encrypt our repo using the key specified **encryption_key = data.aws_kms_alias.s3.arn**, and use the defined service role **aws_iam_role.codebuild-eks-cicd-build-app-service-role**

By default CodeBuild looks for a file called **buildspec.yml** in the root of the code repository. This file defines all the steps to be taken by the build process.

The vpc_config section specifies the VPC and subnet that CodeBuild connects to.

Disclaimer: For production workloads you should use multiple subnets in multiple availability zones

```
# File generated by aws2tf see https://github.com/aws-samples/aws2tf
# aws_codebuild_project.eks-cicd-build-app:
resource "aws_codebuild_project" "eks-cicd-build-app" {
 badge_enabled  = false
 build_timeout  = 60
 encryption_key = data.aws_kms_alias.s3.arn
 name           = "eks-cicd-build-app"
 queued_timeout = 480
 depends_on     = [aws_iam_role.codebuild-eks-cicd-build-app-service-role]
 service_role   = aws_iam_role.codebuild-eks-cicd-build-app-service-role.arn
 source_version = "refs/heads/master"
 tags           = {}
```

```
artifacts {
 encryption_disabled    = false
 override_artifact_name = false
 type                   = "NO_ARTIFACTS"
}

cache {
 modes = []
 type  = "NO_CACHE"
}

environment {
 compute_type                = "BUILD_GENERAL1_SMALL"
 image                       = "aws/codebuild/amazonlinux2-x86_64-standard:3.0"
 image_pull_credentials_type = "CODEBUILD"
 privileged_mode             = false
 type                        = "LINUX_CONTAINER"
}

logs_config {
 cloudwatch_logs {
  status = "ENABLED"
 }

 s3_logs {
  encryption_disabled = false
  status              = "DISABLED"
 }
}

source {
 git_clone_depth     = 1
 insecure_ssl        = false
 location            = aws_codecommit_repository.eksworkshop-app.clone_url_http
 report_build_status = false
 type                = "CODECOMMIT"
```

```
  git_submodules_config {
    fetch_submodules = false
  }
 }

 vpc_config {
  security_group_ids = [
    data.aws_security_group.cicd.id,
  ]
  subnets = [
    data.aws_subnet.cicd.id,
  ]
  vpc_id = data.aws_vpc.cicd.id
 }
}
```

aws_codepipeline__pipe-eksworkshop-app.tf

A CodePipeline resource is created - this consists of two parts - a source location which is our CodeCommit repo and a build stage that references the CodeBuild project.

A previously defined s3 bucket **location = aws_s3_bucket.codepipeline-bucket.bucket** is used to store build artifacts.

```
resource "aws_codepipeline" "pipe-eksworkshop-app" {
 name      = "pipe-eksworkshop-app"
 depends_on = [aws_iam_role.AWSCodePipelineServiceRole-eu-west-1-pipe-eksworkshop-app]
 role_arn   = aws_iam_role.AWSCodePipelineServiceRole-eu-west-1-pipe-eksworkshop-app.arn
 tags      = {}

  artifact_store {
  location = aws_s3_bucket.codepipeline-bucket.bucket
  type     = "S3"
 }

 stage {
```

```
  name = "Source"

  action {
    category = "Source"
    configuration = {
      "BranchName"          = "master"
      "OutputArtifactFormat" = "CODE_ZIP"
      "PollForSourceChanges" = "false"
      "RepositoryName"      = "eksworkshop-app"
    }
    input_artifacts = []
    name            = "Source"
    namespace       = "SourceVariables"
    output_artifacts = [
      "SourceArtifact",
    ]
    owner     = "AWS"
    provider  = "CodeCommit"
    region    = "eu-west-1"
    run_order = 1
    version   = "1"
  }
}
stage {
  name = "Build"

  action {
    category = "Build"
    configuration = {
      "ProjectName" = "eks-cicd-build-app"
    }
    input_artifacts = [
      "SourceArtifact",
    ]
    name     = "Build"
    namespace = "BuildVariables"
```

```
    output_artifacts = [
     "BuildArtifact",
    ]
    owner     = "AWS"
    provider  = "CodeBuild"
    region    = "eu-west-1"
    run_order = 1
    version   = "1"
   }
 }
}
```

ecr-pull-through.tf

Set up a pull though cache from public.ecr.aws called `aws` in our accounts ECR repo.

```
resource "aws_ecr_pull_through_cache_rule" "aws" {
 ecr_repository_prefix = "aws"
 upstream_registry_url = "public.ecr.aws"
}
```

null-load_ecr.tf

This null provisioner starts the load_ecr.sh script below

```
resource "null_resource" "load_ecr" {
 triggers = {
  always_run = timestamp()
 }
 provisioner "local-exec" {
  on_failure  = fail
  when        = create
  interpreter = ["/bin/bash", "-c"]
  command     = <<EOT
```

```
    ./load_ecr.sh ${var.karpenter_version}
    echo "********************************************************************************"
  EOT
 }
}
```

---

load_ecr.sh

This script logs in to ecr - and does some docker pull operations, this in turn initialises the image in the ECR pull through cache we have setup in our private repo.

```
test -n "$AWS_REGION" && echo AWS_REGION is "$AWS_REGION" || "echo AWS_REGION is not set && exit"
test -n "$ACCOUNT_ID" && echo ACCOUNT_ID is "$ACCOUNT_ID" || "echo ACCOUNT_ID is not set && exit"
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --password-stdin $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
docker pull $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws/awsandy/docker-2048
docker pull $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws/eks-distro/kubernetes/pause:3.5
docker pull $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws/bitnami/aws-cli
docker pull $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws/docker/library/busybox
docker pull $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws/nginx/nginx
docker pull $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/aws/karpenter/controller:v${1}
```