MBA USP ESALQ

MBA EM **ENGENHARIA DE SOFTWARE**

TEORIA E PRÁTICA

Infra As Code (IAC)

Professor Leonardo Lima

Sumário

APRESENTAÇÃO


Professor: Leonardo Martins

Formação Acadêmica

- PUC Minas - Especialização, Gestão em Engenharia (2025)
- IPT/USP - Mestrado em Engenharia de Computação (2018)
- FGV - MBA em Administração de Empresas (2011)
- IFRN - 2o/3o graus (2004)

Experiência Profissional

- Sodexo / Pluxee
- Creditas
- DELL
- Walmart.com
- Globo.com
- Yahoo!

Contatos

- Linkedin: https://linkedin.com/in/leonardoml/
- Blog: https://infraascode.com.br
- Newsletter: https://engineeringmanager.com.br
- X: @leonardoml , @infraascode_br

APRESENTAÇÃO DO CURSO INFRAESTRUTURA AS A CODE (IAC)

Prezados(as) alunos(as),

Sejam muito bem-vindos a disciplina de *Infraestrutura como Código (IaC)* do MBA da USP/Esalq. É uma grande satisfação fazer parte dessa jornada de aprendizado e contribuir com o desenvolvimento do conhecimento de vocês em um tema tão transformador, que certamente mudará a forma como enxergam e interagem com a infraestrutura de TI.

Esta disciplina foi cuidadosamente estruturada para prepará-los com as competências técnicas e estratégicas necessárias para liderar processos de modernização em ambientes de tecnologia. Ao dominar os conceitos de IaC, vocês serão capazes não apenas de automatizar e otimizar operações, mas também de elevar significativamente os padrões de qualidade, segurança e eficiência das arquiteturas com as quais trabalham.

Em um cenário cada vez mais dinâmico e competitivo, a capacidade de gerenciar infraestrutura de forma programável, automatizada e escalável deixa de ser um diferencial e passa a ser uma exigência do mercado. O conteúdo abordado aqui será, sem dúvida, um grande impulsionador do crescimento profissional de cada um de vocês.

Ao longo da disciplina, vocês aprenderão a:

- Pensar estrategicamente sobre o papel da infraestrutura na evolução de produtos digitais;
- Gerenciar plataformas baseadas em IaC de forma padronizada e repetível;
- Administrar recursos complexos com precisão e controle;
- Integrar a infraestrutura ao ciclo de vida de desenvolvimento de software, promovendo agilidade e confiabilidade.

Esses conhecimentos proporcionarão uma visão sistêmica e estratégica, capacitando-os a liderar mudanças contínuas e a construir ambientes tecnológicos mais resilientes, seguros e eficientes.

Vejam o nosso planejamento de aulas:

- Tópicos 1-2: Apresentação do curso e Fundamentos teóricos
- Tópicos 3-4: Fundamentos de uso de ferramentas no contexto de IaC
- Tópicos 5-6: Configuração de serviços e servidores
- Tópicos 7-8: Gerenciamento serviços com IaC
- Tópicos 9-10: Governança de infraestruturas e futuro do IaC

Temas centrais das aulas:

Apresentação do curso e Fundamentos teóricos, apresenta os conceitos fundamentais de IaC, evolução histórica, benefícios operacionais e além de trazer práticas e fluxos de trabalho.

Fundamentos de uso de ferramentas no contexto de IaC, monstra como utilizar ferramentas essenciais como Terraform, Ansible e técnicas de provisionamento e gerenciamento de infraestrutura.

Aulas 5-6: Configuração de serviços e servidores, continuamos dos fluxos de configuração de servidores e serviços po meio de código, implementar estratégias de imutabilidade e idempotência.

Gerenciamento de serviços com IaC, Objetivo: Implementar testes automatizados para infraestrutura, validar conformidade e estabelecer práticas de qualidade e segurança.

Governança de infraestruturas e futuro do IaC Objetivo: Estruturar governança para grandes organizações, definir modelos operacionais para equipes e explorar tendências futuras da IaC.

Desejo a todos um excelente curso!!

FUNDAMENTAÇÃO TEÓRICA

O que é Infra as Code

Infraestrutura como Código (IaC) é a prática de definir e gerenciar recursos de infraestrutura de TI por meio de arquivos de configuração legíveis por humanos. Esses arquivos descrevem, de forma declarativa ou imperativa, elementos como servidores, redes, balanceadores de carga, grupos de segurança e serviços em nuvem. A principal característica da IaC é a automação da criação, alteração e destruição de recursos de forma reprodutível e versionável, eliminando a necessidade de configurações manuais.

Com IaC, ferramentas como Terraform, Ansible, AWS CloudFormation e Pulumi são usadas para aplicar configurações em ambientes distintos (DEV, QA, PROD) de maneira padronizada. Isso permite controle de mudanças via versionamento em sistemas como Git, validação por testes automatizados e integração com pipelines de CI/CD. A abordagem aumenta a consistência entre ambientes e facilita auditoria e governança, sendo um componente essencial em práticas modernas como DevOps e engenharia de plataformas.

Qual é a razão de utilizar IaC?

Os processos manuais de configuração de infraestrutura introduzem variabilidade humana que compromete a consistência da entrega de um trabalho. No modelo de trabalho manual, a pessoa que administra o sistemas executa tarefas seguindo interpretações pessoais das documentações, podendo resultar em configurações divergentes entre ambientes. A ausência de padronização gera inconsistências que se acumulam ao longo do tempo, criando ambientes únicos e não replicáveis. Erros de digitação, omissão de passos, falta de atenção e configurações incorretas são frequentes quando dependemos de intervenção humana para tarefas repetitivas. Documentações desatualizadas agravam o problema, pois quem executa essa atividade trabalha com informações obsoletas que não refletem o estado real da infraestrutura.

O modelo de trabalho automatizado por meio de ferramentas de Infraestrutura como Código (IaC) elimina variabilidade humana ao utilizar de definições declarativas que executam de forma idêntica em qualquer ambiente. Códigos de infraestrutura são testados, versionados e validados antes da execução, garantindo que apenas configurações aprovadas sejam aplicadas. A automação permite execução de tarefas complexas sem intervenção manual, reduzindo drasticamente a incidência de erros operacionais.

Os processos automatizados apresentam taxas de erro menores que os processos manuais. A automação por meio de IaC permite validação prévia de configurações, execução de testes automatizados e

verificação de conformidade antes da aplicação em produção. Pipelines de CI/CD para infraestrutura implementam múltiplas camadas de validação, desde verificações sintáticas até testes de integração completos.

Provocar ou fazer a transição de processos manuais para Infraestrutura como Código representa mudança fundamental na forma como organizações gerenciam tecnologia. Enquanto processos manuais limitam escalabilidade e introduzem riscos operacionais significativos, IaC proporciona consistência, confiabilidade e velocidade necessárias para operações modernas. A automação não apenas reduz erros, mas possibilita práticas avançadas como deployment contínuo, recuperação automatizada de desastres e otimização dinâmica de recursos. Organizações que adotam IaC observam redução substancial em incidentes de produção, aceleração na entrega de novos ambientes e melhoria na colaboração entre equipes de desenvolvimento e operações, estabelecendo fundação sólida para crescimento sustentável e ganho de maiores fatias de mercado.

Figura 1: Linhas de automação

Contextualização histórica

Vamos voltar no tempo, vamos para um período até um pouco antes de 2000. Neste período a construção de infraestrutura de TI era majoritariamente manual. As empresas compravam servidores físicos, instalavam sistemas operacionais, configuravam redes e bancos de dados diretamente em cada máquina. Cada servidor era único, com configurações feitas por meio de comandos diretos no sistema, o que dificultava a padronização e aumentava o risco de erros humanos. O tempo para disponibilizar um novo ambiente podia levar dias ou semanas, dependendo da complexidade.

Figura 2: Fluxo de entrega de infraestrutura

Com a popularização da virtualização no início dos anos 2000, especialmente com ferramentas como VMware, as empresas começaram a automatizar partes do processo, criando máquinas virtuais em vez de servidores físicos. Isso reduziu o tempo de provisionamento e trouxe mais flexibilidade, mas as configurações ainda eram feitas manualmente ou com scripts imperativos, como Bash ou PowerShell, sem padronização entre equipes.

A partir de 2008, com o surgimento do movimento DevOps, a integração entre times de desenvolvimento e operações se intensificou. A ideia central era eliminar silos e acelerar o ciclo de entrega de software. Nesse contexto, começou-se a tratar a infraestrutura como parte do código da aplicação, surgindo o conceito de Infrastructure as Code (IaC). Ferramentas como Puppet e Chef, baseadas em modelos declarativos e com foco em configuração de sistemas, ganharam espaço.

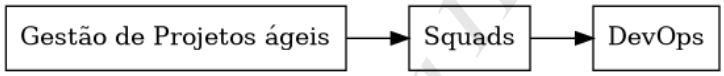


Figura 3: Fluxo de entrega no modelo Ágil

Com a consolidação das nuvens públicas (AWS, Azure, GCP), o paradigma mudou novamente. Os recursos de infraestrutura passaram a ser acessados por APIs. Isso permitiu um nível de automação mais completo do ciclo de vida da infraestrutura. A necessidade de ferramentas mais alinhadas com esse modelo levou ao surgimento de soluções como o Terraform, da HashiCorp, que abstraem provedores de nuvem e usam um modelo declarativo baseado em arquivos de texto versionáveis.

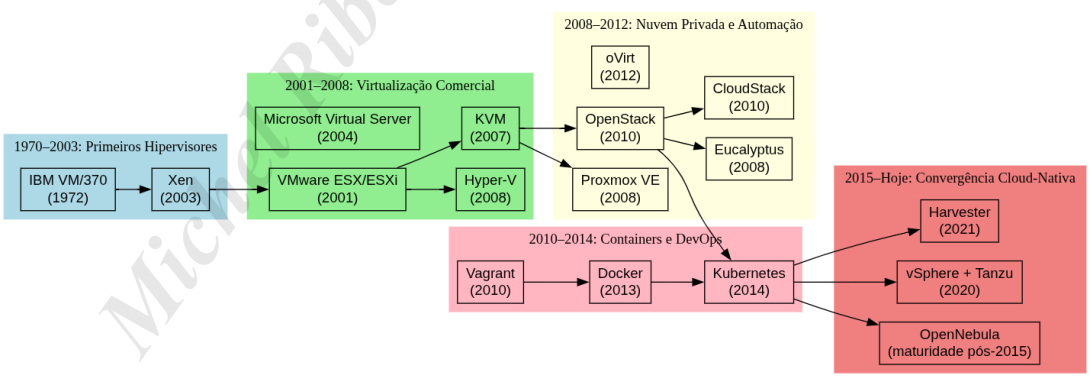


Figura 4: Histórico de mecanismos de virtualização

Hoje, o uso de IaC é uma prática comum em ambientes modernos que procuram aumentar sua

qualidade de entrega ou que buscam se modernizar. A infraestrutura é escrita, testada e versionada como o código da aplicação, possibilitando reprodutibilidade, escalabilidade e rastreabilidade.

Nos dias atuais o movimento IaC evolui para modelos ainda mais integrados, com ferramentas que combinam infraestrutura, segurança e pipelines, e com suporte a práticas como policy as code e compliance as code. O objetivo permanece o mesmo, reduzir erros manuais, acelerar entregas e permitir que os times tenham controle total sobre seu ambiente de forma programável. Vejam a timeline das ferramentas mais utilizadas.

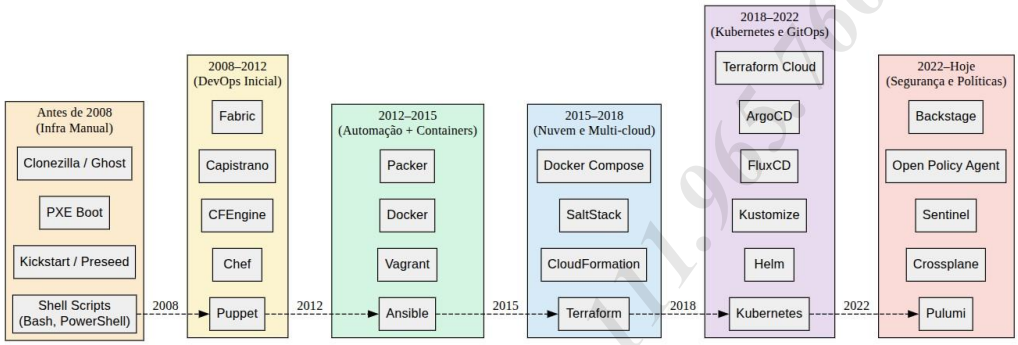


Figura 5: Timeline das ferramentas de IaC

Ciclo de desenvolvimento de software

O ciclo de desenvolvimento de software, representado frequentemente como um loop contínuo de build, test, deploy e release, reflete a prática de entrega contínua e integração contínua (CI/CD), fundamentais para garantir entregas frequentes, seguras e controladas de aplicações. O estágio de build envolve a compilação e empacotamento do código; o de teste executa validações automatizadas para garantir qualidade e funcionalidade; o de deploy movimenta a aplicação para ambientes controlados como: desenvolvimento ou homologação; e o de release representa a liberação efetiva para o ambiente de produção.

A repetição desse ciclo permite detectar erros mais cedo, reduzir tempo de entrega e aumentar a confiabilidade das mudanças. Quando aplicamos esses mesmos princípios à infraestrutura, permite tratar configurações e provisionamento de ambientes com as mesmas práticas de versionamento, testes e automação, promovendo alinhamento entre o código da aplicação e a infraestrutura que a sustenta.

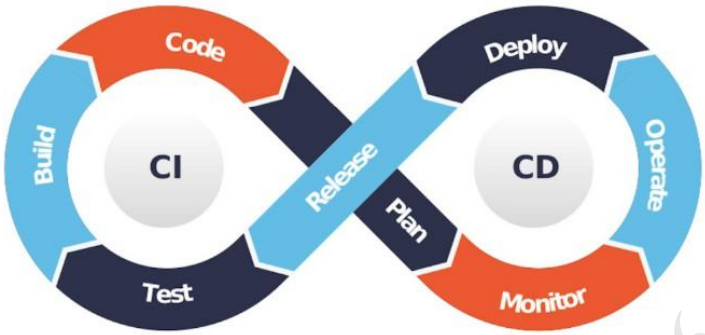


Figura 6: Loop CI/CD

Versionamento de código

O versionamento de código é uma prática essencial no desenvolvimento de software que permite acompanhar e gerenciar as alterações feitas no código-fonte ao longo do tempo. Utilizando sistemas como Git, é possível registrar cada modificação, identificar quem fez a mudança, quando foi feita e qual foi o motivo. Isso facilita a colaboração entre desenvolvedores, possibilita o trabalho simultâneo em diferentes partes do sistema e garante que qualquer erro possa ser revertido com segurança para uma versão anterior.

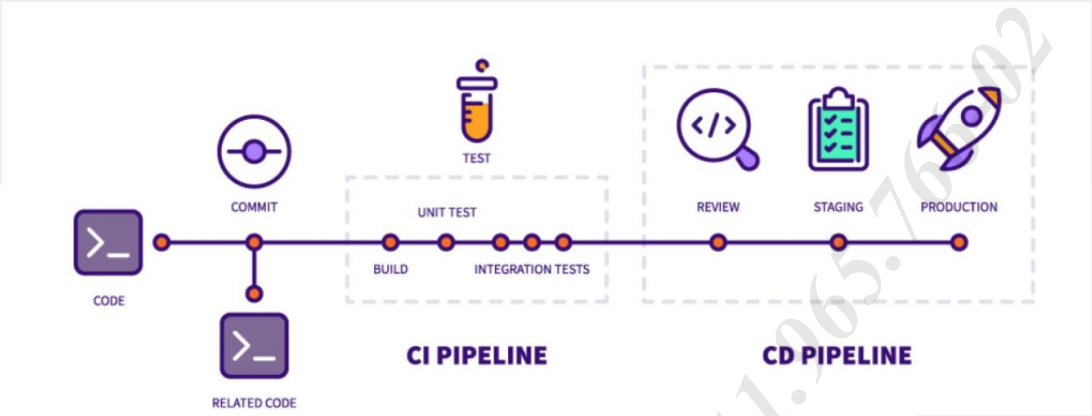
Entre os principais benefícios do versionamento estão o controle de histórico, a rastreabilidade das mudanças, a facilidade de integração entre múltiplos desenvolvedores e a automação de processos de integração contínua. Além disso, o versionamento é essencial para práticas como revisão de código, testes automatizados e controle de releases. No contexto de cloud e infraestrutura, o versionamento também se aplica a scripts e configurações de ambiente, permitindo tratar a infraestrutura como código e mantendo a consistência e rastreabilidade entre os ambientes.

```
1 git ls-tree HEAD
2 git log --pretty=format:"%h %s" --graph
```

Pipelines de CI/CD

Pipelines de CI/CD (Integração Contínua e Entrega Contínua) são fluxos automatizados que integram, testam e implantam mudanças de código com rapidez e segurança. Eles substituem tarefas manuais sujeitas a erros, como rodar testes ou copiar arquivos, garantindo consistência e rastreabilidade. Na prática, a CI valida automaticamente cada mudança no repositório, enquanto a CD entrega essas mudanças em ambientes controlados ou de produção, permitindo ciclos curtos de feedback e correção.

Quando combinamos CI/CD com Infraestrutura como Código (IaC), tanto aplicação quanto infraestrutura passam a ser gerenciadas de forma automatizada e versionada. Alterações em scripts de infraestrutura seguem o mesmo fluxo de validação que o código da aplicação, com testes automatizados e verificações de segurança antes do deploy. Isso garante ambientes consistentes em todas as etapas e reduz o risco operacional, permitindo que infraestrutura e aplicação evoluam de forma sincronizada.



Conceitos fundamentais de IaC

Vamos compreender os conceitos fundamentais de IaC e sua importância para operações modernas.

Definição e evolução da Infraestrutura como Código

A Infraestrutura como Código (IaC) representa um paradigma onde a infraestrutura computacional é gerenciada através de arquivos de definição versionáveis, em vez de configurações manuais. Surgiu como resposta às limitações da administração manual de sistemas, evoluindo de scripts simples feitos em Shell-script até frameworks declarativos completos. Historicamente, a evolução acompanhou o crescimento da computação em nuvem, com marcos importantes incluindo o surgimento do Chef (2009), Puppet (2008), a ascensão das nuvens públicas e posteriormente o Terraform (2014), cada um representando avanços na forma como a infraestrutura é definida e provisionada programaticamente.

Benefícios: consistência, repetibilidade, documentação viva

A consistência proporcionada pela IaC elimina variações entre ambientes, reduzindo drasticamente erros operacionais. A repetibilidade permite recriar ambientes inteiros de forma idêntica, acelerando recuperação de desastres e facilitando a criação de ambientes de teste. O versionamento registra cada alteração na infraestrutura, possibilitando auditoria completa e rollbacks precisos quando necessário.

Como documentação viva, o código de infraestrutura representa o estado atual exato do ambiente, eliminando a desatualização comum em documentações tradicionais e servindo como fonte única de verdade sobre a configuração do sistema MORRIS (2021)

Princípios fundamentais: idempotência, imutabilidade

- Idempotência garante que múltiplas execuções do mesmo código produzam o mesmo resultado, independente do estado inicial do sistema, eliminando efeitos colaterais indesejados.

$$f(f(x)) = f(x)$$

- Imutabilidade promove a substituição completa de componentes em vez de modificações in-place, aumentando a confiabilidade e previsibilidade das implantações. Servidores tornam-se descartáveis, recriados a partir de definições em vez de atualizados incrementalmente.

Desafios

1. Video 01 - https://www.youtube.com/watch?v=6n9ESFJTnHs
2. Video 02 - https://www.youtube.com/watch?v=Kge89TAJGf0
3. Video 03 - https://www.youtube.com/watch?v=qWgy1uHIxwI

FUNDAMENTOS DE USO DE FERRAMENTAS NO CONTEXTO DE IAC

Atualmente, o papel da infraestrutura se expandiu. Ela não é mais uma camada isolada, responsável apenas por disponibilizar servidores, redes e armazenamento. Hoje, a infraestrutura está integrada a todo o ciclo de vida da tecnologia nas empresas.

Tudo começa na **Arquitetura**. É nesse estágio que se definem os padrões técnicos e os componentes que suportarão as soluções — por exemplo, decisões sobre uso de microsserviços, containers, cloud pública ou híbrida. Essas escolhas impactam diretamente como a infraestrutura será provisionada e operada.

Com a arquitetura definida, a **Infraestrutura** executa essas decisões por meio de automação, buscando escalabilidade e disponibilidade. Aqui entram práticas como Infrastructure as Code, uso de pipelines, e serviços gerenciados. A infraestrutura moderna é elástica, observável e orientada por APIs.

A partir daí, surgem exigências de **Governança**. Ou seja, a necessidade de controle sobre quem pode fazer o quê, onde e quando. Isso envolve controle de acesso, rastreabilidade, compliance e padronização. Sem governança, a infraestrutura pode crescer de forma desordenada e gerar riscos operacionais.

A **Segurança** atua de forma transversal. Ela precisa estar presente desde a arquitetura (com princípios como zero trust), passando pela infraestrutura (com segmentações, criptografia, escaneamentos), até os processos de CI/CD e operação. Segurança não é um anexo, mas uma parte integrada do ambiente.

Por fim, entra o **FinOps** — a gestão financeira da nuvem. Em ambientes dinâmicos, é essencial entender o custo real das decisões arquiteturais e operacionais. O FinOps ajuda a alinhar infraestrutura e orçamento, promovendo uso eficiente de recursos com visibilidade e responsabilidade.

Essa integração entre arquitetura, infraestrutura, governança, segurança e FinOps exige que os profissionais pensem de forma sistêmica. Cada decisão em uma camada impacta diretamente as demais. O papel da infraestrutura hoje é ser um elo entre essas áreas.

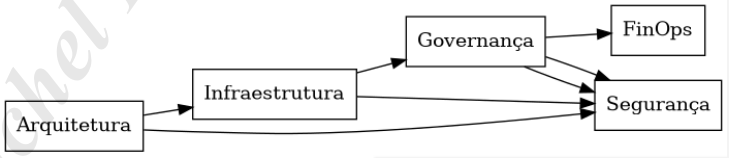


Figura 7: Papel da infraestrutura hoje

Desenhos de Arquiteturas como Código

O desenho de arquiteturas é uma etapa essencial no ciclo de vida de sistemas. Ele descreve de forma visual como os componentes interagem e como os fluxos de dados percorrem a solução. No contexto de Infrastructure as Code, esses diagramas ajudam equipes a compartilhar entendimento sobre ambientes e automatizar a criação dos recursos.

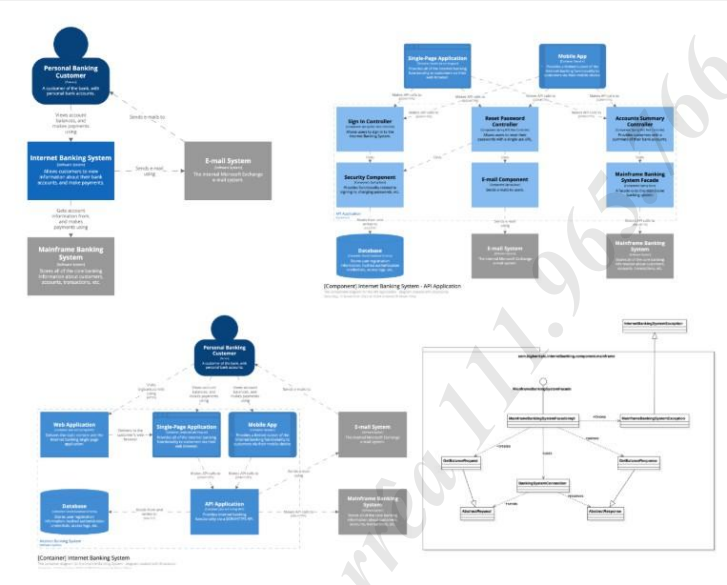


Figura 8: Diagramas com C4

Um modelo bastante utilizado é o C4 Model, que organiza os diagramas em quatro níveis: Contexto, Container, Componente e Código.

- **Contexto:** Esta visão apresenta o sistema como um todo, seus usuários e outros sistemas com os quais interage o público alvo são os times de negócio, stakeholders, times externos.
- **Container:** Os contêineres identificam dentro do contexto do sistema o: front-end, APIs, bancos de dados, serviços de background, essa visão é para os desenvolvedores, arquitetos e operadores do sistema.
- **Componente:** Apresenta os componentes internos de cada container (módulos, serviços, pacotes), o público de interesse desta visão são os desenvolvedores e arquitetos.
- **Código:** Mostra detalhes do código-fonte, como classes, métodos, arquivos, essa visão é destinada para os desenvolvedores.

No fluxo de desenvolvimento, desenhar arquiteturas e fluxos não é apenas documentação. É parte do processo de alinhamento técnico e validação de decisões. Ao criar diagramas consistentes, as

equipes conseguem prever riscos, planejar segurança e otimizar custos. O uso de diagramas gerados por código ainda garante que a documentação acompanhe as mudanças no ambiente.

O desenho de arquiteturas e a prática de Infrastructure as Code estão conectados. Enquanto o código cria os recursos, os diagramas contam a história visual de como esses recursos se relacionam. Essa combinação fortalece a governança e facilita auditorias e revisões.

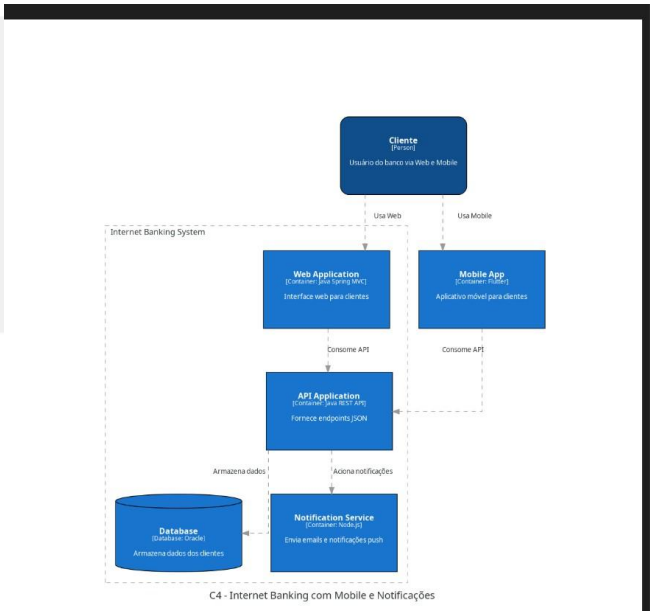
Para a apoiar a teoria, vamos utilizar a biblioteca Python diagrams permitem gerar imagens de arquiteturas a partir de scripts. Com ela, é possível descrever elementos como load balancers, bancos de dados e clusters Kubernetes de forma programável. Isso é útil para manter os diagramas alinhados com o código de infraestrutura real.

Utilizando o Modelo C4

Requisitos de um sistema genérico, construir um diagrama que representa a arquitetura de um sistema de internet banking com suporte a acesso web e móvel, os requisitos são os seguintes:

- O Cliente, que interage com a plataforma de duas maneiras:
  - Utilizando a Web Application, desenvolvida em Java Spring MVC e responsável por fornecer a interface web, ou acessando o Mobile App, implementado em Flutter e destinado ao uso em dispositivos móveis
  - Na interface web no aplicativo Mobile, ambos se conectam ao API Application, uma API REST em Java que fornece os endpoints JSON necessários para as operações de negócio.
  - A API realiza duas ações principais:
    * Armazena e recupera dados no Database, que utiliza tecnologia Oracle para persistir as informações dos clientes.
    * Aciona o serviço de notificação, um componente em Node.js encarregado de enviar e-mails e notificações push.
  - O conjunto formado pela Web Application, API Application, Database e Notification Service está contido no limite lógico denominado Internet Banking System.
  - O Mobile App fica representado como um container externo que consome os mesmos serviços expostos pela API.

DESENHO DE ARQUITETURA EM MODELO C4 - CONTEXTO:




```python
from diagrams import Diagram
from diagrams.c4 import Person, Container, Database, SystemBoundary, Relationship

with Diagram("C4 - Internet Banking com Mobile e Notificações", direction="TB"):
    customer = Person("Cliente", "Usuário do banco via Web e Mobile")

    with SystemBoundary("Internet Banking System"):
        webapp = Container(
            "Web Application",
            "Java Spring MVC",
            "Interface web para clientes"
        )

        api = Container(
            "API Application",
            "Java REST API",
            "Fornece endpoints JSON"
        )

        database = Database(
            "Database",
            "Oracle",
            "Armazena dados dos clientes"
        )

        notification_service = Container(
            "Notification Service",
            "Node.js",
            "Envia emails e notificações push"
        )

    mobileapp = Container(
        "Mobile App",
        "Flutter",
        "Aplicativo móvel para clientes"
    )

    customer >> Relationship("Usa Web") >> webapp
    customer >> Relationship("Usa Mobile") >> mobileapp

    webapp >> Relationship("Consome API") >> api
    mobileapp >> Relationship("Consome API") >> api

    api >> Relationship("Armazena dados") >> database
    api >> Relationship("Aciona notificações") >> notification_service
```

Figura 9: Exemplo 01 - Diagrama C4 - Contexto

DESENHO DE ARQUITETRA EM MODELO C4 - CONTAINER:

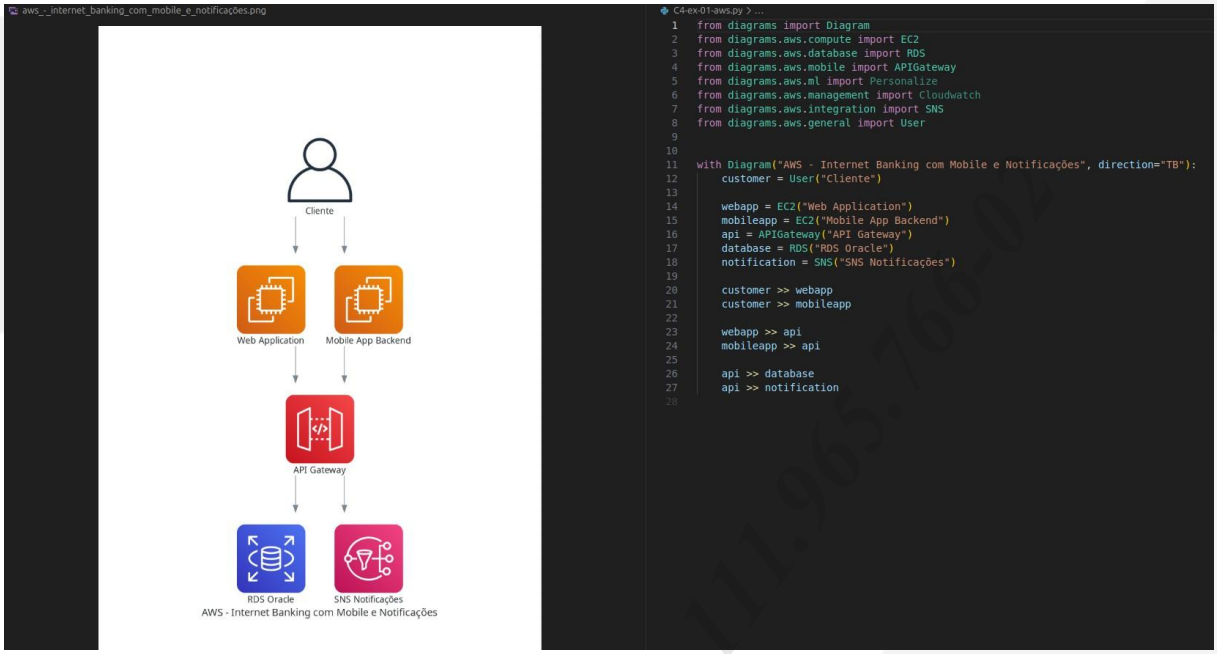


Figura 10: Exemplo 01 - Diagrama Infra

Desafios de Arquitetura

1. Realizar os exercícios de Arquitetura 01
2. Realizar os exercícios de Arquitetura 02
3. Realizar os exercícios de Arquitetura 03
4. Realizar os exercícios de Arquitetura 04
5. Aprofundar os conhecimentos na lib C4 Model https://c4model.com/

INFRAESTRUTURA COMO CÓDIGO

No passado, criar infraestrutura era uma atividade manual. As equipes acessavam o console dos provedores de nuvem ou conectavam diretamente nos servidores para configurar redes, máquinas virtuais e permissões. Cada recurso precisava ser criado individualmente, seguindo passos descritos em documentos ou memórias dos administradores. Esse processo demandava muito tempo e dependia de conhecimento tácito que não ficava registrado de forma estruturada.

Esse método tornava impossível reproduzir o ambiente de forma idêntica e gerava dependência de pessoas. Pequenas diferenças de configuração, ordens de execução ou erros humanos geravam inconsistências entre ambientes de desenvolvimento, homologação e produção. Quando era necessário escalar ou reconstruir a infraestrutura, a equipe precisava repetir tudo do zero, correndo o risco de esquecer etapas ou aplicar parâmetros incorretos.

Utilizar infraestrutura como código no seu dia a dia é uma prática que organiza, automatiza e documenta a criação dos ambientes. Ao invés de configurar recursos manualmente, o time escreve arquivos de definição que descrevem cada componente de forma declarativa. Isso reduz erros operacionais, facilita auditorias e permite reproduzir ambientes DEV, QA, HML e PROD de forma consistente.

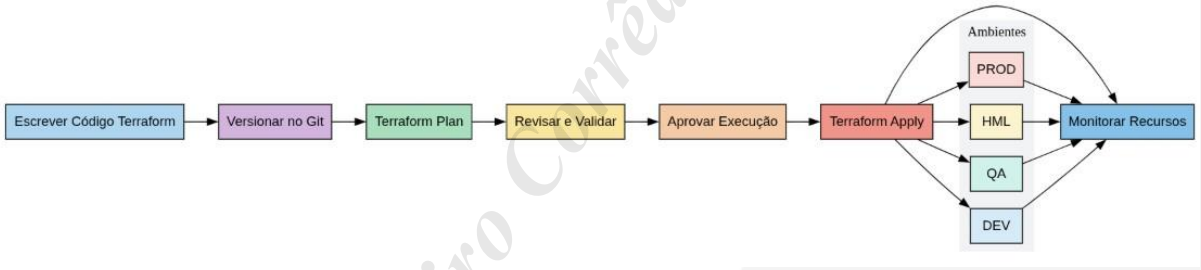


Figura 11: Fluxo para Terraform

O Terraform é uma ferramenta que cria e gerencia recursos de infraestrutura em diversos provedores de nuvem. Com ele, é possível versionar toda a definição do ambiente, aplicar mudanças de forma controlada e revisar o histórico de alterações. O uso de planos de execução permite saber exatamente o que será criado ou modificado antes de qualquer ação, o que dá previsibilidade ao processo.

O Ansible atua na configuração e no provisionamento do sistema operacional e de aplicações. Enquanto o Terraform cria a infraestrutura, o Ansible instala pacotes, copia arquivos, define permissões e executa scripts necessários para que o ambiente funcione corretamente. Esse trabalho conjunto garante que tanto os recursos quanto suas configurações sejam padronizados.

Uma das principais estratégias de gestão de infraestrutura é combinar Terraform e Ansible em fluxos automatizados, como pipelines de CI/CD. Isso permite que qualquer mudança seja revisada, testada e aplicada de forma controlada. Os benefícios incluem rastreabilidade, redução de tempo de provisio-

namento, maior segurança e possibilidade de recuperação rápida em caso de falhas, assim a gente consegue os seguintes benefícios:

- **Reprodutibilidade:** o mesmo código Terraform pode ser executado várias vezes para criar ambientes idênticos.
- **Idempotência:** aplicar o código Terraform repetidamente não altera o estado se nada foi modificado.
- **Construção por blocos**: a infraestrutura é descrita em blocos reutilizáveis como módulos, recursos e variáveis.
- **Evolutividade**: a infraestrutura pode ser modificada de forma incremental mantendo controle sobre as mudanças.

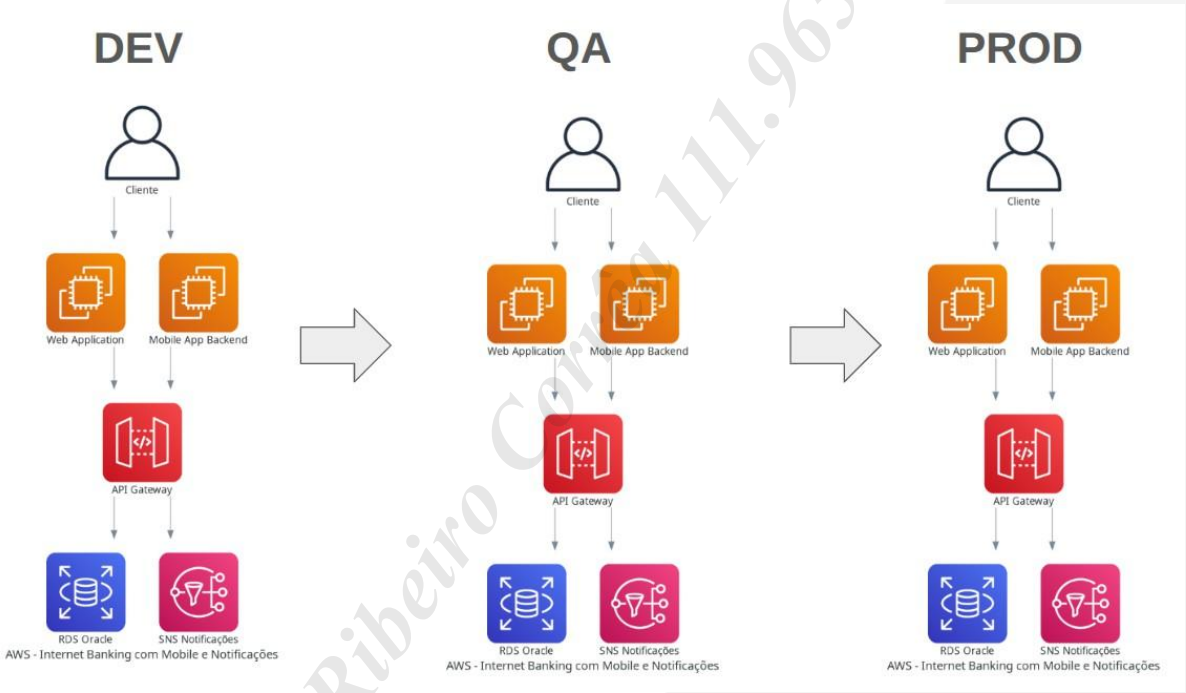


Figura 12: Construindo Ambientes com IaC

Conhecendo o Ansible

O Ansible é uma ferramenta de automação que permite gerenciar configuração, provisionamento e orquestração de servidores. Ele funciona de forma agentless, conectando-se por SSH e executando tarefas definidas em arquivos YAML chamados playbooks. Cada playbook descreve etapas que podem instalar pacotes, copiar arquivos, criar usuários e aplicar configurações de sistema.

Uma das principais possibilidades do Ansible é padronizar ambientes de forma reproduzível. Ele pode ser usado para configurar máquinas locais, servidores em nuvem ou clusters inteiros. Também permite integrar automação em pipelines de CI/CD e criar inventários dinâmicos que descrevem grupos de servidores de forma organizada.

O Ansible é uma ferramenta de automação que permite gerenciar servidores de forma remota e sem a necessidade de agentes instalados. Sua arquitetura básica é composta por quatro elementos principais: Control Node, Managed Nodes, Inventory e Modules.

Arquitetura Básica

O Ansible é uma ferramenta de automação que permite gerenciar servidores de forma remota e sem a necessidade de agentes instalados. Sua arquitetura básica é composta por quatro elementos principais: Control Node, Managed Nodes, Inventory e Modules.

- Control Node, é a máquina onde o Ansible está instalado e de onde os comandos são executados.
- Managed Nodes, são os servidores que serão gerenciados, geralmente acessados via SSH.
- Inventory, é um arquivo (por padrão hosts) que lista os Managed Nodes, podendo ser um simples arquivo INI, YAML ou uma fonte dinâmica. - Os Modules são as unidades de trabalho que executam tarefas específicas, como instalar pacotes, copiar arquivos ou reiniciar serviços.

Ansible Inventory

O Inventory no Ansible é o arquivo que define quais servidores serão gerenciados. Ele informa ao Ansible os nomes, endereços e grupos de máquinas-alvo. Pode ser escrito nos formatos INI ou YAML e permite organizar os servidores de forma lógica para facilitar a aplicação de tarefas.

No formato INI, é possível agrupar servidores com colchetes. Por exemplo:

```
[web]
web01 ansible_host=192.168.1.10
web02 ansible_host=192.168.1.11

[db]
db01 ansible_host=192.168.1.20
```

É possível definir variáveis por host ou por grupo, como o usuário SSH ou o nome do ambiente. Exemplo:

```
[web:vars]
ansible_user=ubuntu
env=dev
```

Esse Inventory pode ser usado em playbooks ou comandos diretos, como:

```
ansible web -m ping
ansible webservers -m apt -a "name=nginx state=present"
```

Esse comando testa a conectividade com todos os servidores do grupo web listados no Inventory.

Ansible Modules

Os modules no Ansible são blocos de funcionalidade que realizam tarefas específicas nos servidores gerenciados. Cada módulo executa uma ação, como copiar arquivos, instalar pacotes, gerenciar serviços ou executar comandos. Eles são usados em comandos diretos ou dentro de playbooks.

O módulo copy é usado para copiar arquivos do Control Node para os Managed Nodes:

```
- name: Copia um arquivo
  ansible.builtin.copy:
    src: /tmp/index.html
    dest: /var/www/html/index.html
```

O módulo file gerencia permissões, criação ou remoção de arquivos e diretórios:

```
- name: Garante que o diretório exista
  ansible.builtin.file:
    path: /var/log/app
    state: directory
    mode: '0755'
```

O módulo service controla serviços do sistema:

```
- name: Garante que o nginx esteja rodando
  ansible.builtin.service:
    name: nginx
    state: started
    enabled: yes
```

Para instalação de pacotes, pode-se usar package, yum (Red Hat) ou apt (Debian):

```
- name: Instala o nginx em sistemas Debian
  ansible.builtin.apt:
    name: nginx
```

```
    state: present
    update_cache: yes
```

Os módulos shell e command executam comandos diretamente no terminal remoto. command é mais seguro e não interpreta variáveis do shell:

```
- name: Executa um comando com shell
  ansible.builtin.shell: "echo $HOME >> /tmp/home.txt"
```

Esses módulos permitem que tarefas comuns sejam padronizadas, reutilizáveis e controladas por código, com idempotência garantida na maioria dos casos.

Ansible Roles

Roles no Ansible são uma forma de organizar o código em estruturas reutilizáveis e padronizadas. Cada role agrupa arquivos relacionados a uma função específica, como instalar um banco de dados ou configurar um servidor web. Isso facilita a manutenção, reutilização e leitura do código.

Uma role segue uma estrutura de diretórios específica, como:

```
roles/
  webserver/
    tasks/
      main.yml
    handlers/
      main.yml
    templates/
    files/
    vars/
      main.yml
```

Para usar uma role em um playbook, basta referenciá-la:

```
- hosts: web
  roles:
    - webserver
```

Dentro da role, o Ansible procura os arquivos padrão como `tasks/main.yml`, que define a lógica da automação.

Ansible Playbooks

Playbooks são arquivos YAML usados no Ansible para definir conjuntos de tarefas que serão executadas nos servidores gerenciados. Eles funcionam como receitas de automação, descrevendo o que deve ser feito, em qual máquina e em qual ordem. A estrutura básica de um playbook inclui três elementos

principais: `hosts` (alvo da automação), `tasks` (lista de ações) e `handlers` (tarefas acionadas por eventos, como reiniciar um serviço).

As `tasks` são executadas de forma sequencial, na ordem em que aparecem no arquivo. Cada task chama um módulo com seus parâmetros. Já os `handlers` são definidos separadamente e só são executados quando uma task anterior gera uma mudança e usa a instrução `notify`.

Exemplo prático: playbook para instalar e configurar um servidor web com Nginx em máquinas do grupo `web`:

```
- name: Instala de servidor web
  hosts: web
  become: yes

  tasks:
    - name: Instala o pacote nginx
      ansible.builtin.apt:
        name: nginx
        state: present
        update_cache: yes
      notify: Reinicia nginx

    - name: Copia página index.html
      ansible.builtin.copy:
        src: files/index.html
        dest: /var/www/html/index.html

  handlers:
    - name: Reinicia nginx
      ansible.builtin.service:
        name: nginx
        state: restarted
```

Esse playbook garante a instalação do Nginx, copia a página principal e reinicia o serviço apenas se houver mudança no pacote.

Desafios de Ansible

1. Realizar os exercícios de Ansible 01
2. Realizar os exercícios de Ansible 02
3. Realizar os exercícios de Ansible 03
4. Realizar os exercícios de Ansible 04

Conhecendo o Terraform

O Terraform é uma ferramenta de infraestrutura como código que permite criar, alterar e gerenciar recursos de forma declarativa. Com ele, toda a definição de ambientes fica registrada em arquivos de configuração que podem ser versionados. Esses arquivos descrevem componentes como redes, máquinas virtuais, bancos de dados e balanceadores de carga, em provedores de nuvem pública ou privada. Nas nossas aulas vamos utilizar um simulador das APIs da AWS.

A principal vantagem do Terraform é a capacidade de aplicar mudanças de forma controlada e previsível. Ele gera planos de execução que mostram o que será criado, modificado ou destruído antes de qualquer ação. Além disso, suporta o uso de módulos reutilizáveis, integração com pipelines de CI/CD e controle de estado para manter o ambiente alinhado com o código.

Comandos principais do Terraform:

- `terraform init` - Inicializa o diretório de trabalho e baixa os plugins necessários.
- `terraform plan` - Mostra as ações que serão executadas para atingir o estado desejado.
- `terraform apply` - Aplica as mudanças descritas no plano ao ambiente.
- `terraform destroy` - Remove todos os recursos gerenciados pelo código.
- `terraform validate` - Verifica se a sintaxe dos arquivos de configuração está correta.
- `terraform fmt` - Formata os arquivos de configuração de forma padronizada.
- `terraform show` - Exibe detalhes sobre o estado atual dos recursos.
- `terraform output` - Mostra os valores de saída definidos no código.

Terraform módulo

Um módulo Terraform é um conjunto de arquivos que define recursos reutilizáveis e organizados para provisionar infraestrutura. Ele pode ser usado localmente ou compartilhado entre projetos. A estrutura básica de um módulo inclui os seguintes arquivos e diretórios:

- `main.tf`: arquivo principal, onde os recursos são definidos.
- `variables.tf`: define as variáveis de entrada que o módulo recebe.
- `outputs.tf`: define os valores de saída que o módulo retorna.
- `terraform.tfvars` (opcional): define valores para variáveis.
- `providers.tf` (opcional no módulo, usado principalmente no root): define o provedor, como AWS, Azure, GCP.
- `README.md` (opcional): documentação do módulo.

Exemplo de estrutura:

```
my-module/
```

```
|-- main.tf
|-- variables.tf
|-- outputs.tf
```

Para usar um módulo, o diretório principal (root) referencia o caminho do módulo:

```
module "webserver" {
  source = "./modules/webserver"
  instance_type = "t2.micro"
  region        = "us-east-1"
}
```

Módulos ajudam a manter o código organizado, reutilizável e padronizado em ambientes com infraestruturas de app complexas, multi-conta ou multi-região.

Terraform criando um módulo

Um módulo Terraform é uma forma de organizar código reutilizável para provisionamento de infraestrutura. Cada módulo é composto por arquivos `.tf` que descrevem os recursos desejados. Para iniciar, crie uma pasta e adicione os arquivos principais: `main.tf`, `variables.tf` e `outputs.tf`.

Para usar um módulo pela primeira vez, execute os comandos na raiz do projeto:

- `terraform init`, inicializa o diretório
- `terraform plan`, mostra as ações que serão executadas
- `terraform apply`, aplica as mudanças e provisiona a infraestrutura

A criação de recursos é feita usando blocos `resource` em uma abordagem **declarativa**, onde se define o estado final desejado. O Terraform calcula a diferença entre o que existe e o que foi definido, e aplica apenas o necessário. Isso garante **idempotência**, ou seja, se rodar o mesmo código várias vezes, o resultado será sempre o mesmo.

O Terraform é **agnóstico de provedor** e funciona com AWS, Azure, GCP e outros. Por exemplo, para criar uma instância EC2 na AWS:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "web" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

Após a execução, é possível validar se o recurso foi criado com `terraform show` ou acessando diretamente o console do provedor (como o AWS Management Console). O estado atual da infraestrutura

é armazenado em um arquivo `terraform.tfstate`.

Terraform componentes básicos

No Terraform, alguns conceitos de apoio são fundamentais para a criação de infraestrutura reutilizável e organizada. Três deles são: **variables, outputs** e **providers**.

Variables permitem parametrizar valores como nomes, regiões ou tamanhos de recursos. Elas são definidas no arquivo `variables.tf`:

```
variable "region" {
  type    = string
  default = "us-east-1"
}
```

E são utilizadas no código com `${var.region}` ou `var.region`:

```
provider "aws" {
  region = var.region
}
```

Outputs são usados para expor informações úteis após a execução, como IPs ou nomes de recursos. Definidos no arquivo `outputs.tf`, por exemplo:

```
output "instance_ip" {
  value = aws_instance.web.public_ip
}
```

Providers são plugins que permitem ao Terraform interagir com diferentes plataformas, como AWS, Azure ou GCP. O bloco `provider` define a conexão e as configurações necessárias para usar um determinado provedor:

```
provider "aws" {
  region = var.region
}
```

A arquitetura básica do Terraform envolve arquivos `.tf`, variáveis, estados (armazenados no `terraform.tfstate`) e os providers que executam as ações. O ciclo padrão envolve `terraform init`, `plan` e `apply`, com todo o controle do que é criado ou modificado baseado no estado atual comparado ao estado desejado.

Terraform variáveis

O Terraform suporta diferentes tipos de variáveis para tornar o código reutilizável e parametrizado. Os tipos mais comuns são:

- **string**: representa um valor textual.
- **number**: representa valores numéricos.
- **bool**: representa valores booleanos (`true` ou `false`).
- **list** ou **tuple**: sequência ordenada de valores.
- **map** ou **object**: estrutura de chave-valor.

As variáveis são declaradas no arquivo `variables.tf`:

```
variable "region" {
  type    = string
  default = "us-east-1"
}
```

Os valores podem ser atribuídos de várias formas:

1. **Direto no código** com `default`.
2. **Arquivo** `.tfvars`: Exemplo `terraform.tfvars`:

   ```
   region = "us-west-2"
   ```

   Executar com:

   ```
   terraform apply -var-file="terraform.tfvars"
   ```

3. **Linha de comando** com o parâmetro `-var`:

   ```
   terraform apply -var="region=us-west-1"
   ```

4. **Variáveis de ambiente**:

   ```
   export TF_VAR_region=us-west-1
   ```

Variáveis sensíveis, como senhas e chaves, devem ser armazenadas com cuidado, preferencialmente fora do código-fonte. O Terraform oferece a opção `sensitive = true` para ocultar a exibição desses valores nos logs. Para casos mais seguros, recomenda-se o uso de ferramentas como Vault ou integração com gerenciadores de segredos da nuvem.

Terraform criando recursos

O uso básico do Terraform envolve a criação de recursos declarando o estado desejado em arquivos .tf. O ciclo de uso é sempre o mesmo:

```
terraform init
terraform plan
terraform apply
```

```
terraform destroy
```

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "web" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

```
terraform init
terraform apply
```

Terraform criando multiplos ambientes

Para organizar um projeto Terraform com **módulo reutilizável** e separação por ambientes (DEV, QA, PROD), é comum manter um único módulo genérico e criar diretórios separados para cada ambiente com seus próprios arquivos `.tfvars`.

A estrutura básica fica assim:

```
infra/
-- modules/
   -- app/
      -- main.tf
      -- variables.tf
      -- outputs.tf
-- dev/
   -- main.tf
   -- dev.tfvars
-- qa/
   -- main.tf
   -- qa.tfvars
-- prod/
   -- main.tf
   -- prod.tfvars
```

O conteúdo de `main.tf` em cada ambiente aponta para o módulo:

```
module "app" {
  source        = "../modules/app"
  instance_type = var.instance_type
  region        = var.region
}
```

O arquivo `.tfvars` de cada ambiente define os valores das variáveis:

Exemplo `dev/dev.tfvars`:

```
instance_type = "t2.micro"
region        = "us-east-1"
```

Para aplicar a infraestrutura de um ambiente, use:

```
cd dev
terraform init
terraform apply -var-file="dev.tfvars"
```

Essa organização separa código comum (no módulo) da configuração específica de cada ambiente, facilitando manutenção e controle.

Para aplicar um script Terraform somente em um ambiente

- Para aplicar só para DEV:

```
cd infra/dev
terraform init
terraform apply -var-file="dev.tfvars"
```

- Para aplicar só para QA:

```
cd infra/qa
terraform init
terraform apply -var-file="qa.tfvars"
```

Desafios de Terraform

1. Realizar os exercícios de Terraform 01
2. Realizar os exercícios de Terraform 02
3. Realizar os exercícios de Terraform 03
4. Realizar os exercícios de Terraform 04

GOVERNANÇA COMO CÓDIGO

Governança como Código representa a evolução natural dos processos de TI, onde políticas, regras de compliance e controles de segurança são definidos por meio de código versionado e executado automaticamente. Este conceito transforma práticas manuais de auditoria e controle em processos automatizados que podem ser testados, revisados e aplicados de forma consistente em toda a infraestrutura. O Terraform e o Ansible e outras ferramentas de IaC se tornaram ferramentas fundamentais neste contexto porque permitem codificar não apenas a infraestrutura, mas também as regras que governam como essa infraestrutura deve ser criada, configurada e mantida.

Utilizar IaC na Governança das empresas permite que padrões de segurança, compliance e boas práticas operacionais sejam aplicados de forma automática e de forma consistente em todos os ambientes. Dessa forma, é possível manter a consistência dos ambientes por meio de versionamento de configurações, rastrear alterações e garantir que cada recurso criado siga regras pré-definidas de acesso, auditoria e controle de custos. Isso reduz riscos, facilita auditorias e melhora a capacidade de manter o ambiente alinhado às políticas corporativas sem depender de processos manuais.

Políticas como Código

Políticas como Código é a prática de definir regras e controles de governança utilizando linguagens declarativas, armazenando essas regras como código versionado. Essa abordagem permite que políticas sejam integradas ao ciclo de vida da infraestrutura, tornando possível sua aplicação e validação de forma automática. Com isso, as regras deixam de ser documentos informais e passam a ser componentes ativos da automação.

A implementação envolve a adoção de frameworks de governança que definem responsabilidades, papéis, níveis de controle e mecanismos de auditoria. Modelos como COBIT e ITIL são utilizados como referência para estruturar esses processos. Na prática, essas diretrizes são traduzidas em políticas como código, utilizando ferramentas como Terraform Sentinel, Open Policy Agent (OPA) ou Cloud Custodian, que integram esses controles diretamente aos pipelines de provisionamento.

Entre os benefícios estão a possibilidade de testar, versionar e aplicar as políticas de maneira contínua e confiável. Exemplos práticos incluem a restrição de tipos de instância por ambiente (ex: evitar instâncias grandes em ambientes de desenvolvimento), exigência de tags obrigatórias (como "owner" e "cost-center") e validação de configurações de segurança (como criptografia habilitada ou uso de portas seguras). Isso garante conformidade e padronização sem depender de revisões manuais.

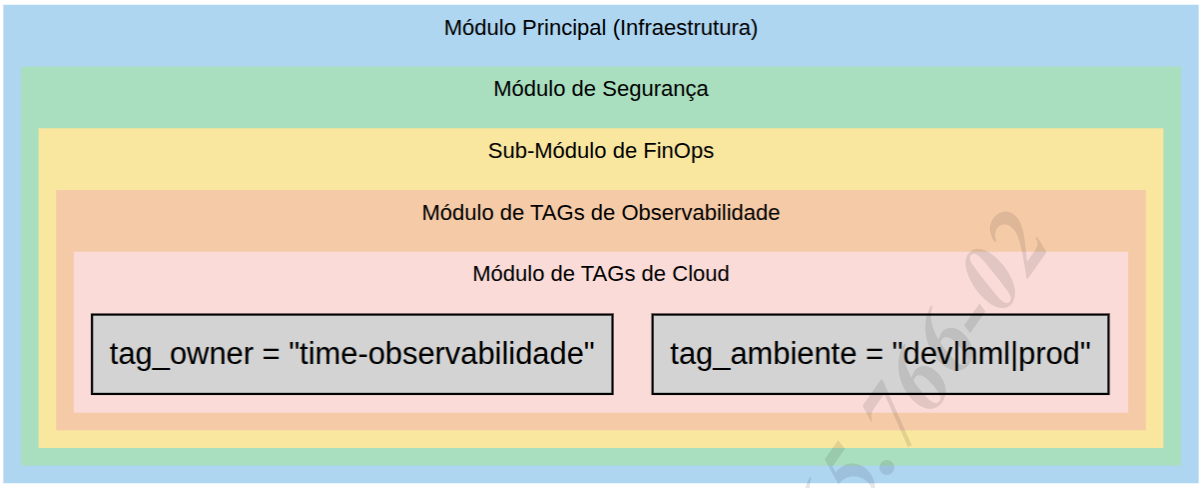


Figura 13: Governança as code

Compliance e auditoria

Compliance e auditoria automatizada como código consistem na prática de garantir conformidade com normas, políticas e boas práticas por meio de código executável. Em vez de processos manuais e esporádicos, utiliza-se scripts e ferramentas automatizadas que realizam verificações contínuas. A lógica de conformidade é registrada como código versionado, o que permite padronização, rastreabilidade e integração com pipelines de CI/CD. Isso transforma a governança em parte do ciclo de vida da aplicação, promovendo verificações consistentes desde o desenvolvimento até a produção.

A implementação envolve três pilares principais: scans automatizados de vulnerabilidades, que detectam falhas de segurança em tempo real; verificação de padrões de arquitetura, que garante aderência a modelos aprovados pela organização; e relatórios automáticos de compliance, que geram evidências rastreáveis para auditorias internas e externas. Os principais benefícios incluem a detecção precoce de violações e a geração contínua de evidências auditáveis, o que reduz riscos e facilita a resposta a incidentes e auditorias.

Controle de custos e otimização

O controle de custos e otimização como código é a prática de aplicar governança financeira por meio de políticas automatizadas e integradas ao ciclo de vida da infraestrutura. Essa abordagem permite definir budgets e alertas como código, promovendo rastreabilidade e automação no controle de gastos. Também permite a aplicação de políticas de right-sizing automático, ajustando recursos ao consumo real, e o uso de schedules para desligamento e ativação de ambientes conforme sua finalidade, como desligar ambientes de desenvolvimento fora do horário comercial.

Ferramentas como o AWS Cost Anomaly Detection, Azure Cost Management APIs e Cloud Custodian permitem aplicar essas práticas em diferentes provedores. As ações incluem a prevenção de recursos superdimensionados, o desligamento automático de ambientes não-produtivos e até a análise de impacto de custo diretamente em pull requests. Essa integração financeira com a automação operacional permite que as equipes tomem decisões informadas e evitem desperdícios, mantendo o ambiente em conformidade com limites orçamentários definidos.

Desafios de Governança

1. Pesquisar sobre os frameworks de governança ITIL3 e COBIT

SEGURANÇA COMO CÓDIGO

Segurança como código é a prática de tratar controles de segurança como parte do mesmo processo de automação da infraestrutura. Isso significa que grupos de segurança, regras de rede, políticas de identidade e permissões são descritos em arquivos de configuração versionados. Essa abordagem garante que a segurança não seja aplicada de forma manual e isolada, mas integrada ao ciclo de vida da infraestrutura.

No dia a dia, ferramentas como Terraform e Ansible permitem definir regras de firewall, criptografia de componentes, canais de comunicação e controle de acesso. Por exemplo, é possível criar módulos que sempre bloqueiam portas específicas por padrão, habilitam logs de auditoria e aplicam criptografia para determinados tipos de componentes. Assim, toda vez que um recurso é criado, ele já nasce com os controles de segurança definidos e revisados pelo time, as práticas a seguir podem ser implementadas por meio de IaC, são elas:

- Criação de grupos de segurança e regras de firewall, definindo quais portas estão abertas e para quais origens, garantindo bloqueio por padrão.
- Aplicação de políticas de ciclo de vida de chaves, provisionando o gerenciamento automático de rotação e expiração de chaves de criptografia.
- Automação de patches e atualizações de sistema, usar playbooks ou módulos para garantir que sistemas operacionais estejam atualizados.

Um outro ponto importante é que segurança como código facilita auditoria e rastreabilidade. Cada mudança em regras e permissões fica registrada no histórico do código, permitindo revisar quem alterou e por que motivo. Além disso, é possível integrar essas definições a pipelines de CI/CD, para que validações automáticas identifiquem configurações fora do padrão antes da aplicação no ambiente.

Segurança Integrada ao Desenvolvimento

Security by Design, ou Segurança Integrada no Desenvolvimento, é a prática de incorporar controles de segurança desde as fases iniciais do ciclo de vida da aplicação. Em vez de tratar segurança como uma etapa final ou separada, as equipes integram requisitos e validações desde o planejamento e desenvolvimento da infraestrutura. Isso inclui o uso de políticas codificadas diretamente nos templates de infraestrutura, aplicação de configurações seguras por padrão (secure defaults) e validação automática das configurações com base em benchmarks de segurança reconhecidos, como CIS.

No contexto de pipelines de CI/CD, testes de segurança devem ser executados automaticamente como parte das validações. Ferramentas de análise estática (SAST), análise dinâmica (DAST) e validação de configurações podem ser integradas aos pipelines para detectar falhas em tempo de desenvolvimento.

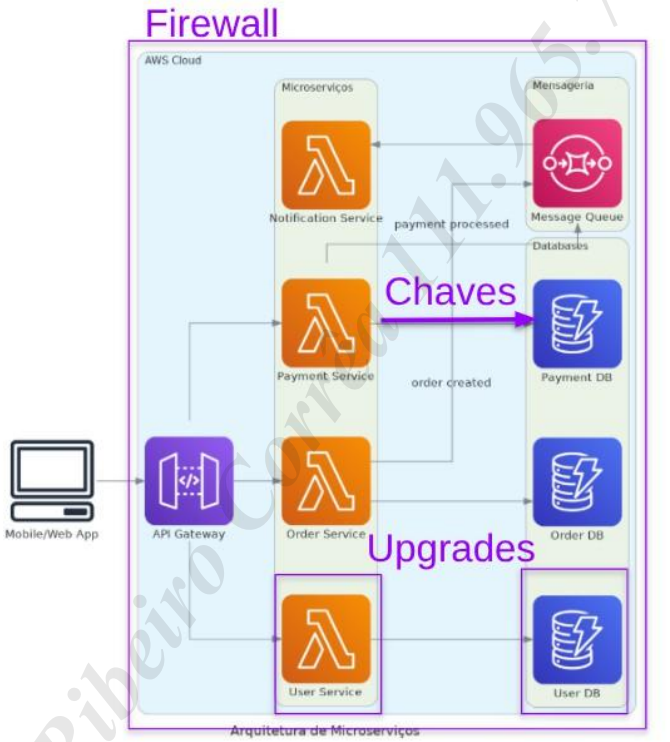


Figura 14: Segurança as code

Isso reduz o retrabalho e acelera a entrega de sistemas mais seguros. Além disso, a prática de Infrastructure as Code permite aplicar os mesmos controles de segurança de forma consistente em todos os ambientes, com versionamento e rastreabilidade.

A segurança aplicada como código inclui definições explícitas de acesso com privilégio mínimo (least privilege access), configurações automáticas de criptografia em repouso e em trânsito, e segmentação de rede declarada por código. Isso permite que os ambientes atendam requisitos de conformidade e segurança de forma padronizada, auditável e escalável. Com essas práticas, a segurança deixa de ser um processo manual e passa a ser parte integrada da automação e da entrega contínua.

Segurança Shift-Left

Shift-Left Security é uma abordagem que move os testes de segurança para as fases iniciais do ciclo de desenvolvimento, com o objetivo de detectar vulnerabilidades o mais cedo possível. Isso é feito integrando práticas como SAST (análise estática de código fonte), verificação de dependências vulneráveis, varredura por segredos em repositórios e análise de imagens de containers antes da implantação. Essas ações permitem que falhas sejam identificadas antes mesmo do código ser mesclado ao repositório principal.

A implementação efetiva dessa abordagem exige a criação de um pipeline de segurança estruturado. Isso inclui o uso de *pre-commit hooks* para validações locais, análise automática de segurança em *pull requests* e a aplicação de *gates* que bloqueiam deployments inseguros. O feedback precisa ser rápido e integrado ao ambiente de desenvolvimento, permitindo que o próprio desenvolvedor corrija o problema sem depender de etapas posteriores ou de equipes externas. Com isso, a segurança se torna parte natural do fluxo de trabalho.

Ferramentas específicas suportam esse processo: - SonarQube permite análise de código para detectar padrões inseguros. - Snyk faz análise de dependências de bibliotecas em busca de vulnerabilidades conhecidas. - Clair e Trivy são usados para escanear imagens de containers em busca de falhas de segurança.

A aplicação dessas ferramentas reduz o custo de correção e o tempo de resposta, ao evitar que falhas cheguem às fases finais do ciclo de vida ou à produção.

Compliance as Code, Auditoria e Conformidade Automatizada

Compliance as Code é a prática de implementar requisitos regulatórios e padrões de conformidade por meio de código executável. Em vez de auditorias manuais e pontuais, a conformidade é verificada continuamente com base em políticas codificadas, como as exigidas por padrões internacionais como:

SOC 2, PCI-DSS e GDPR. Essas políticas são aplicadas diretamente na infraestrutura como código e nos pipelines de entrega, promovendo controle automatizado, rastreável e reproduzível.

Essa abordagem permite auditoria contínua, reduz o tempo de preparação para auditorias externas e mitiga riscos operacionais e regulatórios. A automação garante que a infraestrutura permaneça em conformidade mesmo em ambientes dinâmicos e com mudanças frequentes, eliminando a necessidade de verificações manuais recorrentes e promovendo um estado permanente de prontidão para inspeções externas.

Desafio Segurança como código

- Pesquisar sobre o uso e funcionamento da ferramenta SonarQube
- Pesquisar sobre o uso e funcionamento da ferramenta Snyk
- Pesquisar sobre o uso e funcionamento da ferramenta Trivy
- Pesquisar como incluir o modelo as técnicas de Shift-Left no fluxo de CI/CD
- Pesquisar sobre o padrão SOC 2
- Pesquisar sobre o padrão PCI-DSS

FINOPS

A área de FinOps tem como objetivo principal gerenciar custos de nuvem com o objetivo claro de reduzir desperdícios e gerar redução de custos. Faz parte deste trabalho definir políticas de otimização de custos, orçamentos e alertas de gastos. O conceito surgiu da necessidade de tratar a governança financeira de nuvem com a mesma disciplina aplicada à infraestrutura, buscando implementem controles de custo durante o processo de desenvolvimento e não apenas após o deploy.

A integração entre FinOps e IaC elimina a gestão manual das políticas de custo e cria um ciclo contínuo de otimização financeira. A ideia principal é que as políticas de redução de custos de nuvem seja tratado como um aspecto configurável pelos times de engenharia e dessa forma seja possível alinhar governança financeira com atuação de times técnicos.

Visibilidade e Alocação de Custos

O pilar de Visibilidade e Alocação de Custos em FinOps tem como objetivo principal permitir que as equipes entendam com precisão para onde os recursos financeiros estão sendo direcionados na nuvem. Para isso, é essencial aplicar práticas como o uso padronizado de *tags* e convenções de nomenclatura, permitindo a categorização dos recursos por time, projeto, ambiente ou centro de custo. Essa identificação estruturada é o primeiro passo para uma alocação precisa e para análises posteriores.

Além da categorização, a visibilidade depende da integração com ferramentas nativas dos provedores de nuvem, como AWS Cost Explorer, Azure Cost Management e GCP Billing Reports. Essas ferramentas fornecem relatórios detalhados e permitem a consolidação das informações de uso e custo. Para facilitar o acompanhamento diário, é recomendada a criação de *dashboards* que exibam o consumo por unidade de negócio ou aplicação, com dados em tempo quase real. Isso permite que decisões sejam tomadas com base em dados atualizados e alinhadas com os limites orçamentários definidos.

Otimização Contínua de Recursos

A Otimização Contínua de Recursos em FinOps tem como objetivo garantir o uso eficiente da infraestrutura em nuvem, evitando desperdícios e mantendo a performance adequada. A principal prática é o *right-sizing*, que consiste no ajuste automático ou manual da capacidade de instâncias, bancos de dados e demais serviços, com base no uso real. Esse processo pode ser feito de forma recorrente, utilizando métricas de utilização para redimensionar recursos sob ou superdimensionados.

Além do ajuste de tamanho, a escolha do modelo de contratação influencia diretamente no custo. O uso de instâncias reservadas, savings plans ou spot instances deve ser alinhado ao perfil da carga —

estática, previsível ou volátil. Outra prática importante é o agendamento de desligamento de ambientes não-produtivos, como homologação e desenvolvimento fora do horário comercial, e a eliminação de recursos ociosos, como discos não utilizados ou IPs públicos alocados sem necessidade. Essas ações tornam o ambiente mais enxuto e economicamente sustentável.

Governança Financeira Automatizada

Governança Financeira Automatizada em FinOps tem como objetivo estabelecer políticas que previnam o uso indevido de recursos e garantam controle contínuo dos gastos. A primeira etapa é a definição de *budgets* e alertas automatizados, permitindo que equipes sejam notificadas quando os custos se aproximam ou excedem os limites definidos. Essa prática viabiliza respostas rápidas a desvios e ajuda na manutenção do orçamento planejado.

A automação dessas regras é feita por meio de políticas como código, utilizando ferramentas como Cloud Custodian ou Open Policy Agent (OPA). Essas políticas podem bloquear a criação de recursos sem tags obrigatórias, restringir o uso de serviços fora do padrão ou aplicar regras de desligamento automático. Além disso, é possível incorporar a revisão de impacto de custo nos *pull requests*, garantindo que cada nova infraestrutura proposta passe por validação financeira antes da aprovação e do provisionamento. Isso integra a responsabilidade financeira diretamente ao fluxo de trabalho de engenharia. A utilização de

No dia a dia, ferramentas como Terraform e Ansible podem ser usadas para criar ou ajustar recursos já criados como políticas de custo. Essas ferramentas também nos permite definir/criar/ajustar tags de projeto, unidade de negócios e ambientes, para cada componente criado, associar contas a centros de custo e habilitar alertas de orçamento. Vejamos algumas outras estratégias de FinOps que podem ser implementadas por meio de IaC:

- Aplicação de tags de custo obrigatórias, definir, via código, tags como `owner`, `environment`, `cost_center` e `project` em todos os recursos criados.
- Definição de limites de uso e cotas por ambiente, criando políticas que limitam quantidade de CPUs, memória, instâncias ou volume de armazenamento por conta, projeto ou região.
- Desligamento automático de ambientes não produtivos, agendar, via código, a criação de jobs ou funções serverless que desligam ambientes DEV/QA fora do horário comercial.
- Identificação e remoção de recursos órfãos, usar código e automações para identificar e excluir recursos sem tags ou sem uso por um período definido.

Utilizar IaC na gestão de FinOps permite que práticas de controle de custo sejam aplicadas de forma massiva, padronizada, automática e auditável em todos os ambientes de empresa. Essa integração

torna o gerenciamento dos custos parte do processo de provisionamento, evitando ações manuais e melhorando a eficiência operacional.

O FUTURO DA IaC

Caras alunas e caros alunos, chegamos ao final da nossa disciplina. Espero que tenham aproveitado o conteúdo e que possam aplicar esses conhecimentos em suas trajetórias profissionais. Gostaria de deixar uma última contribuição. Sem a intenção de prever o futuro, recomendo que se mantenham atentos a alguns campos de conhecimento que podem abrir portas para novas oportunidades de trabalho.

Percebo que algumas áreas do mercado de TI apresentam grande potencial de crescimento e oferecem diversas possibilidades de atuação, são elas:

- Arquitetura de soluções, caminha para a integração completa entre IaC e FinOps, criando um ecossistema onde decisões técnicas e financeiras acontecem simultaneamente.
- Projetar sistemas considerando performance, escalabilidade e custo como disciplinas interdependentes desde o primeiro momento.
- Engenharia de custos de cloud, pessoas que conhecem o funcionamentos de componentes que podem sugerir componentes e otimizações para reduzir custos.
- Expertise em ferramentas de IaC como (Terraform, CloudFormation, Pulumi) criam um diferencial competitivo significativo.
- Gestão automatizadas de ambientes das aplicações por meio de pipelines de configuração e gestão.
- IaC com AI, é um campo muito recente com muitas oportunidades para surgir.

Desejo muita sorte em sua jornada!!

Abraços

Prof. Leonardo Martins

REFERÊNCIAS

MORRIS, KIEF. 2021. "INFRASTRUCTURE AS CODE : dynamic systems for the cloud age".