**GABRIEL MACIEL DE MORAIS**

**Automação de Web Scraping com Selenium usando AWS Lambda, Terraform, AWS Step Functions e Amazon S3**

**Introdução**

Este documento descreve uma solução completa para automatizar a execução de um script de web scraping utilizando Selenium em uma função Lambda da AWS. A solução utiliza o Terraform para provisionar e gerenciar os recursos necessários na AWS, o AWS Step Functions para orquestrar o processo de automação, o Amazon S3 para armazenar os arquivos do Terraform e uma programação de eventos com o Amazon CloudWatch Events para acionar a automação em intervalos regulares.

**Requisitos**

* Conta na AWS com permissões para criar e gerenciar recursos.
* Terraform instalado na máquina local para a configuração inicial.
* Script Python com Selenium para web scraping.

**Arquitetura de Solução**

**Diagrama, Esquemático

Descrição gerada automaticamente**

**1. Instalação do Terraform**

1.1. Faça o download e a instalação do Terraform a partir do site oficial, seguindo as instruções fornecidas para o seu sistema operacional.

**2. Estrutura do Projeto Terraform**

├── main.tf

├── variables.tf

└── lambda\_function.zip

**3. Configuração do Terraform e Armazenamento no Amazon S3**

3.1. No arquivo **main.tf**, configure os recursos da AWS, incluindo a função Lambda, o IAM e quaisquer permissões necessárias.

Exemplo de **main.tf**:

provider "aws" {

region = var.region

}

resource "aws\_lambda\_function" "my\_lambda\_function" {

function\_name = var.lambda\_function\_name

...

}

# Defina outras configurações do Terraform conforme necessário

3.2. No arquivo **variables.tf**, defina as variáveis necessárias, como o nome da função Lambda, a região da AWS, entre outras.

Exemplo de **variables.tf**:

variable "region" {

type = string

default = "us-east-1"

}

variable "lambda\_function\_name" {

type = string

default = "my-lambda-function"

}

# Defina outras variáveis do Terraform conforme necessário

3.3. Faça o upload dos arquivos do Terraform (como **main.tf**, **variables.tf**, etc.) e quaisquer outros arquivos necessários para um bucket do Amazon S3.

**4. Empacotamento do Script Python**

4.1. Empacote o script Python com Selenium, juntamente com quaisquer dependências necessárias, em um arquivo ZIP chamado **lambda\_function.zip**.

**5. Configuração do AWS Step Functions**

5.1. Configure um fluxo de trabalho no AWS Step Functions para orquestrar o processo de automação. O fluxo de trabalho deve incluir as seguintes etapas:

* **CriarRecursosTerraform:** Esta etapa executará o Terraform para criar os recursos na AWS.
* **DeployLambda:** Após a criação dos recursos pelo Terraform, esta etapa implantará a função Lambda na AWS.
* **ExecutarWebScraping:** Uma vez que a função Lambda está implantada, esta etapa invocará a função Lambda para executar o script Python com Selenium.
* **DestruirRecursosTerraform:** Após a execução bem-sucedida da automação e, possivelmente, um período de tempo especificado, esta etapa executará o Terraform novamente para destruir os recursos provisionados na AWS.
* **RetornoAoEstadoInicial:** Configure uma transição de volta ao estado inicial após a conclusão bem-sucedida do processo, permitindo que o fluxo de trabalho reinicie automaticamente.

5.2. Configure as transições entre as etapas do fluxo de trabalho, garantindo que cada etapa seja executada na ordem correta.

**6. Execução Automatizada**

6.1. Configure um evento no Amazon CloudWatch Events para acionar o fluxo de trabalho do AWS Step Functions em intervalos regulares, por exemplo, a cada hora.

6.2. Monitore a execução automatizada e ajuste conforme necessário.

**Relatório Final**

Com base na proposta de cenário apresentada, onde se utiliza as tecnologias Python, Terraform e Docker em uma execução de Lambda Function no ambiente da AWS, é evidente que embora seja possível incorporar contêineres Docker dentro de funções Lambda usando o recurso de Layers, essa abordagem pode não ser a mais ideal por diversos motivos.

Ao considerar a implementação de um serviço de contêiner, especificamente o Docker, dentro de uma função Lambda, é necessário empacotar o contêiner em um formato compatível e incluir sua imagem como parte do pacote da função Lambda. No entanto, gerenciar o ciclo de vida do contêiner dentro do Lambda pode ser complexo e pode dificultar o monitoramento de métricas e serviços, especialmente garantindo a disponibilidade da imagem do contêiner.

Uma alternativa mais robusta e escalável seria utilizar o Amazon Elastic Container Registry (ECR) para armazenar as imagens de contêiner, permitindo que sejam utilizadas por qualquer serviço conforme necessário. No entanto, essa abordagem pode acarretar custos mais elevados devido ao armazenamento e à utilização do serviço ECR.

Considerando essas limitações, a solução proposta enfoca o desenvolvimento de um pipeline gerenciado pelo serviço Step Functions da AWS, oferecendo maior controle e facilidade de monitoramento por meio do CloudWatch. Isso permite acesso simplificado às métricas necessárias para manter o serviço em funcionamento conforme planejado.

Com a necessidade de automatizar o processo de reestruturação e implantação do scraping após bloqueios, uma abordagem mais eficiente e segura envolveria a alocação do arquivo de configuração do Terraform em um bucket do Amazon S3, garantindo sua disponibilidade contínua na AWS. Esse arquivo é acionado por uma função lambda secundária, que inicia o código Terraform para provisionar um novo Lambda responsável pelo scraping. Após um bloqueio, o Step Functions orquestra a destruição e a criação de um novo Lambda, evitando a manutenção de serviços "fantasmas" e garantindo que apenas recursos em uso estejam online, reduzindo assim custos desnecessários. Além disso, a gestão de permissões é assegurada pelo serviço IAM (Identity and Access Management), proporcionando uma camada adicional de segurança ao ambiente.

Além disso, é importante ressaltar que essa abordagem proposta não se limita apenas ao ambiente da AWS, mas também pode ser um projeto passível de migração para outras nuvens públicas, como a Google Cloud Platform (GCP), adaptando-se aos serviços equivalentes oferecidos por essas plataformas. A arquitetura baseada em infraestrutura como código (IaC) e serviços de orquestração de fluxo de trabalho, como o Step Functions, pode ser implementada de forma semelhante em diferentes provedores de nuvem, garantindo a portabilidade e a flexibilidade do sistema conforme as necessidades e requisitos do projeto. Assim, essa solução não apenas atende aos requisitos atuais de infraestrutura na AWS, mas também oferece a possibilidade de expansão e escalabilidade em ambientes multi-cloud.

**Considerações Finais**

* Mantenha as credenciais da AWS e as políticas de permissão configuradas adequadamente.
* Monitore a execução do fluxo de trabalho do Step Functions e da função Lambda para garantir que não haja erros.
* Esteja preparado para fazer ajustes conforme necessário, à medida que o projeto