AWS CloudFormation

Conceito Fundamental

O AWS CloudFormation é um serviço da AWS que permite criar e gerenciar recursos de infraestrutura como código (IaC — Infrastructure as Code).
Você descreve sua infraestrutura (VPCs, EC2, S3, IAM, Lambda, etc.) em arquivos de texto (YAML ou JSON) chamados templates, e o CloudFormation cria, atualiza e deleta tudo automaticamente conforme essa configuração.

- Sem CloudFormation, você cria tudo manualmente pelo console. Com ele, você automatiza, versiona e replica sua infraestrutura de forma previsível e reprodutível.

Objetivo do CloudFormation

- Automatizar a criação e configuração de recursos AWS.
- Garantir consistência entre ambientes (dev, teste, produção).
- Reduzir erros humanos e o tempo de deploy.
- Integrar infraestrutura ao controle de versão (Git).

- O CloudFormation é a “receita” da sua infraestrutura — a AWS se encarrega de “cozinhar” (provisionar) os recursos para você.

Conceitos-Chave

Conceito	*Descrição*
Template	Arquivo YAML/JSON que define os recursos.
Stack	Conjunto de recursos criados a partir de um template.
Change Set	Prévia das mudanças que serão aplicadas a uma stack existente.
Parameters	Valores configuráveis passados ao template.

Outputs	Informações retornadas após a criação (como endpoints ou IDs).
Resources	A parte principal do template — define o que será criado.
Mappings/Conditions	Usados para lógica condicional (por exemplo, região ou ambiente).

→ O template é a definição, o stack é a instância em execução.
→ É como um *“molde”* e o *“objeto criado a partir dele”*.

Estrutura Básica de um Template

Exemplo simples (YAML):

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Exemplo de criação de uma instância EC2"

Parameters:
  InstanceType:
    Type: String
    Default: t2.micro
    Description: Tipo da instância EC2

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: !Ref InstanceType
      ImageId: ami-0abcdef1234567890
      Tags:
        - Key: Name
          Value: ExemploCF

Outputs:
  InstanceId:
    Description: ID da instância criada
    Value: !Ref MyEC2Instance
```

- O template é declarativo — você diz *“o que quer”*, e não *“como fazer”*.

Fluxo de Funcionamento

1. Crie o template (YAML/JSON).
2. Envie-o para o CloudFormation (pelo Console, CLI ou API).
3. O serviço cria um Stack com os recursos definidos.
4. Você pode:
    - Atualizar o stack (Change Set mostra o que mudará).
    - Excluir o stack (deleta todos os recursos juntos).

- CloudFormation mantém o controle de estado — ele sabe o que já existe e o que precisa mudar. Isso é essencial para infraestruturas versionadas.

Benefícios Práticos

- Reprodutibilidade: mesmo template → mesma infraestrutura em qualquer região.
- Controle de versão: templates podem ser armazenados no Git.
- Automação: integração com CI/CD (CodePipeline, GitHub Actions, Jenkins).
- Rollback automático: se algo falhar na criação, tudo é revertido.
- Sem custo adicional: você só paga pelos recursos criados.

- Isso é o núcleo da infraestrutura como código — replicar e versionar ambientes de forma confiável.

Parâmetros e Variáveis Dinâmicas

Permitem tornar o template flexível e reutilizável:

```
Parameters:
  EnvironmentType:
    Type: String
    AllowedValues: [dev, prod]
```

Default: dev

- Você pode criar um único template e usá-lo para vários ambientes apenas alterando parâmetros.

Integrações e Reuso

- Nested Stacks: permite quebrar grandes templates em partes reutilizáveis.
- Modules: componentes prontos que encapsulam recursos (ex: um módulo de VPC).
- AWS CDK (Cloud Development Kit): alternativa programável ao CloudFormation (gera templates automaticamente em Python, TypeScript, etc.).

- Use Nested Stacks ou CDK para evitar duplicação e tornar grandes infraestruturas mais gerenciáveis.

Gerenciamento e Monitoramento

- Console AWS: visualiza o status de criação e logs de erro.
- CloudWatch: coleta métricas e eventos das stacks.
- Stack Events: histórico detalhado do que foi criado, atualizado ou falhou.
- Drift Detection: identifica se algum recurso foi modificado fora do CloudFormation.

- Evite alterar recursos manualmente — isso causa *drift*, ou seja, divergência entre o template e o ambiente real.

Boas Práticas

→ Separar templates por função: rede, segurança, aplicação, etc.
→ Usar parâmetros e outputs de forma padronizada.
→ Versionar templates no Git.
→ Testar alterações com Change Sets antes de aplicar.

→ Evitar dependências circulares (um recurso depender de outro).
→ Adicionar descrições e tags em tudo.
→ Validar templates com aws cloudformation validate-template.

- CloudFormation é mais confiável que scripts de CLI manuais — ele entende as dependências e ordem correta de criação automaticamente.

Casos de Uso Comuns

→ Criação de ambientes completos (dev/test/prod).
→ Deploy automatizado de aplicações web.
→ Infraestrutura de Data Lake (S3 + Glue + Athena).
→ Pipelines CI/CD com CodePipeline.
→ Gerenciamento de políticas IAM e configurações de segurança.

- Com o CloudFormation, você pode “recriar” sua infraestrutura inteira a partir do zero — ideal para disaster recovery e ambientes padronizados.