

Fundamentos de DevOps e SRE



Sobre Mim

Manoel Carlos Martins Mineiro Junior

- 30 anos de idade
- 11 anos de experiência em TI
- Apaixonado por um bom churrasco
- Bacharel em Ciências da Computação
- MBA em Arq. de Redes e Cloud Computing
- Certificações em Cloud, SRE, IaC e Kubernetes
- Atualmente Staff SRE no Itaú Unibanco
- Professor na FullCycle



Agenda

- Introdução
- Integração Contínua (CI) e Entrega Contínua (CD)
- Práticas de Desenvolvimento para infraestrutura
- SRE princípios e práticas
- Observabilidade

Introdução



Overview do módulo

- A cultura Devops
- Fundamentos de SRE
- Diferenças entre Devops e SRE

Cultura DevOps

DevOps é uma cultura e conjunto de práticas que visam integrar os times de desenvolvimento de software (Dev) e operações de infraestrutura (Ops). O objetivo é promover uma colaboração mais estreita entre esses dois grupos, automatizar processos de desenvolvimento, testes e implantação, e criar um ambiente de entrega contínua e infraestrutura ágil. Em essência, o DevOps busca acelerar o ciclo de vida do desenvolvimento de software (SDLC), melhorar a qualidade do código e aumentar a eficiência operacional.

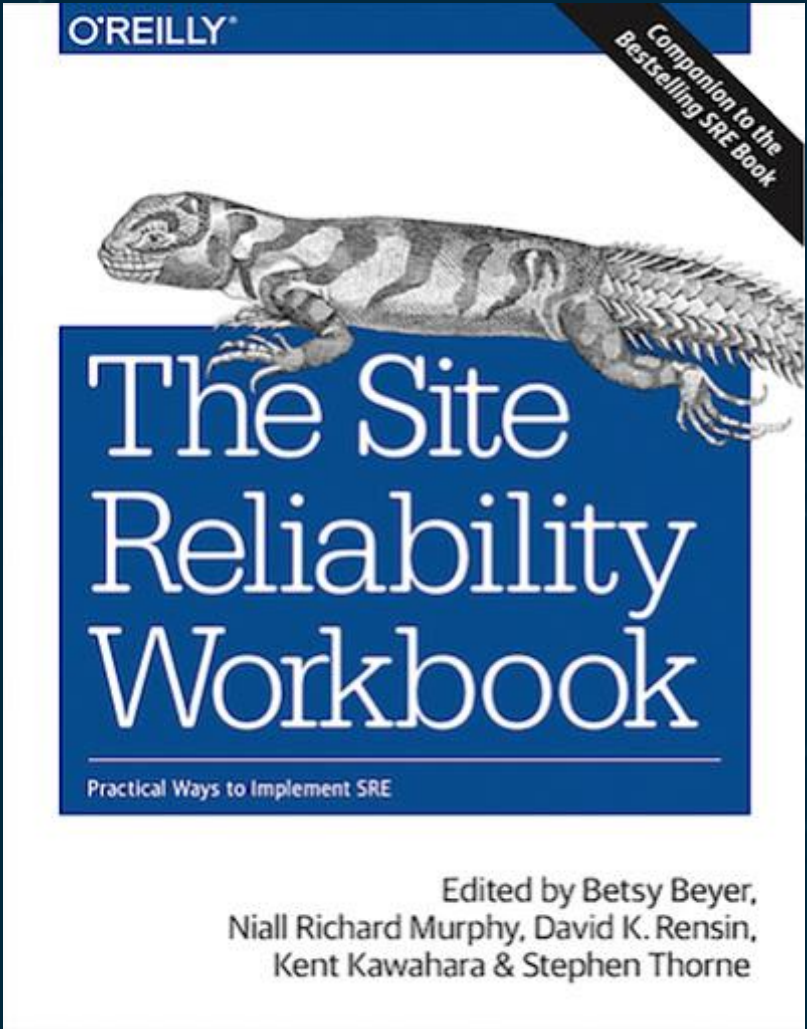
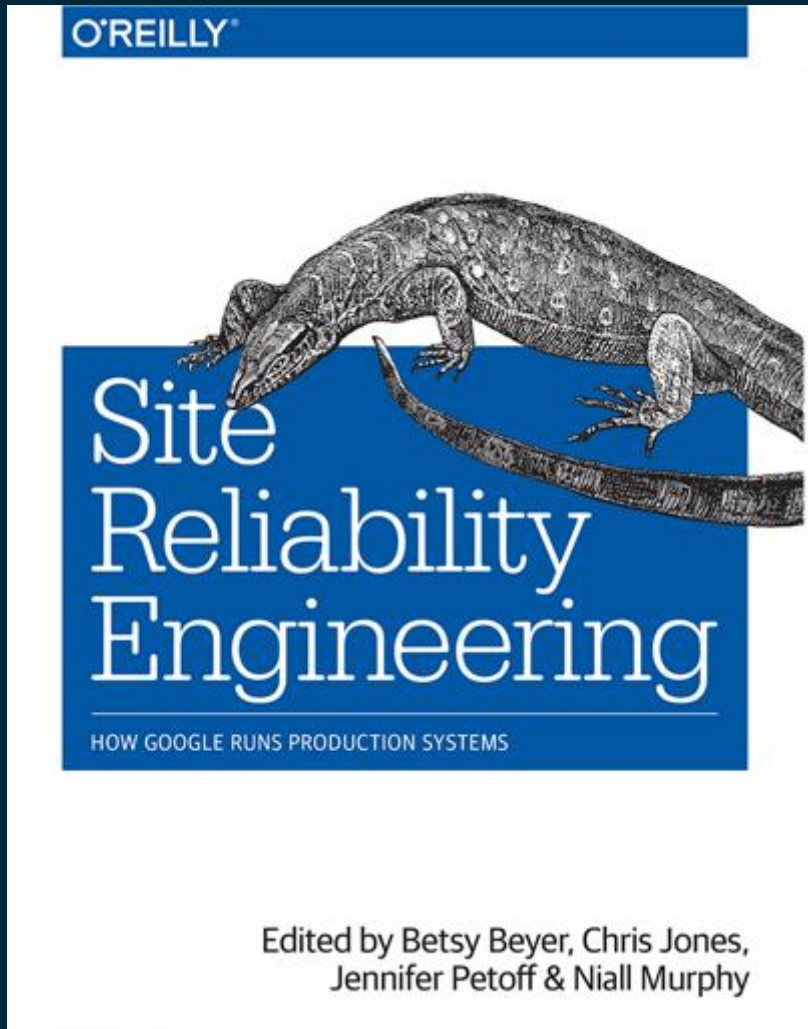
Cultura DevOps - CALMS

- Cultura
- Automação
- Lean
- Medição
- Compartilhamento (Sharing)

Fundamentos de SRE

Site Reliability Engineering (SRE) é um termo desenvolvido por Ben Treynor Sloss, vice-presidente de engenharia do Google criado por volta de 2003. SRE é uma disciplina que incorpora aspectos da engenharia de software e os aplica a problemas de infraestrutura e operações. Enquanto o DevOps é uma filosofia sobre colaboração entre operações e desenvolvimento, o SRE é mais concreto como uma função, focando na criação de sistemas altamente confiáveis e escaláveis através de automação e engenharia de software. Assim, podemos dizer que a classe SRE implementa a interface DevOps. SRE foi amplamente difundido por meio dos livros e adoção de empresas como o próprio Google, Netflix e Spotify.

Fundamentos de SRE



Fundamentos de SRE

- Operações são um problema de software
- Orientação a Níveis de Serviço
- Redução de Toil
- Automação
- Reduzir o custo das falhas
- Responsabilidade compartilhada

DevOps x SRE

Devops é um conjunto de práticas, diretrizes e cultura projetada para quebrar silos em desenvolvimento, operações, arquitetura, rede e segurança.

SRE é uma disciplina que incorpora aspectos da engenharia de software e os aplica a problemas de infraestrutura e operações.

Objetivo Principal: Promover uma cultura de colaboração entre equipes de desenvolvimento (Dev) e operações (Ops) para melhorar a entrega de software.

DevOps

Foco: Abrange todo o ciclo de vida do desenvolvimento de software, desde a codificação até a operação.

Principais Práticas: Automação, colaboração, integração contínua, entrega contínua, monitoramento e feedback rápido.

Cultura: Destaca a importância da colaboração, comunicação e responsabilidade compartilhada entre as equipes de desenvolvimento e operações.

Objetivo Principal: Garantir a confiabilidade e disponibilidade dos sistemas de software, concentrando-se em práticas e ferramentas específicas para atingir esses objetivos.

Foco: Prioriza a confiabilidade, escalabilidade e desempenho dos sistemas em produção.

Principais Práticas: Definição de metas de serviço (SLOs), monitoramento proativo, gerenciamento de incidentes, automação de tarefas operacionais e

engenharia de confiabilidade.

SRE

Cultura: Promove a automação, padronização e melhoria contínua dos processos operacionais para garantir a confiabilidade do sistema.

Diferenças entre DevOps e SRE

Semelhanças

Cultura de Colaboração: Ambas enfatizam a importância da colaboração entre equipes de desenvolvimento e operações.

Automação: Tanto DevOps quanto SRE valorizam a automação de processos repetitivos e tarefas operacionais.

Feedback Rápido: Ambas as abordagens priorizam o feedback rápido e a melhoria contínua dos processos.

Diferenças

Escopo: DevOps abrange todo o ciclo de vida do desenvolvimento de software, enquanto SRE se concentra especificamente na operação e confiabilidade dos sistemas.

Objetivos Primários: DevOps visa melhorar a entrega de software, enquanto SRE visa garantir a confiabilidade e disponibilidade dos sistemas.

Práticas Específicas: Embora compartilhem algumas práticas, como automação e colaboração, DevOps e SRE têm práticas específicas que se concentram em diferentes aspectos do desenvolvimento e operação de software.

Integração Contínua (CI) e Entrega Contínua (CD)



Overview do módulo

- SDLC (Software Development Life Cycle)
- Continuous Integration
- Continuous Delivery e Continuous Deployment
- Abordagens de testes automatizados
- DevSecOps
- Pipeline de container

Software Delivery Life Cycle (SDLC)

O Ciclo de Vida do Desenvolvimento de Software (SDLC) contempla todas as etapas do processo de desenvolvimento, desde a concepção da ideia até a manutenção do produto final. Cada etapa é cuidadosamente planejada e executada para garantir que o software atenda aos requisitos funcionais, qualidade, prazo e dentro do budget.



Continuous Integration

Continuous Integration (CI) é uma prática de desenvolvimento que usa etapas automatizadas de criação e de teste para fornecer pequenas alterações na aplicação de maneira confiável. A cada integração é verificada uma série de etapas de forma automatizada, permitindo que as equipes detectem erros mais rapidamente. CI tem como objetivo otimizar o processo de desenvolvimento, melhorar a qualidade do código e reduzir o tempo necessário para entregar atualizações. Com CI é possível promover a colaboração entre os membros da equipe e garantir que a base de código esteja sempre em um estado funcional, além de permitir ciclos de feedback mais rápidos possibilitando que as equipes identifiquem e corrijam problemas no início do desenvolvimento.

Continuous Integration

Os principais benefícios da Integração Contínua incluem:

- Mudanças pequenas e gradativas no código
- Isolamento de falhas
- Agilidade no desenvolvimento
- Detecção antecipada de Problemas
- Melhoria da Qualidade de Software
- Redução de riscos
- Facilidade na manutenção e atualização

Continuous Delivery

Continuous Delivery (CD) é uma abordagem no desenvolvimento de software que busca automatizar e simplificar o processo de entrega de código em ambientes produtivos de forma rápida, segura e confiável. Ao contrário da Integração Contínua que se concentra em etapas do desenvolvimento, o principal objetivo do contínuos delivery é permitir que as equipes entreguem alterações de código em produção de maneira eficiente e com baixo risco, garantindo que o software possa ser implantado, de forma automatizada e sem problemas sendo requerido apenas alguma aprovação manual.

Continuous Deployment

Continuous Deployment (CD) assim como continuous delivery passa por todos os processos de integração e testes, porém para implantação em produção não é requerido uma aprovação manual. Isso significa que novas funcionalidades, correções de bugs e melhorias são implantadas automaticamente para os usuários assim que estiverem prontas e testadas. Por questões de governança e qualidade a change ainda é registrada, porém a aprovação acontece também de forma automática servindo apenas como registro.

Continuous Deployment

Os principais benefícios do continuous deployment incluem:

- Automação completa
- Deployment contínuos
- Feedback rápido
- Rollbacks podem ser automatizados
- Testes em produção

Benefícios CI/CD

Os principais benefícios do CI/CD incluem:

- Automação via Pipeline
- Entrega rápida e de valor
- Redução de riscos
- Melhoria da colaboração
- Implantação incremental
- Ambientes heterogêneos
- Rollbacks podem ser automatizados
- Testes em produção

Ferramentas CI/CD

Ferramentas populares de CI/CD:

- Github Action
- Gitlab CI
- Jenkins
- Circle CI
- Azure DevOps
- AWS CodePipeline

Testes de Software

Test-Driven Development (TDD) é uma abordagem de desenvolvimento em que os testes são escritos antes mesmo do código de produção. O ciclo típico de TDD envolve três etapas:

Red: O desenvolvedor escreve um teste automatizado que define um comportamento desejado ou uma funcionalidade que ainda não existe no sistema. Este teste inicialmente falha, pois o código de produção correspondente ainda não foi implementado.

Green: O desenvolvedor escreve o código mínimo necessário para fazer o teste passar. O objetivo aqui é implementar apenas o suficiente para satisfazer o teste recém criado.

Refactor: Após o teste passar, o desenvolvedor pode refatorar o código para melhorar sua qualidade, sem alterar seu comportamento observável.

Este ciclo é repetido iterativamente, com o desenvolvedor escrevendo novos testes para novas funcionalidades ou alterações no código existente.

Testes de Software

Behavior-Driven Development (BDD) é uma abordagem que se concentra no comportamento esperado do sistema em vez de nas implementações específicas. Ele promove uma colaboração mais estreita entre desenvolvedores, testers e stakeholders do negócio, utilizando uma linguagem comum para descrever o comportamento do sistema em termos de cenários de usuário.

Os cenários de usuário são escritos em uma linguagem natural compreensível por todas as partes interessadas, geralmente usando uma estrutura como Given-When-Then (Dado-Quando-Então). Por exemplo:

Dado que o usuário esteja na página de login

Quando ele inserir um nome de usuário e senha válidos

Então ele deverá ser redirecionado para a página inicial

Em seguida, esses cenários de usuário são traduzidos em testes automatizados, geralmente usando uma ferramenta de BDD, como Cucumber por exemplo.

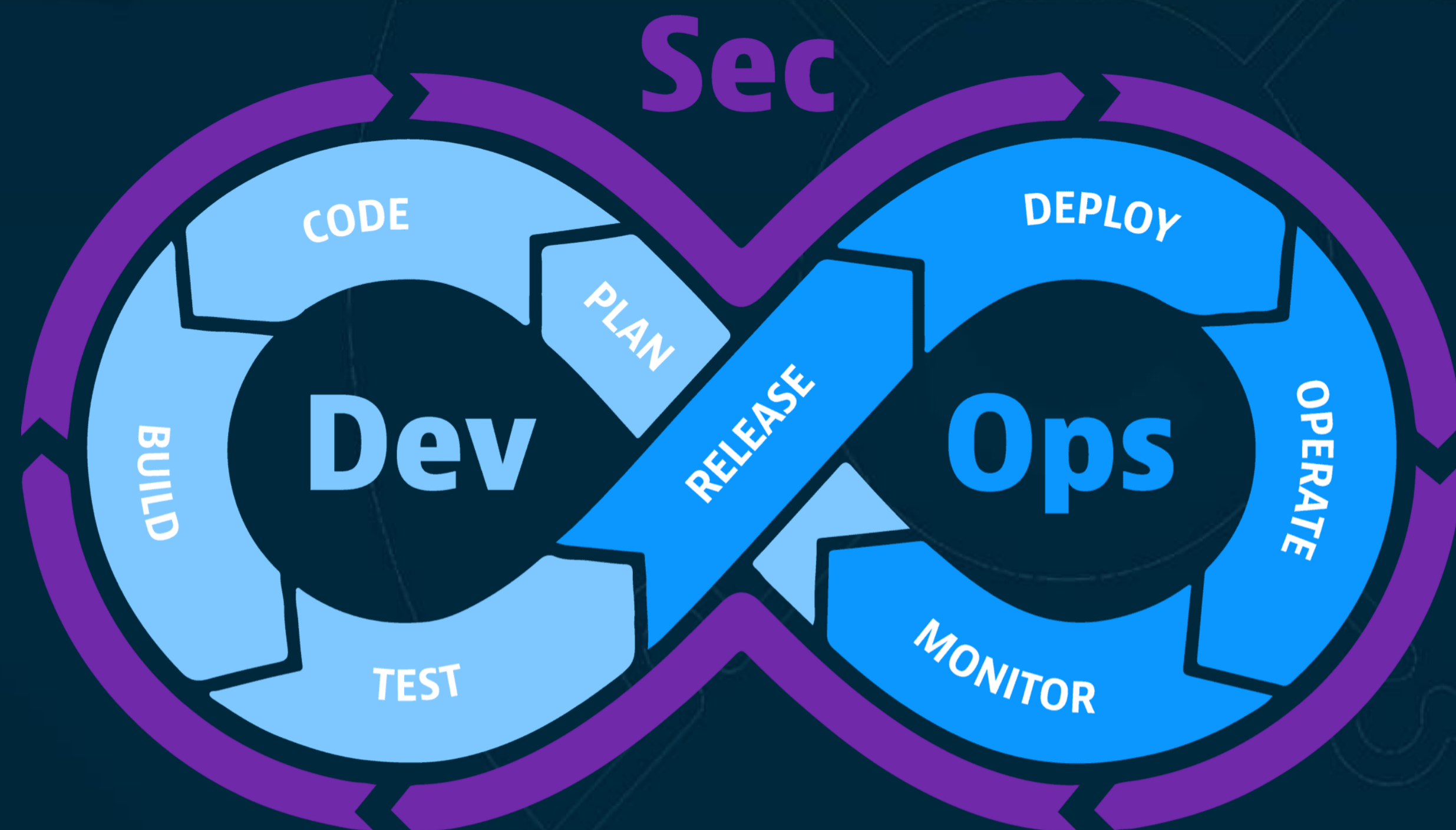
Abordagens de testes automatizados

Tipos de testes automatizados:

- Testes Unitários
- Testes Integrados
- Testes End to End
- Testes Regressivos
- Testes sintéticos
- Testes mutantes
- Testes Contínuos
- Matriz de testes

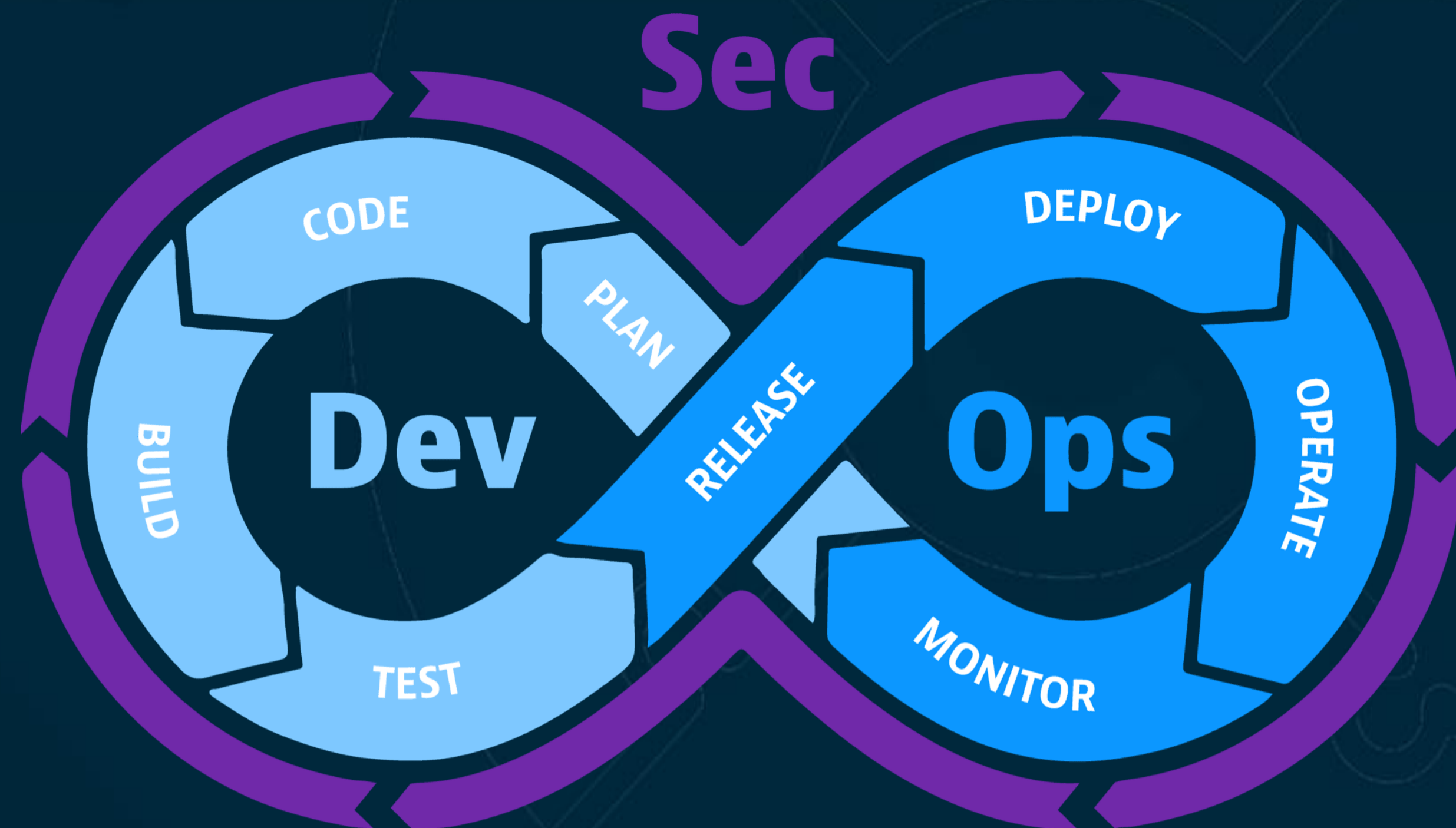
DevSecOps nada mais é que incluirmos segurança no ciclo do desenvolvimento de software e não mais como uma equipe ou departamento apartado onde tínhamos mais interação quando o software já estava em produção. Essa abordagem traz muito mais confiabilidade ao produto pois trazemos etapas de segurança para fase inicial do desenvolvimento e garantindo que nosso software esteja sem vulnerabilidades críticas a serem exploradas, até porque não adianta um código funcional que não tem segurança.

DevSecOps



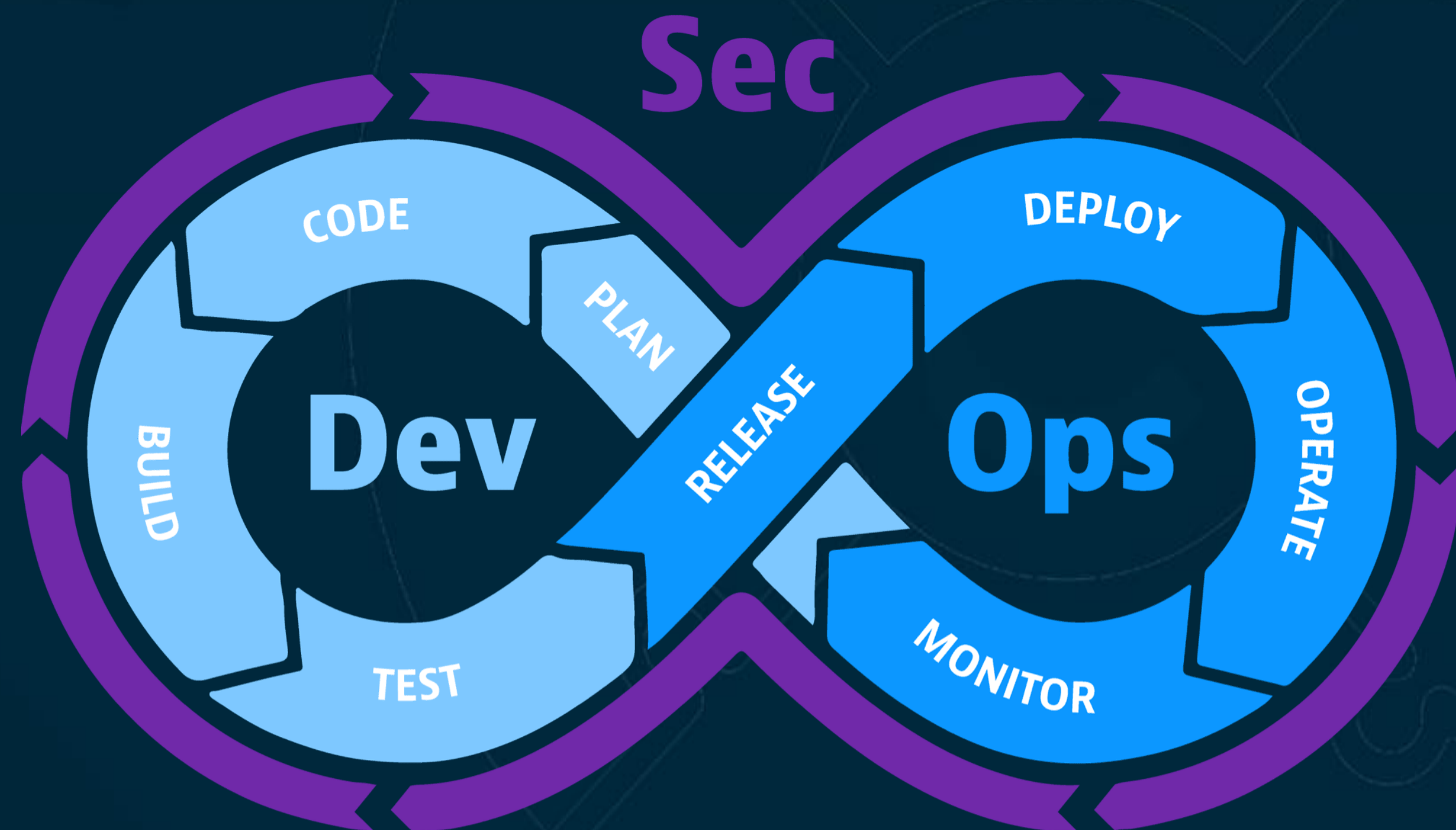
SAST (Static Application Security Testing):

é uma técnica de análise estática que examina o código-fonte ou o código compilado de um aplicativo em busca de vulnerabilidades de segurança. Ela procura por falhas de segurança, como vulnerabilidades de injeção de código (como SQL injection e XSS), uso inadequado de APIs e outras vulnerabilidades de código. O SAST é realizado durante a fase de desenvolvimento do software, geralmente como parte do processo de contínuos integration.

DevSecOps

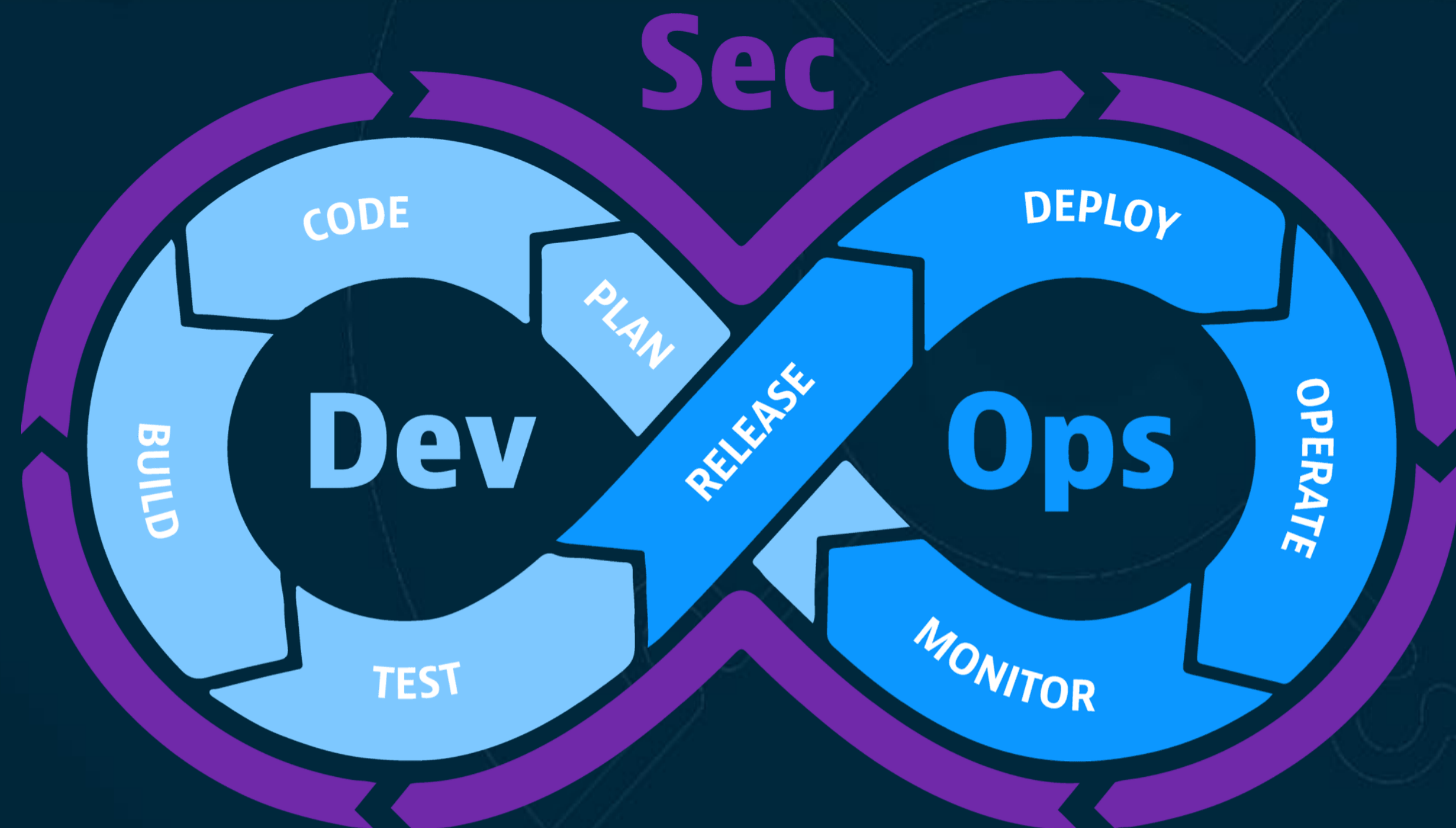
DAST (Dynamic Application Security Testing):

é uma técnica de análise dinâmica que avalia o software em execução para identificar vulnerabilidades de segurança enquanto ele está sendo executado. DAST envolve pen tests automatizados ou manuais, onde é feito a simulação de ataques de um invasor externo. Ele procura por vulnerabilidades que podem ser exploradas quando o software está em execução, como configurações de servidor inadequadas, falhas de autenticação e autorização, entre outros.

DevSecOps

IAST (Interactive Application Security Testing):

é uma técnica que combina elementos de SAST e DAST. Ao contrário do SAST, que analisa o código-fonte em repouso, e do DAST, que testa o aplicativo em tempo de execução, o IAST monitora a execução do aplicativo durante os testes de segurança e examina o código-fonte em busca de vulnerabilidades enquanto o aplicativo está sendo executado. Isso permite uma análise mais precisa e contextualizada das vulnerabilidades, pois leva em consideração o contexto da execução do aplicativo.

DevSecOps

Trunk-based Development vs. Git Flow

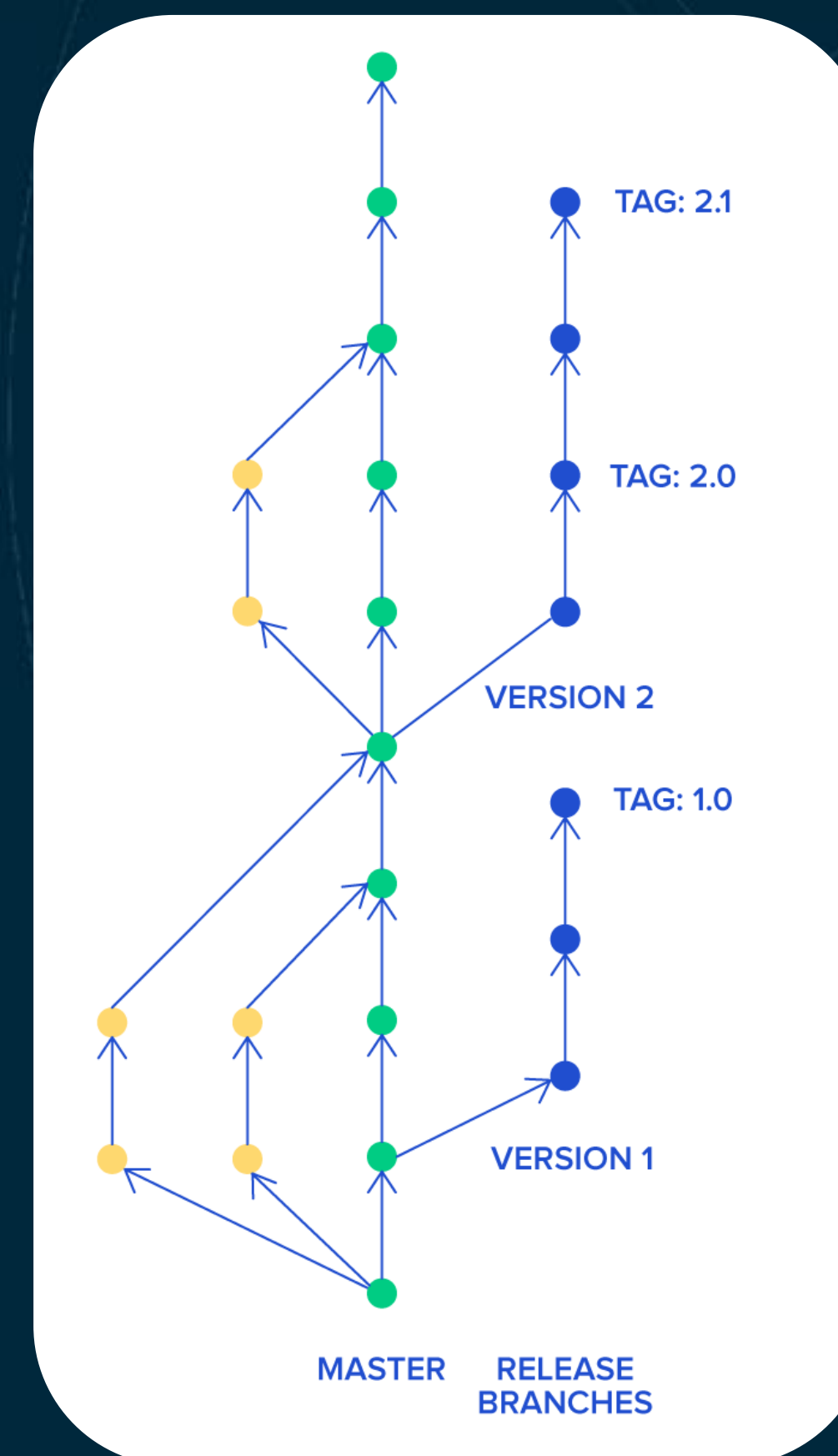
Trunk-based Development

Vantagens:

- Simplicidade
- Entrega Contínua
- Agilidade

Desvantagens:

- Conflitos frequentes
- Risco de regressões
- Dificuldade com equipes grandes



Trunk-based Development vs. Git Flow

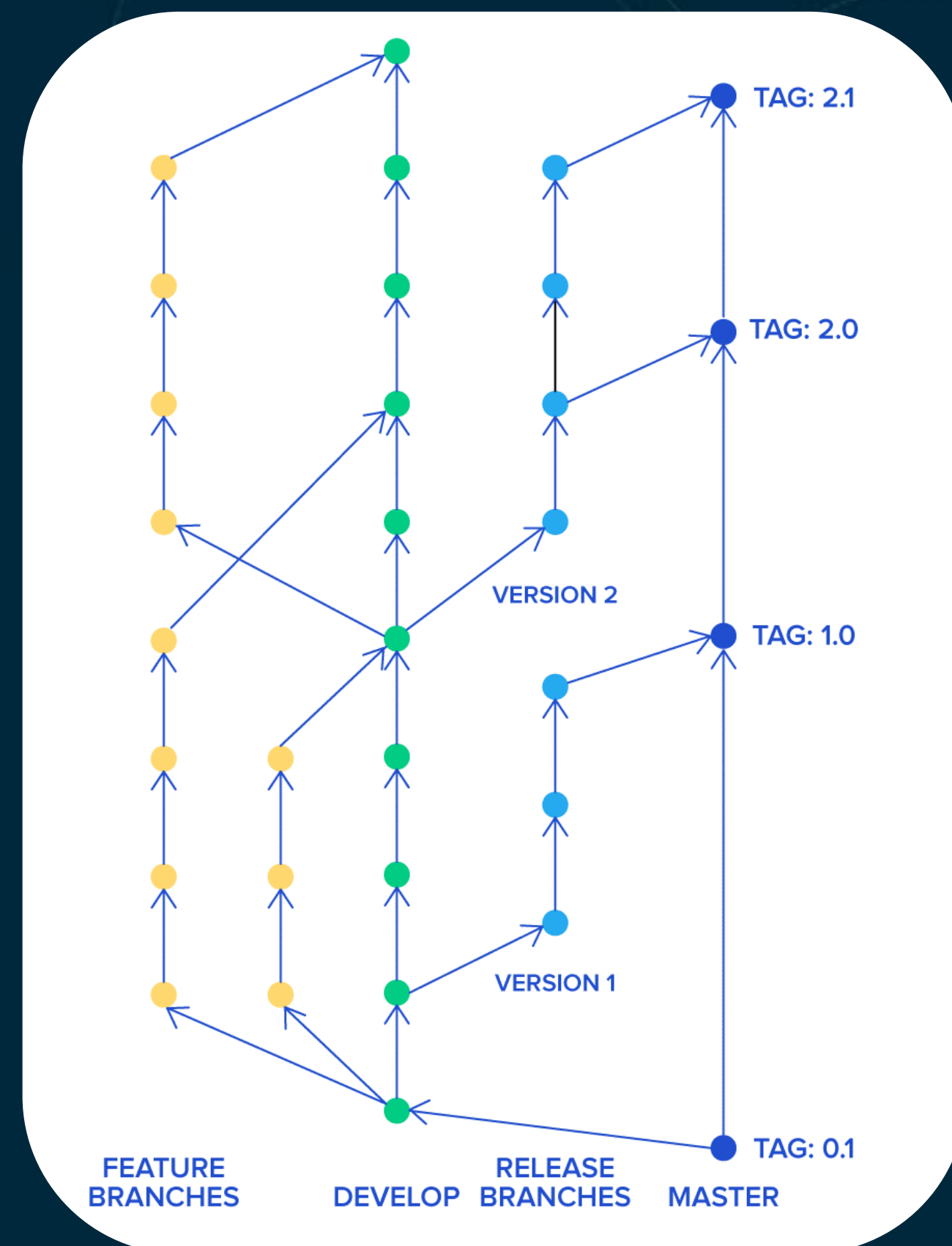
Git Flow

Vantagens:

- Organização
- Mitigação de conflitos
- Estabilidade

Desvantagens:

- Complexidade
- Overhead de gerenciamento



Best Practices

Change log

- <https://keepachangelog.com/pt-BR/1.1.0/>

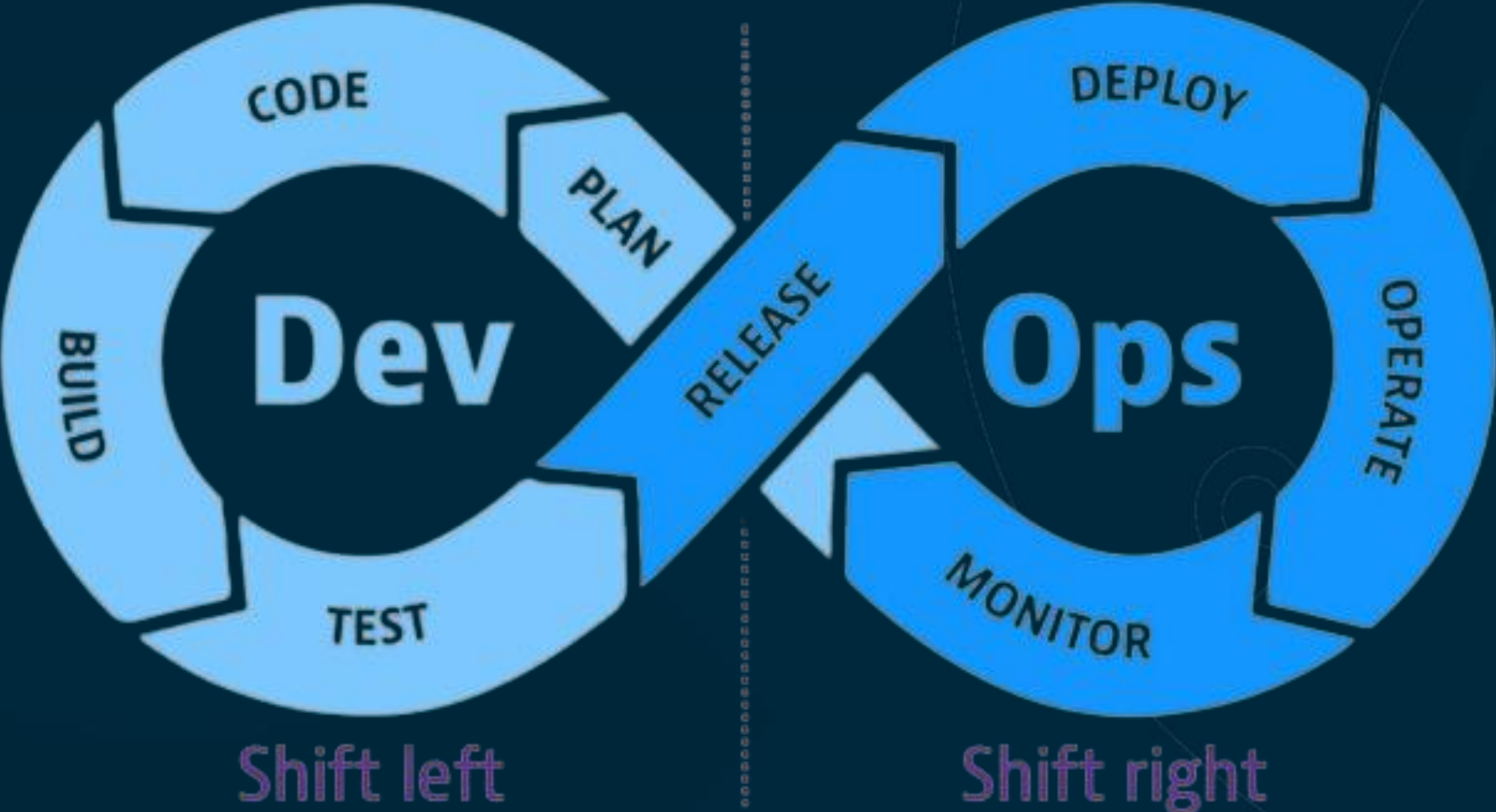
Conventional commits

- <https://www.conventionalcommits.org/en/v1.0.0/>

Semantic Versioning

- <https://semver.org/spec/v2.0.0.html>

Best Practices



Github Actions

Overview sobre os principais componentes e funcionalidades:

- Workflows => YAML file com mapeamento dos jobs a serem executados
- Events => Trigger do workflow
- Jobs => Mapeamento dos steps a serem executados
- Action => Custom app para reuso de atividades frequentes
- Runners => Servidor onde será executado o workflow
- Reusable => Workflow para reuso de atividades frequentes
- Inputs => Valor de entrada para um workflow, action ou reusable
- Outputs => Compartilha valores de saída entre jobs
- Vars => Variável configurada a nível de org, repository e environment
- Secrets => Variável com valor sensível configurada a nível de org, repository e environment

Pipeline de imagem de container

Demonstração:

- Build imagem docker
- Testes unitários
- SAST
- Documentação
- Versionamento
- Publicação

Automação de infraestrutura



Overview do módulo

- IaC (Infraestrutura como código)
- GitOps
- Gerenciamento de Configuração
- Pipeline de Infraestrutura

Infrastructure as Code (IaC)

Infraestrutura como código (IaC) é a capacidade de provisionar e suportar a infraestrutura usando código em vez de configurações e processos manuais. Para suportar as aplicações se faz necessário diversos componentes de infraestrutura, como sistemas operacionais, banco de dados, serviços de fila e armazenamento.

IaC veio como uma solução para aproximar práticas da engenharia de software na infraestrutura, permitindo gerenciar ambientes através arquivos de configuração integrados a um SCM e se beneficiar de práticas como o uso de módulos para reuso de código.

Infrastructure as Code (IaC)

Existem dois tipos de abordagem de IaC: declarativa e imperativa.

A abordagem declarativa define o estado desejado do sistema, incluindo os recursos necessários, as propriedades que eles precisam ter e uma ferramenta de IaC para configurá-lo. Essa abordagem também mantém uma lista do estado atual dos objetos do seu sistema, simplificando o gerenciamento da desativação da infraestrutura.

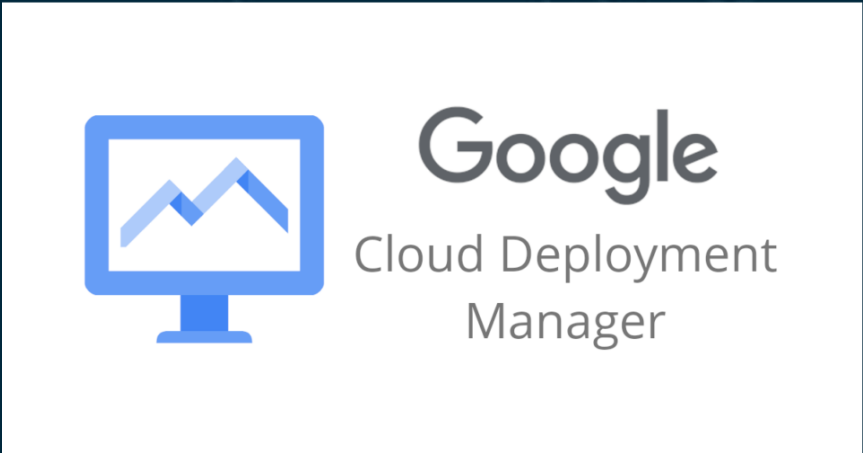
Por outro lado, a abordagem imperativa define os comandos específicos necessários para alcançar a configuração desejada. Depois, esses comandos precisam ser executados na ordem correta.

Muitas das ferramentas de IaC que usam uma abordagem declarativa provisionam automaticamente a infraestrutura desejada. Se você alterar o estado desejado, uma ferramenta de IaC declarativa aplicará as alterações para você. Uma ferramenta imperativa exige que você saiba como as alterações deverão ser aplicadas.

IaC Tools



AWS CloudFormation



Policy as Code

Policy as Code é uma abordagem de gerenciamento de políticas na qual as mesmas são definidas, atualizadas, compartilhadas e aplicadas por meio de código. Nesse contexto policy pode ser considerado qualquer tipo de regra, condição ou instrução de governança ou processos de TI. Usando essa abordagem é possível adicionarmos essas ferramentas tanto em tempo de desenvolvimento quanto também pós implementação garantindo o compliance dos ambientes. Usando policy as code é possível que desenvolvedores e engenheiros de segurança “falem” a mesma língua aproximando também esses dois papéis.

Policy as Code


checkov



Open Policy Agent



Cloud Custodian



HashiCorp

Sentinel



turbot

Infraestrutura como Código (IaC)

Benefícios:

- Redução de custos
- Aumento na velocidade das implantações
- Facilidade em replicar o ambiente
- Documentação viva
- Redução de erros
- Melhoria na consistência da infraestrutura
- Automação
- Controle de versão

GitOps

GitOps é uma forma de implementar continuous deployment para aplicações cloud native. Ele se concentra em uma experiência centrada no desenvolvedor para operar a infraestrutura, usando ferramentas com as quais os devs já estão familiarizados.

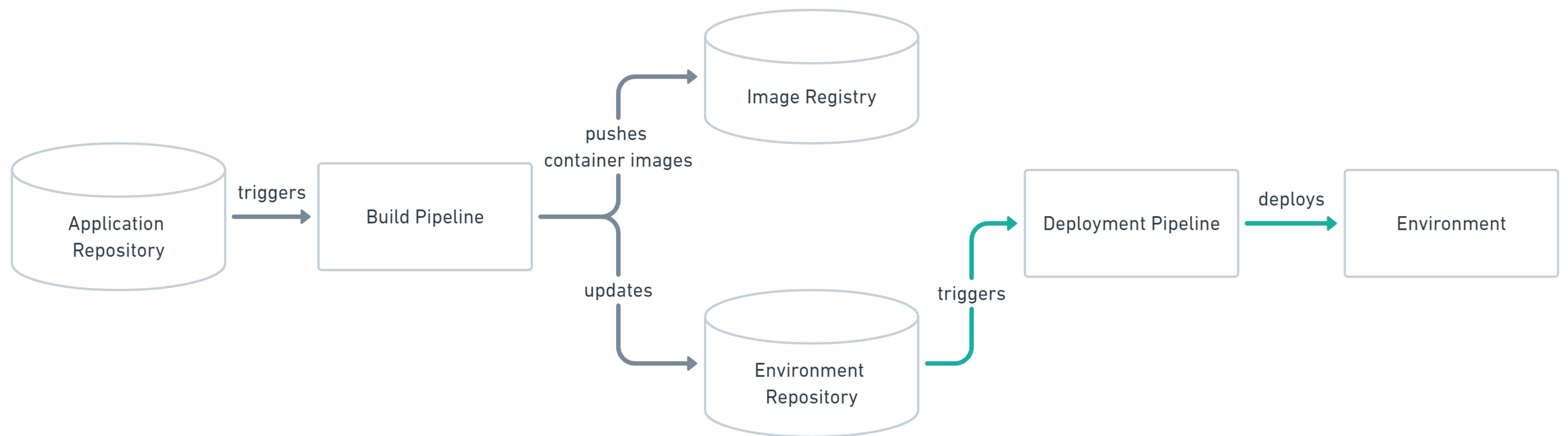
A ideia central do GitOps é ter um repositório Git que sempre contenha arquivos declarativos da infraestrutura desejada no ambiente de produção e um processo automatizado para fazer com que o ambiente de produção corresponda ao estado descrito no repositório. Para implantar uma nova aplicação ou atualizar uma existente, basta atualizar o repositório – o processo automatizado cuida de todo o restante.

GitOps

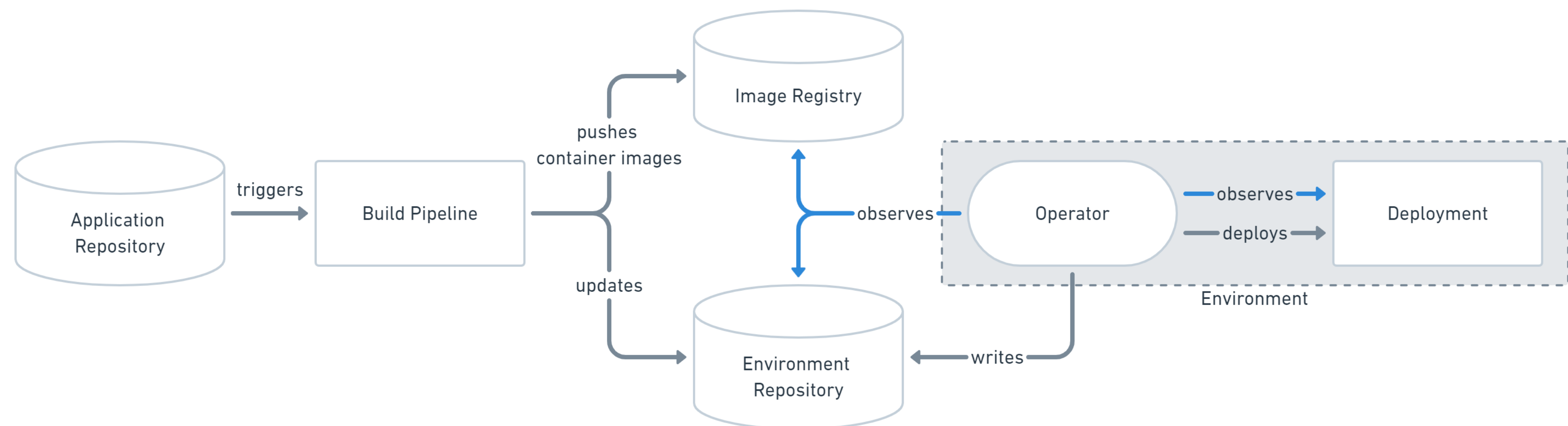
Existem pelo menos dois repositórios: o repositório de app e o repositório de configuração do ambiente. O repositório do app contém o código-fonte e os manifestos de implantação para implantação enquanto repositório de configuração do ambiente contém todos os manifestos de implementação da infraestrutura desejada. Ele descreve quais apps e serviços de infraestrutura (broker de mensagens, clusters K8S, service mesh, ferramenta de monitoramento, ...) devem ser executados, com qual configuração e versão no ambiente de implantação.

Existem duas maneiras de implementar a estratégia de implantação para GitOps: implantações baseadas em Push e implantações baseadas em Pull. A diferença entre os dois tipos de implantação é como é garantido que o ambiente de implantação realmente esteja no estado da infraestrutura desejada.

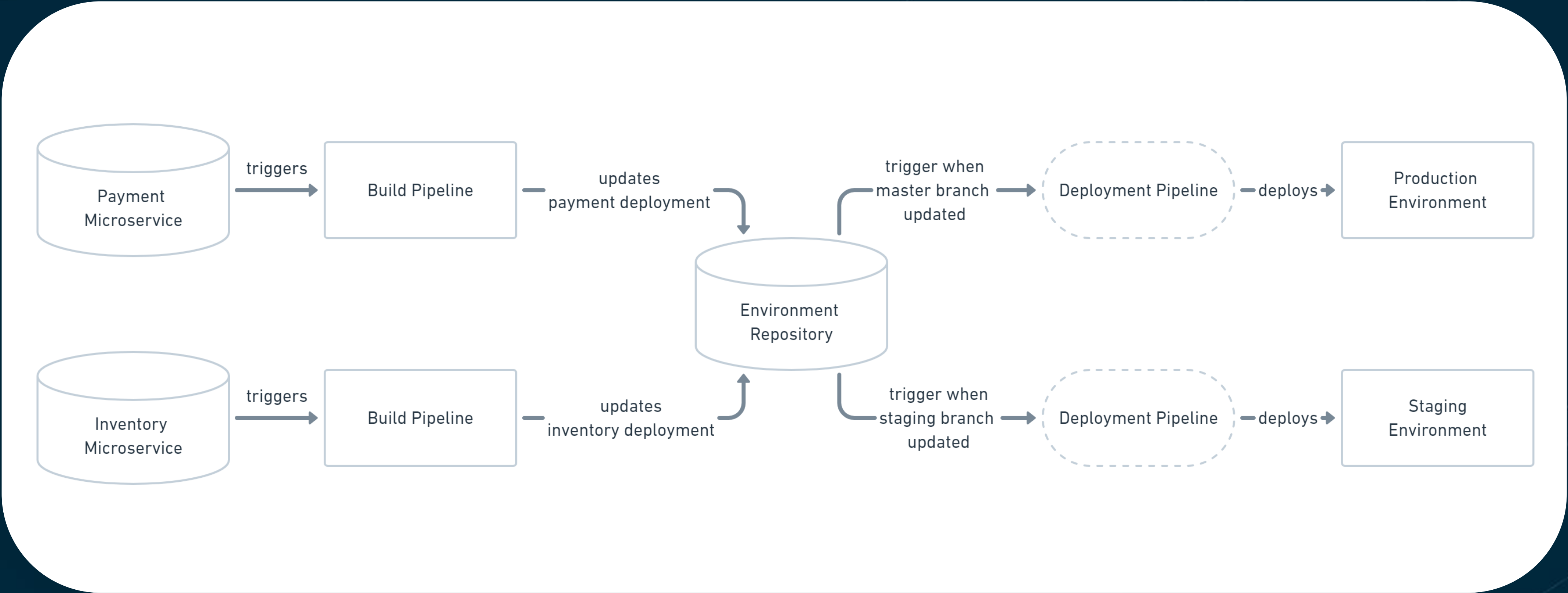
GitOps Push-based



GitOps Pull-Based



GitOps Multiple Apps e ambientes

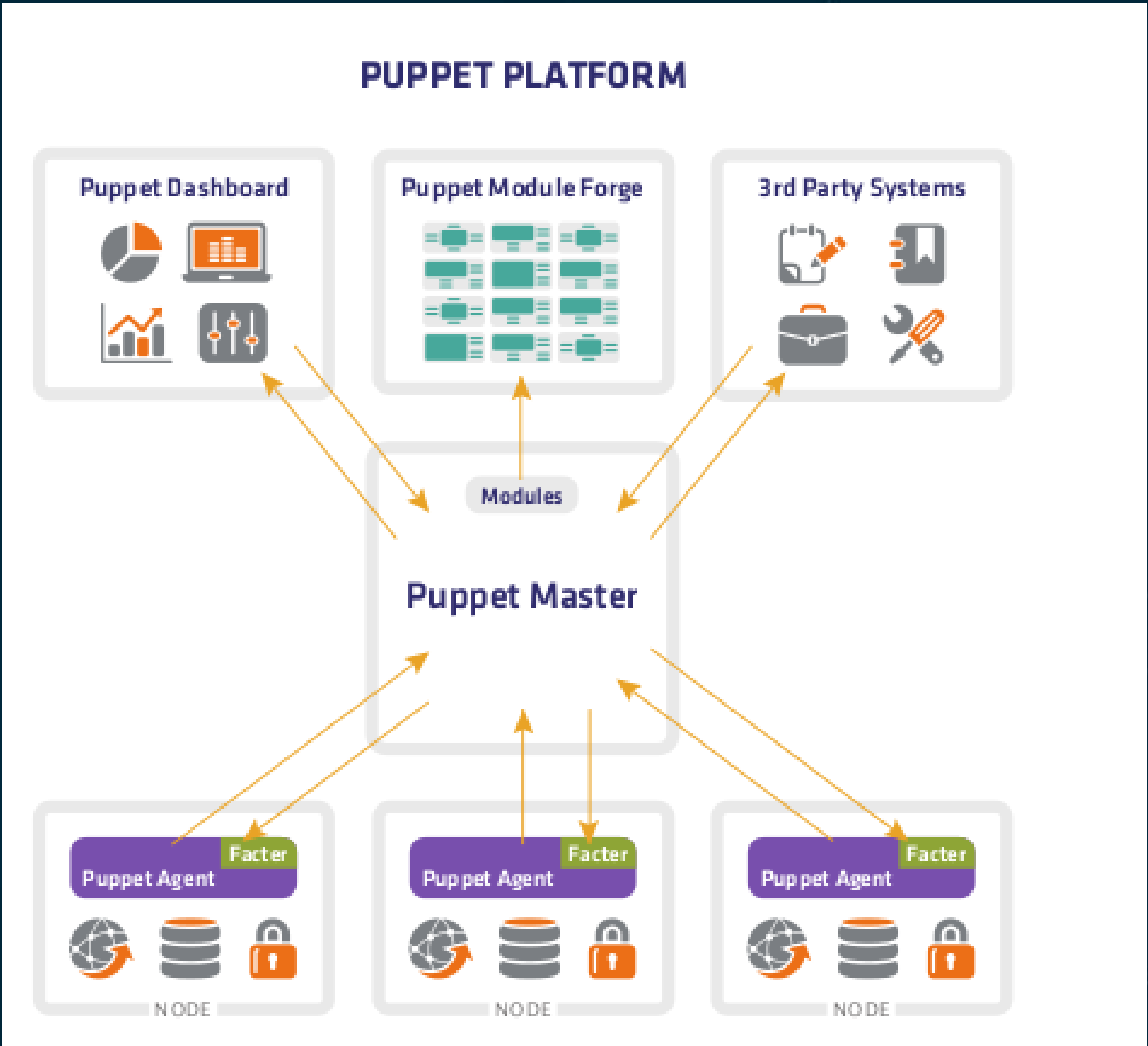


Gerenciamento de configuração

O gerenciamento das configurações é o processo para garantir que as configurações dos servidores, aplicações e outros ambientes de um sistema permaneçam conhecidas, consistentes e confiáveis ao longo do tempo. Qualquer sistema de TI tem determinadas configurações relacionadas a versões de software, segurança, rede e outras configurações que são essenciais para o funcionamento ideal.

O gerenciamento das configurações rastreia, atualiza e mantém essas configurações para que o sistema funcione em uma linha de base predeterminada e permaneça seguro independentemente de quaisquer alterações. Esse recurso estabelece e mantém a consistência do desempenho de um sistema e de seus atributos físicos e funcionais. Além disso, o gerenciamento das configurações leva em consideração as informações operacionais, o design e os requisitos do sistema ao longo de sua vida útil.

Gerenciamento de configuração



Gerenciamento de configuração

Benefícios:

- Melhorar a capacidade de recuperação do sistema
- Reduzir a interrupção do sistema
- Possibilitar a auditoria do ambiente
- Tracking de alterações
- Documentação viva
- Simplificar as atualizações das configurações

Pipeline de Infraestrutura

Demonstração:

- Linters
- Testes
- Dry-Run
- Deployment
- Documentação
- Versionamento

Site Reliability Engineering (SRE) Princípios e Práticas



SRE Princípios e Práticas

Princípios

- Operação como software
- Objetivos de nível de serviço
- Redução de toil
- Automação
- Reduzir custo de falha
- Responsabilidade compartilhada

Práticas

- Observabilidade

- On-Call / Resposta a incidentes
- Postmortem e Root-Cause Analysis
- Testes
- Capacity Planning
- Desenvolvimento
- Produto

Objetivos de Nível de Serviço

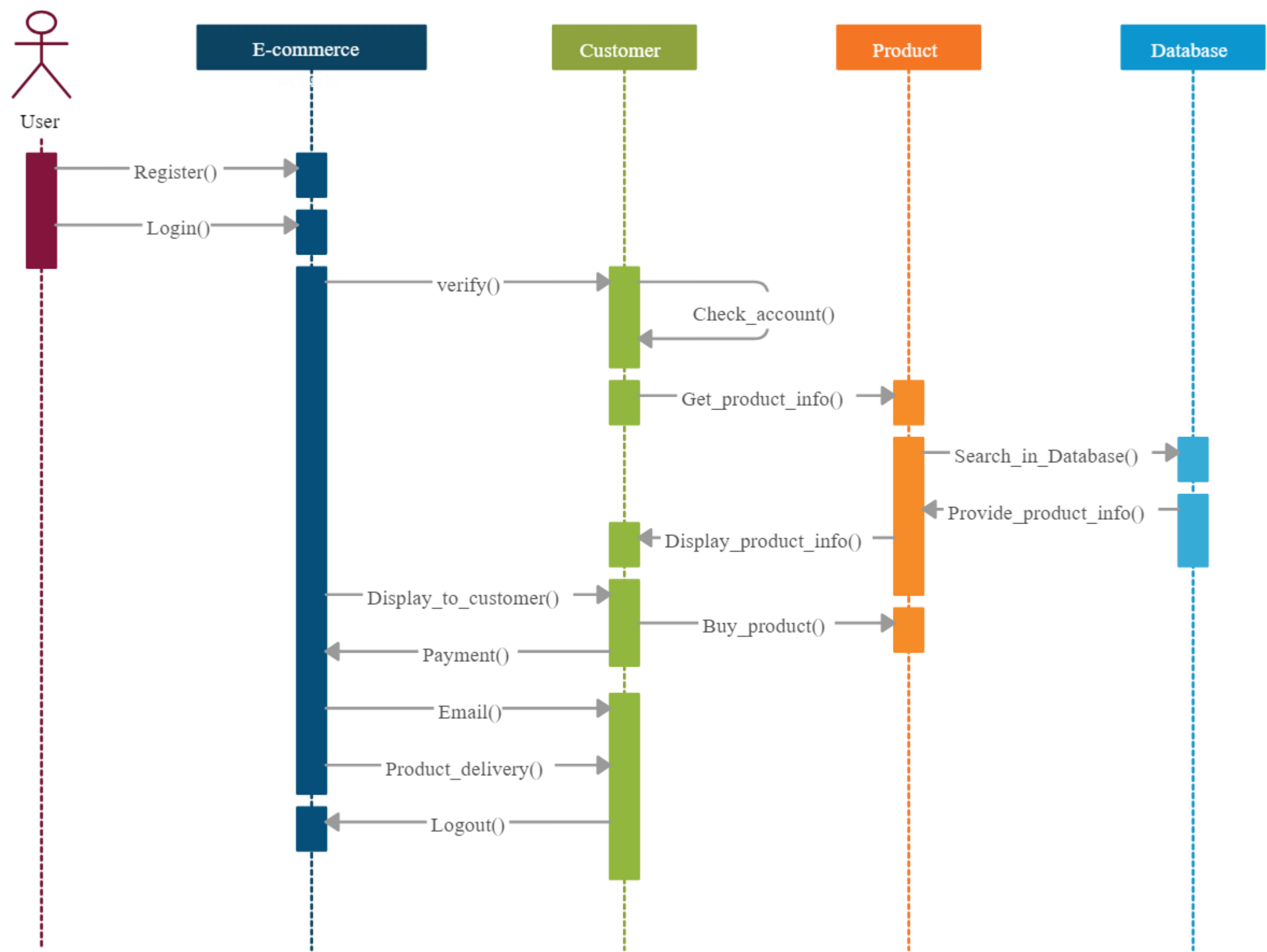
Compreender e medir os comportamentos essenciais de um serviço é fundamental para gerenciá-lo eficientemente. Isso envolve definir e cumprir um nível de serviço para os usuários, através de SLIs, SLOs e SLAs. Essas métricas indicam o que é importante, os objetivos desejados e como reagir caso o serviço não atenda às expectativas. Escolher as métricas certas permite ações corretivas adequadas e dá confiança à equipe de SRE na integridade do serviço.

As principais responsabilidades dos SREs não são apenas automatizar “todas as coisas” e manter o ambiente de operações. As suas tarefas e projetos diários são conduzidos pelos SLOs: garantindo que os SLOs são defendidos a curto prazo e que podem ser mantidos a médio e longo prazo. Podemos até afirmar que sem SLOs não há necessidade de SREs

Objetivos de Nível de Serviço

- **SLI (Service Level Indicator):** Métrica utilizada para qualificar o desempenho ou qualidade de um serviço, como latência, disponibilidade, durabilidade ou taxa de transferência.
- **SLO (Service Level Objective):** É uma meta de quão bem um produto ou serviço deve operar, por exemplo 99,5% de disponibilidade na API de consulta ao saldo. SLOs nunca devem ser 100%
- **SLA (Service Level Agreement):** Acordo formal entre um provedor de serviço e seus clientes, por exemplo 99,7% de disponibilidade na API de consulta ao saldo.
- **Error Budget:** É a quantidade tolerada de desvios em um determinado período de tempo, sem comprometer os objetivos do serviço.
- **Error Budget Policies:** São diretrizes ou políticas estabelecidas para determinar as consequências de exceder o Error Budget.

Diagrama de Sequência



Total de requests: 3.663.253
Total de requests com sucesso: 3.557.865 (97,123%)
Latência do 90º percentil: 432 ms
Latência do 99º percentil: 891 ms

SLOs estabelecidos

Tipo de SLO	Objetivo
Disponibilidade	97%
Latência	90% < 450ms
Latência	99% < 900ms

Error Budget 30 dias

SLO	Falhas permitidas
97% de disponibilidade	109.897
90% < 450 ms	366.325
99% < 900 ms	36.632

Objetivos de Nível de Serviço (SLOs)

Para auxiliar na geração de métricas dos SLO e error budgets podemos utilizar algumas ferramentas, segue abaixo a recomendação de algumas delas:

OSS:

- SLO-Generator (<https://github.com/google/slo-generator>)
- Sloth (<https://github.com/slok/sloth>)

Enterprise:

- Dynatrace
- Datadog

Vídeo com demo de como é usado em um ambiente produtivo:

<https://www.youtube.com/watch?v=7IUgTNcPFIU&t=2463s>

Redução Toil

Toil são atividades manuais, repetitivas e que consomem tempo, como resolver problemas de rotina, executar tarefas de manutenção simples e responder a solicitações comuns. Essas tarefas consomem recursos preciosos das equipes de Engenharia de Confiabilidade de Sites (SRE) e podem afetar negativamente sua eficiência e motivação. O objetivo é identificar e eliminar o toil por meio de automação e simplificação de processos, liberando tempo e recursos para atividades mais estratégicas e de maior valor.

Redução Toil

Características do Toil:

- Manual
- Repetitivo
- Automatizavel
- Não tático / Reativo
- Falta de valor perene
- Cresce tão rápido quando sua origem

Redução Toil

Atividades ofensoras que podem gerar Toil:

- Processos
- Interrupções em produção
- Release Shepherding
- Migrações
- Finops e Capacity Planning
- Resolução de Problemas em Arquiteturas frágeis

Redução Toil

Atividades que podem reduzir o Toil:

- Identificar e mensurar Toil
- Engenharia para Eliminar o Toil do Sistema
- Rejeitar o Toil
- Usar SLOs para Reduzir o Toil
- Automações parciais
- Fornecer Métodos de Autoatendimento
- Promover a Redução do Toil como uma Característica
- Aumentar a Uniformidade
- Avaliar Riscos na Automação
- Automatizar a Resposta ao Toil
- Usar Ferramentas de Código Aberto e de Terceiros
- Usar Feedback para Melhorar

On-Call

On-Call é um sistema em que os engenheiros são designados para estar disponíveis para responder a incidentes ou problemas relacionados aos produtos e serviços que suportam durante o horário comercial e também fora do horário normal de trabalho. Esses engenheiros são contatados automaticamente ou por outros membros da equipe quando ocorre um problema que requer atenção imediata. Podem ser acionados a qualquer momento incluindo fins de semana e feriados. Devem estar prontos para responder e resolver problemas rapidamente para minimizar o impacto para os usuários finais afim de manter o não rompimento dos SLOs.

On-Call

Para que o On-Call não se torne uma atividade pesada a ponto de afetar a produtividade e o psicológico dos engenheiros é importante que seja estabelecido uma escala onde haverá uma rotação entre os membros do time. Portanto, as equipes de SRE buscam equilíbrio, garantindo que pelo menos 50% do tempo seja dedicado a projetos, permitindo abordar estrategicamente problemas encontrados em produção.

Durante o On-Call os membros terão como prioridade realizar o primeiro combate aos incidents e root cause analysis (RCA), porém na ausencia dessas atividades devem focar em atividades como redução de toil, automações, finops, capacity planning, house keeping, oportunidades de melhorias nos pilares da observabilidade, análise de reinicidência de incidentes...

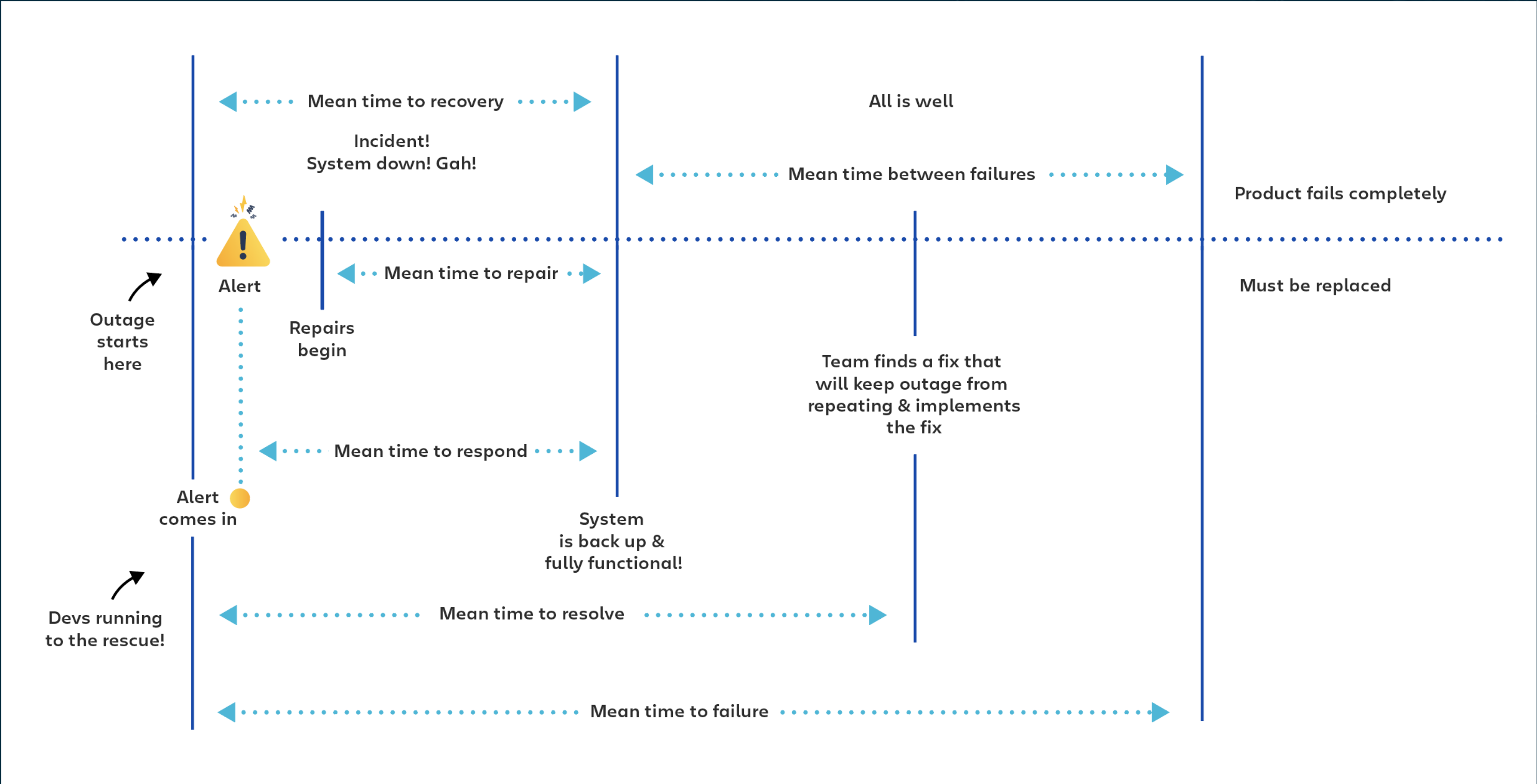
Métricas de incidentes

As métricas de gerenciamento de incidentes são essenciais para avaliar e aprimorar continuamente a eficácia do processo. Elas oferecem insights valiosos sobre o impacto, tempo de resolução e frequência de diferentes tipos de incidentes, permitindo otimizar a resposta e garantir a disponibilidade contínua dos serviços críticos.

Principais métricas:

- Time to resolution (TTR)
- Mean time to repair (MTTR)
- Mean time to resolve (MTTR)
- Mean time to acknowledge (MTTA)
- Número de Incidentes por Período de Tempo
- Mean time between failures (MTBF)

Métricas de incidentes



Postmortem

O termo post-mortem refere-se a uma prática organizacional de revisão e aprendizado após a ocorrência de incidentes ou falhas em sistemas. Em vez de culpar indivíduos, a abordagem proativa busca entender as causas raiz dos problemas, documentar detalhadamente o incidente e implementar ações corretivas para evitar recorrências.

Essa prática promove uma cultura de transparência, confiança e responsabilidade compartilhada, onde as equipes se sentem seguras para relatar incidentes e compartilhar aprendizados, contribuindo assim para a melhoria contínua do sistema e a prevenção de futuros problemas.

Postmortem

Informações que devem conter em um bom postmortem:

- Contexto
- Análise de causa raiz (RCA)
- Ações corretivas
- Aprendizados
- Revisão e acompanhamento

Exemplo de um bom postmortem:

<https://sre.google/sre-book/example-postmortem/>

“ *The cost of failure is education.* ”

Devin Carraway

Correction of Errors (CoE)

Correction of Erros é um processo criado pela AWS para melhorar a qualidade da operação e dos produtos documentando e resolvendo problemas. O CoE visa definir uma maneira padronizada de documentar as causas básicas críticas e garantir que elas sejam revisadas e abordadas, ajudando os times a entender as RCAs, serem revisadas de forma consistente e endereçar os pontos de ação corretamente.

Correction of Errors (CoE)

Estrutura de um COE:

- O que aconteceu?
- Qual foi o impacto nos clientes e no seu negócio?
- Qual foi a causa raiz?
- Que dados você tem para apoiar isso?
 - especialmente métricas e gráficos
- Quais foram as implicações críticas dos pilares do WAF, especialmente a segurança?
 - Ao criar arquiteturas, você faz concessões entre pilares com base no contexto do seu negócio. Essas decisões de negócios podem orientar suas prioridades de engenharia. Você pode otimizar para reduzir custos em detrimento da confiabilidade em ambientes de desenvolvimento, ou para soluções de missão crítica, pode otimizar a confiabilidade com custos maiores. Segurança é sempre prioridade zero, pois você precisa proteger seus clientes.
- Que lições você aprendeu?
- Que ações corretivas você está tomando?
 - Itens de ações
 - Itens relacionados (tíquetes de problemas, etc.)

Revisão

- Devemos fazer revisões dos CoEs com os times a fim de compartilhar conhecimento
- CoEs de alto impacto devem ser revisadas durante a ops meeting

OpsMeeting

Uma "OpsMeeting" é uma reunião operacional que segue as práticas recomendadas SRE. Essas reuniões são projetadas para facilitar a comunicação e a colaboração entre equipes de operações, desenvolvimento e outras partes interessadas envolvidas na entrega e operação de sistemas de software.

Segue alguns aspectos chave de como uma OpsMeeting:

- Agenda bem definida
- Participantes relevantes
- Frequência regular
- Revisão de incidentes e problemas
- Revisão da observabilidade
- Planejamento e prioridades
- Feedback e melhoria contínua

ir que todos os
ões, a AWS
eres seniores,
es encontros,
ções, além de
o ao crescente
o resposta, foi
l cada equipe
te de serem



Operational Readiness Review (ORR)

A Amazon Web Services (AWS) criou a Operational Readiness Review (ORR) para resumir os aprendizados dos incidentes operacionais da AWS em perguntas selecionadas com orientação de práticas recomendadas.

"Good intentions never work, you need good mechanisms to make anything happen" — Jeff Bezos

O objetivo é ajudar na construção de sistemas altamente disponíveis e resilientes sem comprometer a agilidade e velocidade de desenvolvimento. A ORR utiliza uma abordagem baseada em dados para identificar e mitigar riscos operacionais, complementando as revisões do Framework Well-Architected da AWS.

Operational Readiness Review (ORR)

1. Disponibilidade e Escalabilidade:

- Como você garante a disponibilidade dos seus serviços?
- Quais são os mecanismos de escala automática implementados?
- Quais medidas foram adotadas para lidar com picos de tráfego inesperados?

2. Resiliência e Recuperação:

- Quais são os seus procedimentos de recuperação de desastres?
- Como você gerencia backups e replicação de dados?
- Quais são os seus planos de failover em caso de falhas em uma região específica?

3. Monitoramento e Alarmes:

- Quais métricas são monitoradas para detectar problemas de desempenho?
- Como você define alertas para eventos críticos?
- Quais ações são tomadas em resposta a um alerta?

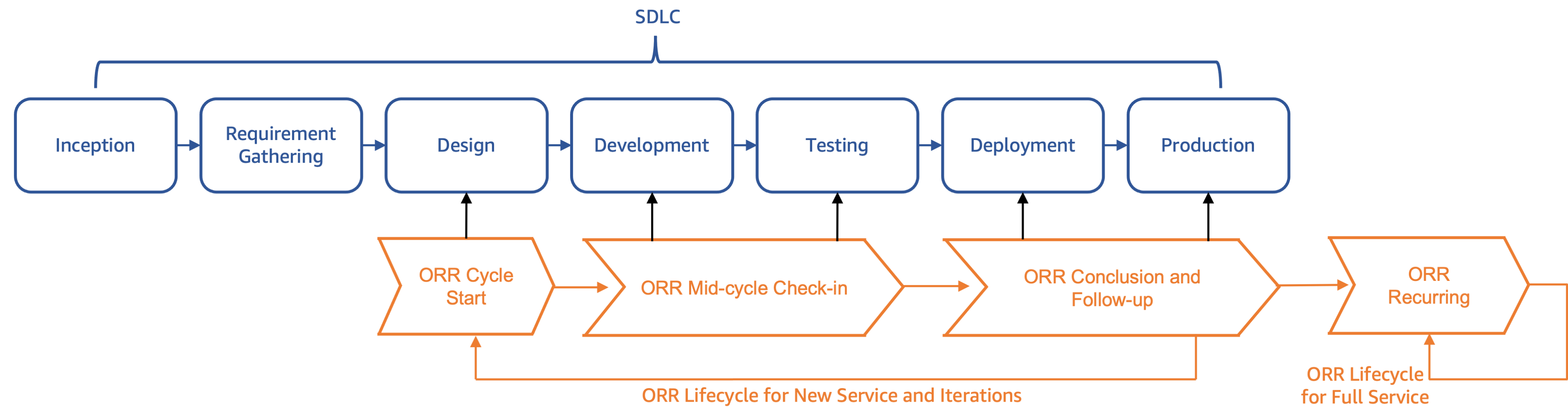
4. Segurança:

- Quais são as suas práticas de gerenciamento de identidade e acesso (IAM)?
- Como você protege seus dados em trânsito e em repouso?
- Quais são os controles de segurança implementados para proteger contra ataques externos e internos?

5. Gerenciamento de Mudanças:

- Como você gerencia e implementa alterações em sua infraestrutura e aplicativos?
- Quais são os processos para revisão e aprovação de alterações?
- Como você lida com reversões e backouts de alterações problemáticas?

Operational Readiness Review (ORR)



Chaos Engeneering

Chaos Engineering é uma prática que visa testar a resiliência de sistemas distribuídos sob condições adversas e inesperadas. O objetivo é expor falhas em sistemas de forma controlada, permitindo que as equipes de engenharia identifiquem e corrijam vulnerabilidades antes que elas afetem os usuários finais. Essa abordagem ajuda as equipes a construir sistemas mais robustos e confiáveis.

A origem do Chaos Engineering remonta à Netflix, que introduziu a prática no início dos anos 2010. A equipe de engenharia da Netflix desenvolveu uma ferramenta chamada "Chaos Monkey" para introduzir aleatoriamente falhas em seus sistemas de produção, para garantir que seus sistemas fossem resilientes o suficiente para lidar com falhas inesperadas.

Princípios Chaos Engineering

- **Planejar:** Decida o que você quer testar e como vai fazer isso. O objetivo aqui é criar uma hipótese. O que poderia dar errado em um sistema? Quais são as possíveis vulnerabilidades que podem ser exploradas?
- **Faça experimentos:** Injete falhas no sistema e veja como ele reage. A injeção de falhas é simplesmente o processo de introduzir um problema em um sistema existente para expor uma vulnerabilidade.
- **Análise:** Use os dados de seus experimentos para identificar possíveis pontos de falha.
- **Mitigar:** Se encontrar um problema, você pode encerrar o experimento para se concentrar em atenuá-lo. Caso contrário, você pode dimensionar o experimento até chegar ao ponto crucial do problema.

Game Day

Os Game Days visam replicar falhas e desvios de comportamento para testar sistemas, processos e respostas da equipe diante de incidentes e problemas. O objetivo é simular de fato falhas, permitindo que a equipe aja como se estivesse diante de uma situação real no ambiente de produção. Essas simulações devem ser realizadas regularmente para que a equipe desenvolva uma resposta rápida e eficaz, criando uma espécie de "memória muscular" para lidar com incidentes. Os game days devem abranger todas as áreas envolvidas na operação dos workloads, incluindo operações, teste, desenvolvimento, segurança, finanças e áreas de negócios.

Os game days podem ser conduzidos com réplicas do ambiente de produção utilizando Infraestrutura como Código (IaC), ou mesmo em um ambiente de homologação que seja uma réplica fiel do ambiente produtivo. Isso proporciona um ambiente seguro para testes que se assemelha muito ao ambiente real de produção.

Game Day

1. Defina o cenário que deseja praticar:

- Escolha o evento ou falha que deseja simular, baseando-se em falhas anteriores, fraquezas conhecidas nos processos ou na equipe, picos sazonais na demanda, entre outros. Isso ajudará a direcionar o foco do exercício.

2. Execute a simulação:

- Realize a simulação em um local separado, fora do ambiente de produção.
- Anuncie o início do game day.
- Execute os eventos simulados ao longo do dia.
- Execute cada evento simulado utilizando seu roteiro, confirmando as execuções para garantir o resultado desejado.
- Identifique através dos observadores quando o problema for corretamente abordado.
- Use o feedback dos observadores para determinar se deve atrasar algum evento simulado.

3. Analise o game day:

- Documente áreas onde suas ferramentas, processos, procedimentos e pessoas não atendem às suas necessidades e expectativas usando um processo de RCA (Root Cause Analysis).
- Realize uma análise após o término do dia de simulação para determinar se é necessário fornecer educação, treinamento ou ferramentas adicionais.
- Documente oportunidades para testar áreas adicionais em Game Days subsequentes.
- Examine a execução do Game Day em si e veja se pode ser melhorada.

Observabilidade



Overview do módulo

- Overview Observabilidade
- Logs
- Metricas
- Traces

Observabilidade

Observabilidade refere-se à capacidade de entender o comportamento interno de um sistema, principalmente através da análise de dados gerados por seus componentes. Isso inclui logs, métricas, traces e outros tipos de informações que podem ser coletadas e analisadas para diagnosticar problemas, entender o desempenho e tomar decisões sobre como melhorar ou otimizar o sistema. A observabilidade é fundamental para garantir que um sistema seja robusto, resiliente e capaz de atender aos seus objetivos de disponibilidade e confiabilidade.

Sintoma vs Causa

Symptom	Cause
I'm serving HTTP 500s or 404s	Database servers are refusing connections
My responses are slow	CPUs are overloaded by a bogosort, or an Ethernet cable is crimped under a rack, visible as partial packet loss
Users in Antarctica aren't receiving animated cat GIFs	Your Content Distribution Network hates scientists and felines, and thus blacklisted some client IPs
Private content is world-readable	A new software push caused ACLs to be forgotten and allowed all requests

Black-box vs White Box

Black-box:

- Focado em sintomas e problemas ativos.
- Detecta quando o sistema não está funcionando corretamente no momento.
- Útil para alertas, pois aciona a intervenção humana apenas quando há um problema real em andamento.

White-box:

- Inspecciona detalhes internos do sistema, como logs e endpoints HTTP.
- Identifica problemas iminentes e falhas mascaradas por tentativas de repetição.
- Essencial para depuração, pois permite entender a percepção de desempenho de diferentes componentes (ex.: servidor web e banco de dados).

Diferenças:

- Black-box é orientada por sintomas, enquanto White-box pode ser orientada tanto por sintomas quanto por causas, dependendo da informação disponível.
- Black-box é limitada para detectar problemas iminentes, enquanto White-box é mais abrangente e detalhada para monitoramento e diagnóstico.

O que é Observabilidade



Logs

Logs são registros de eventos que ocorrem em um sistema. Eles contêm informações detalhadas sobre atividades, erros, exceções e outros eventos relevantes. Os logs são essenciais para entender o comportamento de um sistema ao longo do tempo, identificar problemas, investigar falhas e realizar auditorias. Eles fornecem uma visão detalhada das operações do sistema e são frequentemente utilizados para diagnóstico e troubleshooting.

Logs

Boas práticas:

- Estrutura e formato
- Inclusão de Metadados Relevantes
- Contexto da Solicitação
- Desempenho
- Segurança e privacidade
- Integração com ferramentas de observabilidade

Métricas

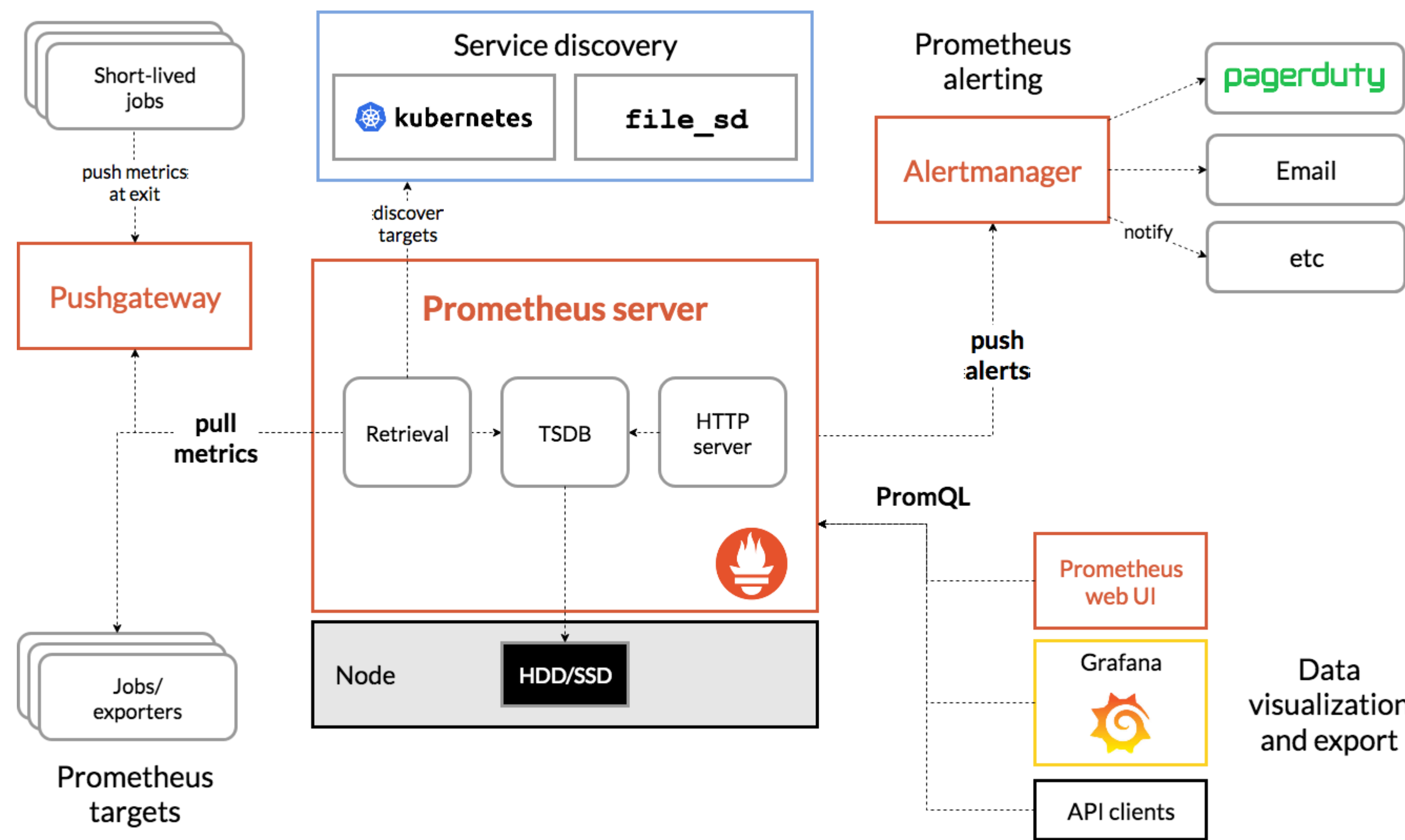
Métricas são medidas quantitativas que descrevem o comportamento de um sistema em um determinado momento. Elas podem incluir informações como uso de CPU, utilização de memória, taxa de transferência de rede, número de solicitações por segundo e muito mais. Métricas são fundamentais para monitorar o desempenho do sistema, identificar tendências ao longo do tempo, definir alertas e tomar decisões com base em dados objetivos. Elas ajudam a entender o estado atual e a saúde do sistema.

Anatomia de uma métrica

- Nome: Facilita a identificação da métrica.
- Valor: Representa a medida quantitativa.
- Timestamp: Permite análise temporal.
- Labels: Adicionam contexto para filtragem e agregação.
- Tipo: Define a interpretação e o uso adequado da métrica.

2024-05-30T12:34:56Z http_requests_total {status="200", path="/api/v1/resource", region="sa-east-1"} 100

Prometheus



Traces

Traces são uma forma de monitoramento que captura e rastreia as chamadas entre diferentes serviços em um sistema distribuído. Eles fornecem uma visão detalhada do fluxo de uma solicitação, desde o início até o fim, incluindo todas as interações entre os serviços ao longo do caminho. Cada parte do trace (ou rastreamento) é chamada de "span" e representa uma unidade de trabalho, como uma chamada de função, uma requisição HTTP, ou uma consulta ao banco de dados.

Traces

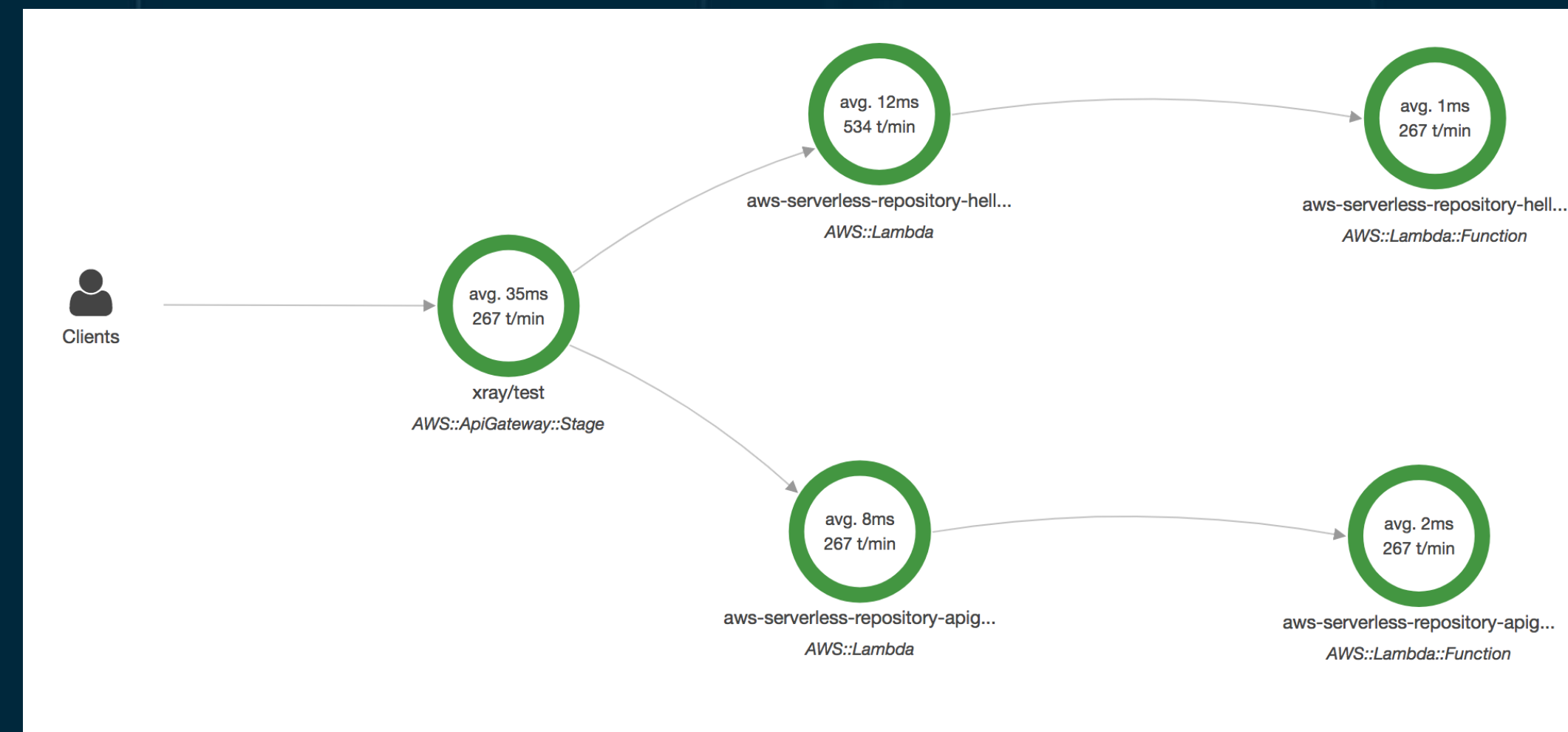
Componentes de um trace distribuído:

- Trace ID: Um identificador único para uma solicitação ou transação que atravessa vários serviços.
- Spans: Unidades individuais de trabalho dentro de um trace. Cada span tem seu próprio identificador (span ID), além de detalhes como início e término, duração e metadados.
- Metadados: Incluem informações como nome do serviço, nome da operação, status (sucesso ou falha), e qualquer outro dado relevante.
- Contexto de Correlacionamento: Informação passada entre serviços para manter o contexto do trace, geralmente incluída nos headers das requisições.

Traces

Imagine um sistema de e-commerce onde um cliente faz um pedido. A solicitação pode passar por vários serviços:

- Frontend Service: Recebe a requisição do cliente.
- Auth Service: Verifica a autenticidade do cliente.
- Order Service: Processa o pedido.
- Inventory Service: Verifica a disponibilidade do produto.
- Payment Service: Processa o pagamento.
- Notification Service: Envia uma confirmação por email.



Metodologias de observabilidade

4 Golden Signals

- Latência: O tempo que leva para atender uma solicitação. É importante distinguir entre a latência de solicitações bem-sucedidas e falhas.
- Tráfego: Medida da demanda no sistema, expressa em uma métrica específica do sistema. Para um serviço web, pode ser o número de requisições HTTP por segundo.
- Erros: Taxa de solicitações que falham, seja explicitamente (como erros HTTP 500), implicitamente (como respostas HTTP 200 com conteúdo errado), ou por política (como respostas que excedem um tempo limite).
- Saturação: Mede a fração de utilização do sistema, destacando os recursos mais limitados (como memória, cpu ou storage).

Metodologias de observabilidade



Metodologias de observabilidade

USE:

- Utilização
- Saturação
- Erros

RED:

- Rate
- Erros
- Duração

VALET:

- Volume (tráfego)
- Availability
- Latência
- Erros
- Tickets

Obrigado !

