Overview

The following document contains an explanation of the infrastructure implemented for static SEO page generation.The following documentation contains explanations on:

- Terraform Infrastructure deployment
- Architecture of Static SEO Page Generator

Terraform

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied. The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

Terraform Installation

To install terraform on your local workstation you can refer to the official documentation: Terrafor Installation Guide

Terraform Project

The entire infrastructure is created using terraform CLI tool, on the following diagram you can have an idea of the project structure and files used by terraform, explanation on each module/file will come later.

Getting started

Setting AWS Credentials

All resources created by this terraform project will be deployed to AWS so it is necessary to set up your credentials for programmatic access, you can set them up using environment variables:

```
$ export AWS_ACCESS_KEY_ID="<YOUR AWS KEY ID>"
$ export AWS_SECRET_ACCESS_KEY="<YOUR AWS SECRET ACCESS KEY>"
```

```
$ export AWS_DEFAULT_REGION="<REGION>"
```

Or by setting up an AWS profile:

```
$ aws configure --profile <PROFILE NAME>
AWS Access Key ID [None]: <AWS KEY ID>
AWS Secret Access Key [None]: <AWS ACCESS KEY>
Default region name [None]: <REGION>
$ export AWS_PROFILE="<PROFILE NAME>"
```

Terraform Basic Commands

Terraform Init

To initialize terraform configuration for your working directory (root of this project) you can run the following command, you will notice a new directory ".terraform":

```
$ terraform init
```

This command is safe to run it multiple times to keep it up to date to new changes made to the files.

Terraform Validate

If you want to validate your current configuration files and that everything is consistent. This does not make any changes or deploys resources into aws.

```
$ terraform validate
```

Terraform Plan

You need to run this command to generate an execution plan and you can check what resources will be deployed.

```
$ terraform plan
```

Terraform Apply

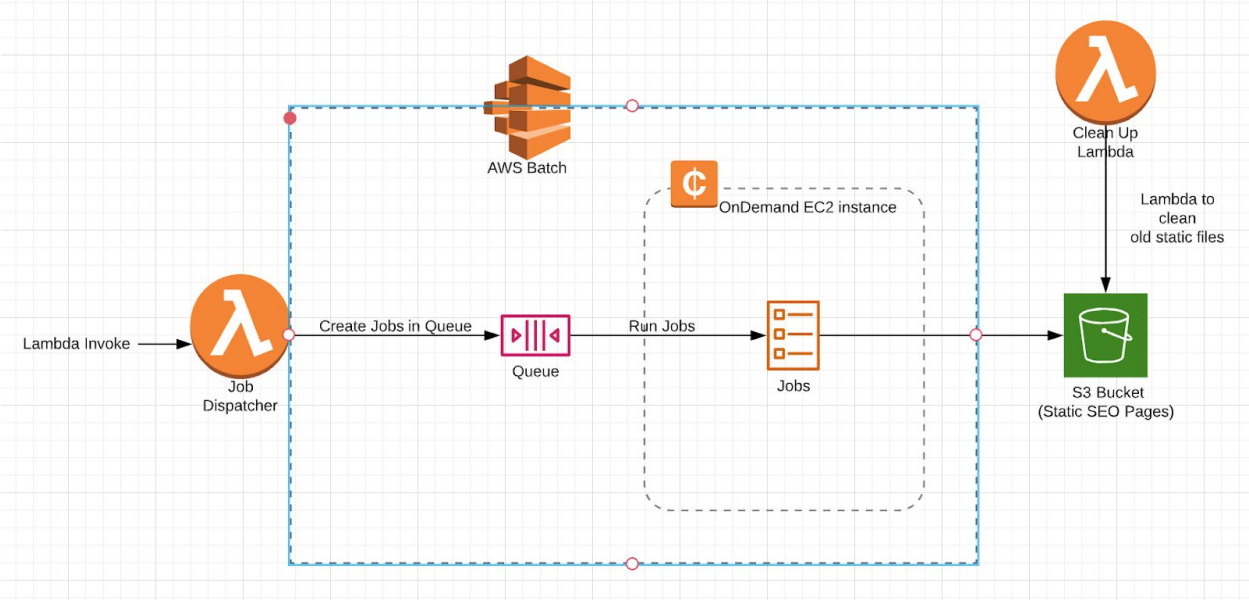
To start resource deployment you need to run the following command, resources will be created or updated according to the changes made in the terraform files.

```
$ terraform apply
```

For further explanation on terraform commands and options: Terraform CLI Documentation

IMPORTANT: Codecommit repository needs to be created before deploying the terraform project infrastructure.

Architecture of Static SEO Page Generator



Introduction
For generating static pages, nClouds uses AWS Lambda, AWS S3, and AWS Batch. For infrastructure as code, Terraform is being used.

Job Dispatcher Lambda

For triggering static pages, the client will invoke the AWS Lambda Function using an AWS CLI. During the AWS Lambda Function invocation, the user will pass the necessary information in the payload Request. Then, lambda runs batch's job in AWS Batch. Hashing Strategy for running multiple batch jobs concurrently is also implemented in this lambda function. Based upon hashing strategy, multiple jobs will be run in AWS Batch.

AWS Batch Jobs

For creating static pages concurrently, we are running jobs in AWS Batch. In AWS Batch, Jobs are the unit of work executed in it. Jobs are executed as containerized applications running on Amazon Batch container instances. The purpose of the job is to invoke

Spokesly's Python Function. Same Child Job will get the URL and HTML file for each invocation. It will then put the HTML file in a precise path of the S3 bucket. The job will run on On-Demand EC2 Instances.

Clean Up Lambda

This Lambda updates the origin of cloudfront distribution to the newly created s3 bucket. It also invalidates the files from edge caches. So, the next time a viewer requests the file, CloudFront returns to the origin to fetch the latest version of the file. Moreover, It also creates expiration life cycle policy in the old s3 bucket for deletion of objects.

Project Structure

Complete project contains three main components:

1. **Terraform** - Infrastructure as code, to deploy infrastructure
2. **AWS Batch Job script** - That will create html files & upload on s3
3. **Lambda** - To trigger AWS Batch job

1) Terraform :

Modules Directory -

Ecr -

This module creates an Elastic Container Repository, this repository will contain all AWS batch job files. The module has the following files:

- **Variables.tf**: This file contains all input parameters optional/required to create a repository.
- **Main.tf**: This file contains resources definition for the aws ecr repository.
- **Outputs.tf**: If there are any outputs to be used on other modules, this file needs to have them defined. If it's not used by any other module then it's optional.

batch_environment -

This module creates multiple resources related to aws batch job like compute environment , job queue, job definition :

- **Variables.tf**: This file contains all input parameters optional/required to create aws batch.
- **Main.tf**: This file contains resources definition for the aws batch, it creates resources and roles with required permissions to execute batch jobs.
- **Outputs.tf**: If there are any outputs to be used on other modules, this file needs to have them defined. If it's not used by any other module then it's optional.

aws-batch-lambda -

This module creates lambda that will trigger aws batch job. Main thing here is you need to upload your lambda code on some S3 bucket as the lambda code will be pulled from the s3 bucket.

- **Variables.tf**: This file contains all input parameters optional/required to create lambda resource.
- **Main.tf**: This file contains lambda resources & role permission required to trigger aws batch.
- **Outputs.tf**: If there are any outputs to be used on other modules, this file needs to have them defined. If it's not used by any other module then it's optional.

cloudfront-route53 -

This module creates cloudfront entry with origin as s3 bucket & also creates route 53 record with your domain name.

- **Variables.tf**: This file contains all input parameters optional/required to create cloudfront & route 53 resources
- **Main.tf**: This file contains mainly two resources cloudfront & route53
- **Outputs.tf**: If there are any outputs to be used on other modules, this file needs to have them defined. If it's not used by any other module then it's optional.

Root Directory:

The root directory contains two files only:

- **Main.tf**: This main file calls all previous modules defined: ecr, batch_envorinment module etc..

You can change bucket name & region to support terraform

```
terraform {

 backend "s3" {

   bucket = "spokesly"

   key = "state.tfstate"

   region = "us-west-2"

 }

}
```

- **Provider.tf :** Provider name & region in which you want to deploy your infrastructure

```
provider "aws" {

   region = "us-west-2"

}
```

- **Variables.tf**: defines the input parameters to configure all the resources that will be created from: main.tf

Variables explanation:

Make changes in default values of each variable as required

1) Identifier will be used in all resources name

```
variable "identifier" {
```

```
 description = "The name for the cluster"
 default     = "spokesly"
 type        = string
}
```

2) Tag to be attached with resources, you can add more in it

```
variable "tags" {
 description = "Tags to be applied to the resource"
 default     = {
   Environment = "dev"
   Team = "Layer2"
   Developer  = "nclouds"
 }
 type        = map
}
```

3) How many ec2machine you need for your batch job

```
variable "desired_vcpus" {
 description = "Desired VCPUs"
 default     = 2
 type        = number
}
```

4) Max machine you need for your batch job

```
variable "max_vcpus" {
 description = "Max VCPUs"
 default     = 4
 type        = number
}
```

5) Min ec2 machine you need

```
variable "min_vcpus" {
 description = "Min VCPUs"
 default     = 0
```

```
 type        = number
}
```

6) Key-pair required for ec2 machines login, add name of existing key-pair

```
variable "ec2_key_pair" {
 description = "EC2 Key pair"
 default     = "oregon-sample-key"
 type        = string
}
```

7) In which vpc you want to launch ec2 machine

```
variable "vpc_id" {
 description = "vpc id"
 default     = "vpc-XXXXX"
 type        = string
}
```

8) Subnets of same vpc

```
variable "subnets" {
 description = "subnets"
 default     =
"subnet-XXXXXX,subnet-XXXXXX,subnet-XXXXXX,subnet-XXXXXX"
 type        = string
}
```

9) ECR image path to be used by the child job of aws batch. Just replace XXXXXXXXX with your account id , if the region is us-west-2 , otherwise change the region as well. Our ECR repo name is spokesly , so no need to change name.

```
variable "image" {
 description = "Docker image to be used"
 default     = "XXXXXXXXX.dkr.ecr.us-west-2.amazonaws.com/spokesly"
 type        = string
}
```

10) Pricecclass of cloudfront, that is related to latency of the content served by cloudfront. For more information check - https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/PriceClass.html

```
variable "cloudfront_price_class" {
 type        = string
 description = "PriceClass for CloudFront distribution"
 default     = "PriceClass_100"
}
```

11) Domain name you want to use for cloudfront

```
variable "domain" {
 description = "The domain name."
 default     = "test-something.com"
}
```

12) Fully qualified domain name you want to use or you can use same domain name.

```
variable "fqdn" {
 description = "The domain name."
 default     = "spokeslynew.test-something.com"
}
```

13) Trigger lambda name:

```
variable "trigger_lambda_name" {
 description = "Name of lambda function"
 default = "aws_trigger_spokesly"
 type        = string
}
```

14) From which bucket terraform pick trigger lambda code

```
variable "trigger_lambda_s3_bucket" {
```

```
 description = "lambda S3 bucket"
 type        = string
 default = "spokesly-lambda"
}
```

15) Trigger Lambda zip file name from bucket

```
variable "trigger_lambda_s3_key" {
 description = "lambda S3 bucket key"
 type        = string
 default = "trigger-lambda.zip"
}
```

16) Bucket prefix, this prefix will be used by lambda to create bucket for file

```
variable "aws_batch_bucketPrefix" {
 description = "AWS batch bucket prefix"
 type        = string
 default = "spokesly"
}
```

17) Cleanup job lambda name

```
variable "cleanup_job_lambda_name" {
 description = "Name of lambda function"
 default = "cleanup_job_spokesly"
 type        = string
}
```

18) Cleanup job lambda s3 bucket

```
variable "cleanup_job_lambda_s3_bucket" {
```

```
 description = "lambda S3 bucket"

 type        = string

 default = "spokesly-lambda"

}
```

19) Cleanup job lambda zip file name

```
variable "cleanup_job_lambda_s3_key" {

 description = "lambda S3 bucket key"

 type        = string

 default = "lambda_cleanup.zip"

}
```

Infrastructure Deployment:

To deploy the entire infrastructure you will need the commands from the **Terraform Basic Commands** section, this will be a step by step guide on how to deploy from scratch and make changes to the infrastructure if needed.

Steps

1. Browse into the terraform root directory
2. Configure your AWS Credentials with proper permissions to deploy infrastructure.
3. Initialize terraform configuration

```
$ terraform init

Initializing modules...

Initializing the backend...
```

```
Initializing provider plugins...
.
.
.

If you ever set or change modules or backend configuration for
Terraform, rerun this command to reinitialize your working directory. If
you forget, other commands will detect it and remind you to do so if
necessary
```

4. After initializing terraform configuration, you can run the planning process to check all the resources that will be created for the project. This command by itself will not create any resources

```
$ terraform plan

Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.
.
.
.
```

5. After reviewing the resources that will be created by this project you can deploy the entire infrastructure applying the previous plan.

```
$ terraform apply
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:

.
.
.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes
```

6. After applying the plan all resources will be created and deployed. You will see something like this after a successful deployment.

```
Apply complete! Resources: <ADDED> added, 0 changed, 0 destroyed
.
.
```

2) AWS Batch Job:

Python script that will create html files & upload same on s3 bucket. Need to push this code on AWS elastic container repository which we created in the previous step (terraform ecr).

For example your

ECR repo link - is 1234567890.dkr.ecr.us-west-2.amazonaws.com

Repo Name - spokesly

Deployment Steps:

1. aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 1234567890.dkr.ecr.us-west-2.amazonaws.com
2. docker build -t spokesly .
3. docker tag api-credential-vault:latest 1234567890.dkr.ecr.us-west-2.amazonaws.com/spokesly:latest
4. docker push 1234567890.dkr.ecr.us-west-2.amazonaws.com/spokesly:latest

3) Lambda:

AWS-trigger-lambda :

This lambda will be used to trigger aws batch job.

Deployment Steps:

1) Need to create a zip of this folder. Run command - **zip trigger-lambda.zip *** inside the lambda directory. variable name used in terraform main `trigger_lambda_s3_key`
2) Upload this zip in some s3 bucket. This bucket name should be used in terraform **aws-batch-lambda** variable name**:** `trigger_lambda_s3_bucket`

clean-up-job-lambda :

This lambda will be used to clean the previous bucket. Change cloudfront origin to new s3 bucket

Deployment Steps:

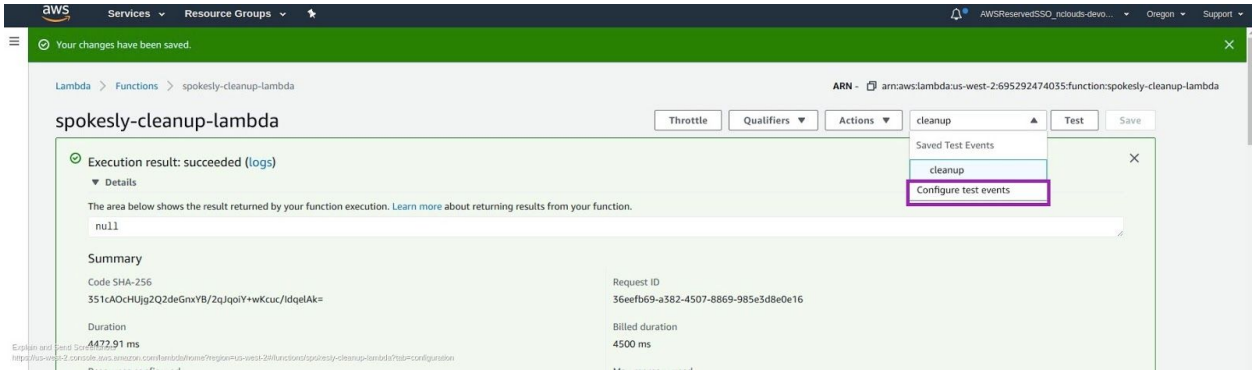
1) Need to create a zip of this folder. Run command - **zip lambda_cleanup.zip *** inside clean-up-job-lambda directory. variable name used in terraform main `cleanup_job_lambda_s3_key`
2) Upload this zip in some s3 bucket. This bucket name should be used in terraform **cleanup-job-lambda** variable name **:** `cleanup_job_lambda_s3_bucket`

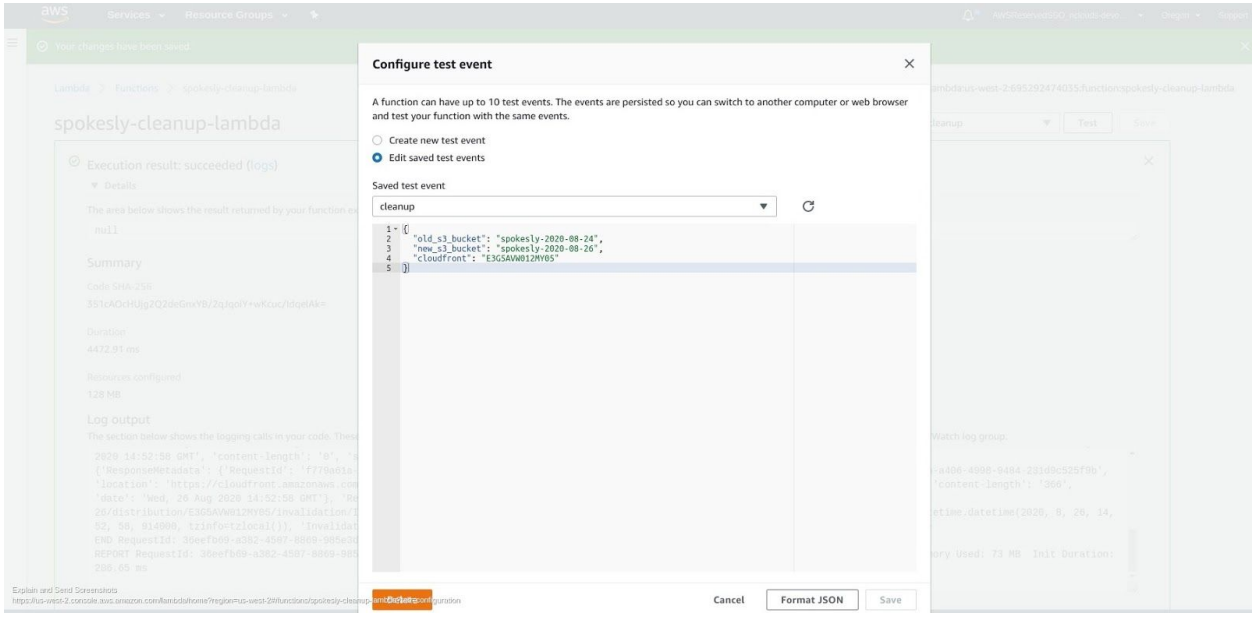
Run:

This lambda need three parameters to run:
Old s3 bucket - Old bucket for which we want to delete file or content in it
New s3 bucket - Bucket which is created by trigger lambda , in which we have newly create files
Cloudfront - Id of the cloudfront , which is created by terraform

Sample:

```
{
  "old_s3_bucket": "spokesly-2020-08-24",
  "new_s3_bucket": "spokesly-2020-08-26",
  "cloudfront": "E3G5AVW012MY05"
}
```





Save test event and click on Test to run lambda to start cleanup job