DevOps

Aula 03 – Parte 1

Recapitulando

Git e GitHub

RECAPITULANDO

...

Atividade Conceitual

ATIVIDADE CONCEITUAL

1. Organizem-se em dois grupo
2. Avaliem a situação problema
3. Apresentem uma solução

ATIVIDADE CONCEITUAL

Tecnologias a serem avaliadas:

- infraestrutura como código (IaC)
- pipelines CI/CD

ATIVIDADE CONCEITUAL

Situação 1:

Imagine que você trabalha em uma empresa de tecnologia que está expandindo rapidamente. A equipe de TI precisa configurar novos servidores e redes para suportar o crescimento, mas fazer isso manualmente é demorado e propenso a erros.

ATIVIDADE CONCEITUAL

Situação 2:

Você faz parte de uma equipe de desenvolvimento que lança atualizações frequentes para um aplicativo web. Cada vez que uma nova funcionalidade é desenvolvida, ela precisa ser testada e implantada rapidamente para manter a competitividade no mercado.

Terraform

Conceitos Fundamentais

Senac

TERRAFORM

🎯 Ferramenta de **Infraestrutura como Código** (IaC) desenvolvida pela HashiCorp.

🎯 Permite a **definição, provisionamento e gerenciamento** infraestrutura de forma declarativa usando arquivos de configuração.

🎯 Descreve-se a infraestrutura desejada em arquivos de texto, e o Terraform se encarrega de criar e gerenciar os recursos necessários.

TERRAFORM

INFRAESTRUTURA COMO CÓDIGO (IAC)

Prática que envolve a gestão e provisionamento de infraestrutura através de código, em vez de processos manuais

Vantagens:

- ❑ **Versionamento**: código pode ser versionado
- ❑ **Reprodutibilidade**: configuração em diferentes ambientes (dev, staging, production).
- ❑ **Colaboração**
- ❑ **Automação**: Processos de deploy e atualização automatizados.

TERRAFORM

PROVIDERS E RECURSOS

Providers:

- Plugins utilizados para interagir com APIs de diferentes serviços, como AWS, Azure, Google Cloud, etc.
- Cada provider oferece um conjunto de recursos que podem ser provisionados.

Recursos:

- são os componentes de infraestrutura que você deseja criar
- como instâncias EC2, buckets S3, etc.

TERRAFORM

VARIÁVEIS E OUTPUTS

Variáveis: permitem que você parametrize sua configuração, tornando-a mais flexível e reutilizável.

```
variable "instance_type" {
  description = "Tipo da instância EC2"
  default     = "t2.micro"
}
```

TERRAFORM

VARIÁVEIS E OUTPUTS

Outputs: são valores que você pode expor após a execução do Terraform, como endereços IP, IDs de recursos, etc.

```
output "instance_ip" {
  description = "IP público da instância EC2"
  value       = aws_instance.example.public_ip
}
```

TERRAFORM

MÓDULOS E REUTILIZAÇÃO DE CÓDIGO

Módulos:

- ❑ são containers para múltiplos recursos usados juntos.
- ❑ permitem que encapsule e reutilize configurações.

```
module "ec2_instance" {
  source = "./modules/ec2"

  instance_type = "t2.micro"
  ami           = "ami-0c55b159cbfafe1f0"
}
```

TERRAFORM

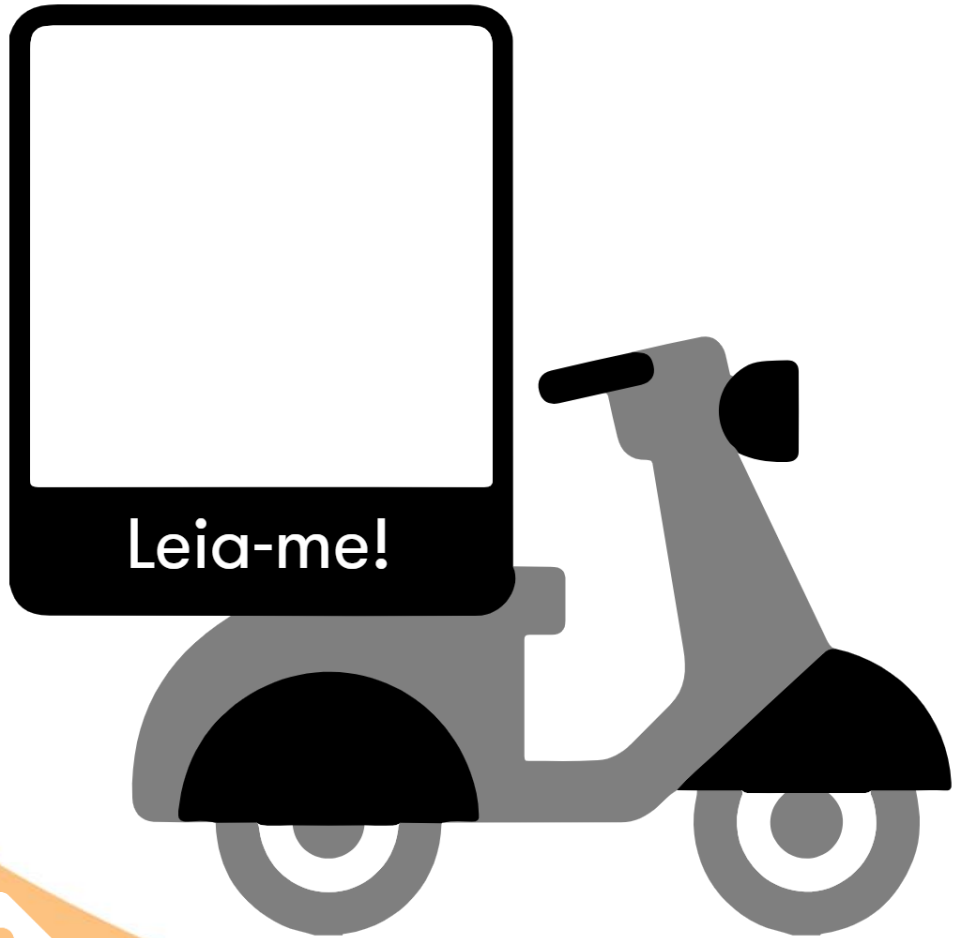
ESTRUTURA DE DIRETÓRIO

Diretório do Projeto

```
├──── main.tf
└──── modules
    └──── ec2
        ├──── main.tf
        ├──── variables.tf
        └──── outputs.tf
```

Atividade Prática 01:

ATIVIDADE PRÁTICA 01



Comandos Básicos do Terraform

.....

Pipeline CI/CD e DevOps

Conceitos Fundamentais

Senac

INTEGRAÇÃO CONTÍNUA (CI)

Prática de integrar alterações de código em um repositório central frequentemente, garantindo validação automática por meio de testes e builds.

Principais Benefícios

- Detecção precoce de erros.
- Redução de conflitos de integração.
- Feedback rápido para desenvolvedores.

Práticas Essenciais

- Versionamento de Código
- Testes Automatizados
- Build Automatizado (Compilação e empacotamento do código)
- Notificações (em caso de falha no pipeline.

ENTREGA CONTÍNUA (CD)

A Entrega Contínua (Continuous Delivery) e Implantação Contínua (Continuous Deployment) automatizam a entrega de código em ambientes de staging ou produção.

Diferenças Entre CD e CD

- **Continuous Delivery**: Requer aprovação manual para implantação.
- **Continuous Deployment**: Implanta automaticamente após testes bem-sucedidos.

ENTREGA CONTÍNUA (CD)

Práticas Recomendadas

- **Infraestrutura como Código (IaC):** Use Terraform ou CloudFormation.
- **Deploy Imutável**: Containers Docker ou imagens pré-construídas.
- **Rollbacks Automáticos**: Reverta em caso de falha.

FERRAMENTAS DE CI/CD

Ferramentas:

- Jenkins
- Gitlab Cli
- GitHub Actions

FERRAMENTAS DE CI/CD

Jenkins

Vantagens:

- Flexibilidade
- Plugins extensivos.

Exemplo de Pipeline (Jenkinsfile)

```
pipeline {
    agent any
    stages {
      stage('Test') {
        steps {
          sh 'python -m pytest tests/'
        }
      }
      stage('Build') {
        steps {
          sh 'docker build -t data-analysis-app:latest .'
        }
      }
      stage('Deploy') {
        steps {
          sh 'kubectl apply -f k8s/deployment.yaml'
        }
      }
    }
  }
```

FERRAMENTAS DE CI/CD

GitLab CI

Vantagens:

- Integração nativa com repositórios GitLab

Exemplo de Pipeline (.gitlab-ci.yml)

```
stages:
  - test
  - build
  - deploy

test:
  image: python:3.9
  script:
    - pip install -r requirements.txt
    - pytest tests/

build:
  image: docker:latest
  script:
    - docker build -t data-analysis-app:latest .
    - docker push registry.gitlab.com/seu-projeto/data-analysis-app:latest

deploy:
  image: bitnami/kubectl:latest
  script:
    - kubectl apply -f k8s/deployment.yaml
```

FERRAMENTAS DE CI/CD

GitHub Actions

Vantagens:

- Integração direta com repositórios GitHub

- Exemplo de Pipeline (.github/workflows/pipeline.yml)

```
name: CI/CD Pipeline
 on: [push]
 jobs:
   test:
     runs-on: ubuntu-latest
     steps:
       - uses: actions/checkout@v4
       - name: Run Tests
         run: |
           pip install -r requirements.txt
           pytest tests/
   build:
     runs-on: ubuntu-latest
     needs: test
     steps:
       - uses: actions/checkout@v4
       - name: Build Docker Image
         run: docker build -t data-analysis-app:latest .
   deploy:
     runs-on: ubuntu-latest
     needs: build
     steps:
       - uses: actions/checkout@v4
       - name: Deploy to Kubernetes
         run: kubectl apply -f k8s/deployment.yaml
```

MONITORAMENTO E LOGGING

Ferramentas:

Monitoramento:

- Prometheus – Coleta métricas em tempo real.
- Grafana – Visualização de dados via dashboards.

Logging:

- ELK Stack (Elasticsearch, Logstash, Kibana)
- Loki + Grafana: Solução leve para logs em Kubernetes

MONITORAMENTO E LOGGING

Integração no Pipeline

- Coleta de Métricas: Adicione endpoints Prometheus ao aplicativo.
- Logs em Kubernetes: Use Fluentd ou Loki para agregar logs de containers.

CULTURA DEVOPS E PRÁTICAS COLABORATIVAS

Princípios Fundamentais

- Colaboração: Quebre silos entre desenvolvedores e operações.
- Automação: Elimine tarefas manuais repetitivas.
- Feedback Contínuo: Retroalimentação rápida entre equipes.

CULTURA DEVOPS E PRÁTICAS COLABORATIVAS

Práticas Recomendadas

- **Postmortems Sem Culpa:** Analise falhas sem atribuir culpas.
- **Shift-Left Testing**: Execute testes de segurança e performance cedo no ciclo.
- **Documentação como Código**: Mantenha docs atualizados usando ferramentas como **MkDocs** ou **Sphinx**.

Atividade Prática 02: