

Cultura DevOps

FAST - MÓDULO 3



C E S A R



MENTORES



CRIS APOLINÁRIO

Administradora de Sistemas
Especialista DevOps



ROMULOS MACHADO

SRE Especialista

MENTORES



ANDRÉ CASTRO

Administrador de Sistemas
Sênior DevOps



FRANK JÚNIOR

Engenheiro de Software
Sênior DevOps

INTRODUÇÃO

Necessidade

O que a sua solução se propõe a resolver? A identificação do problema a ser resolvido é o ponto chave de qualquer proposta de valor.

Abordagem

Qual será a abordagem (única ou não) que usaremos para resolver o problema do cliente ou atender às suas necessidades?

Benefícios

Quais serão os principais benefícios da nossa abordagem de projeto ou solução para o cliente e para os demais stakeholders?
(mostrar o possível impacto quantitativamente pode ser melhor)

Concorrência

Alguém já resolveu o problema que você se propõe a solucionar?
Identificar quem já está no mercado, quais suas propostas de valor e quais são seus gaps representam oportunidades.



A Cultura DevOps

Cultura DevOps

A palavra "DevOps" é a combinação dos termos "desenvolvimento" e "operações". No entanto, ela representa um conjunto de ideias e práticas que ultrapassam o significado desses dois termos. O termo DevOps inclui segurança, maneiras colaborativas de trabalhar, análise de dados e muitas outras práticas e conceitos.

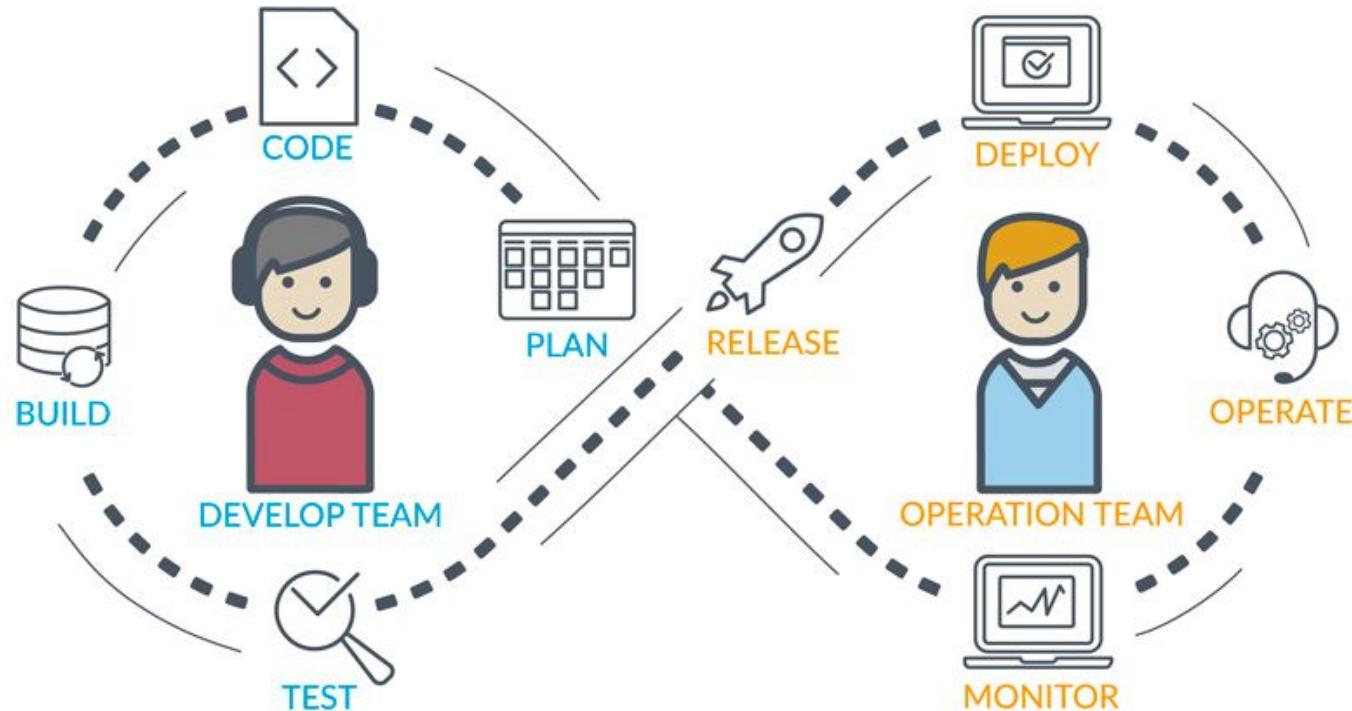
A cultura DevOps descreve abordagens que ajudam a acelerar os processos necessários para levar uma ideia do desenvolvimento à implantação em um ambiente de produção no qual ela seja capaz de gerar valor para o usuário.

Benefícios da Cultura DevOps

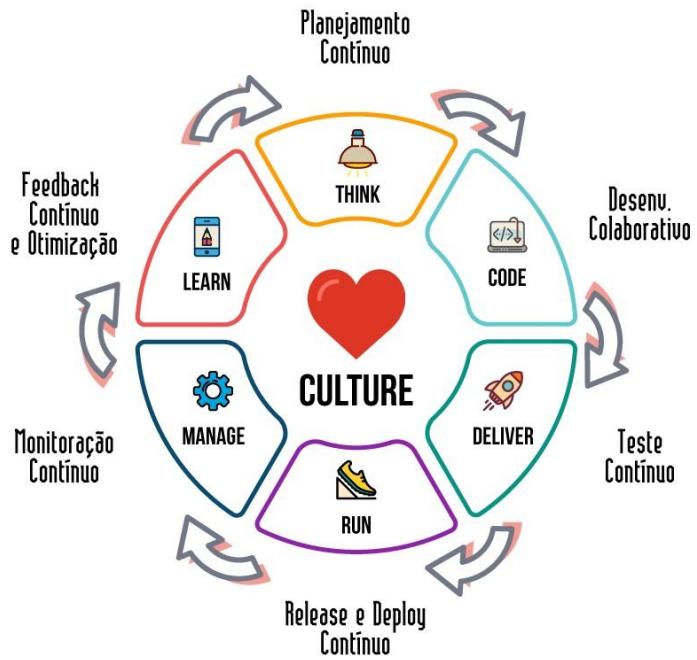
DevOps é um processo constante e que precisa ser adaptado às necessidades e expectativas de cada empresa e traz alguns benefícios, como:

- Economia de Recursos
- Otimização de processos
- Melhora na qualidade
- Mais rapidez na produção
- Aumento da motivação

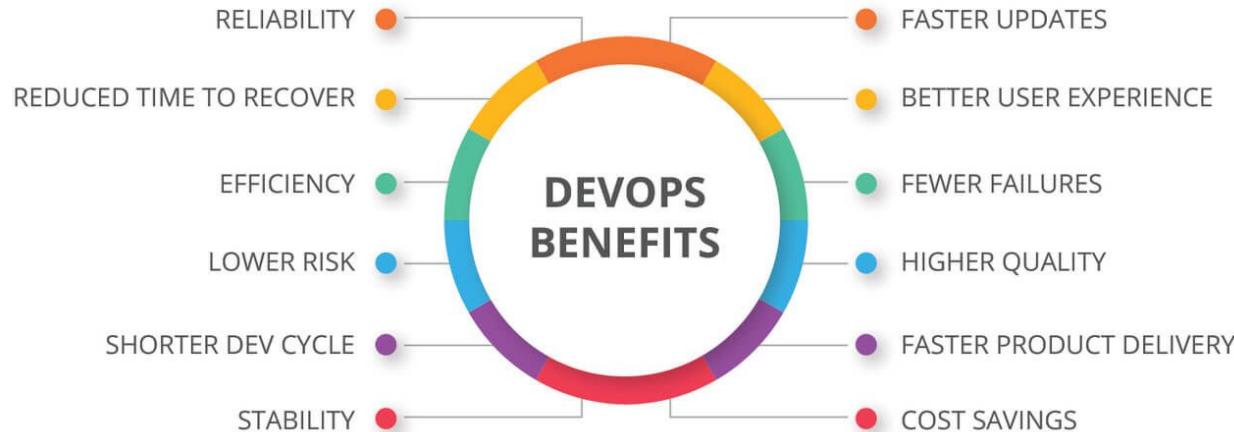
A CULTURA DEVOPS



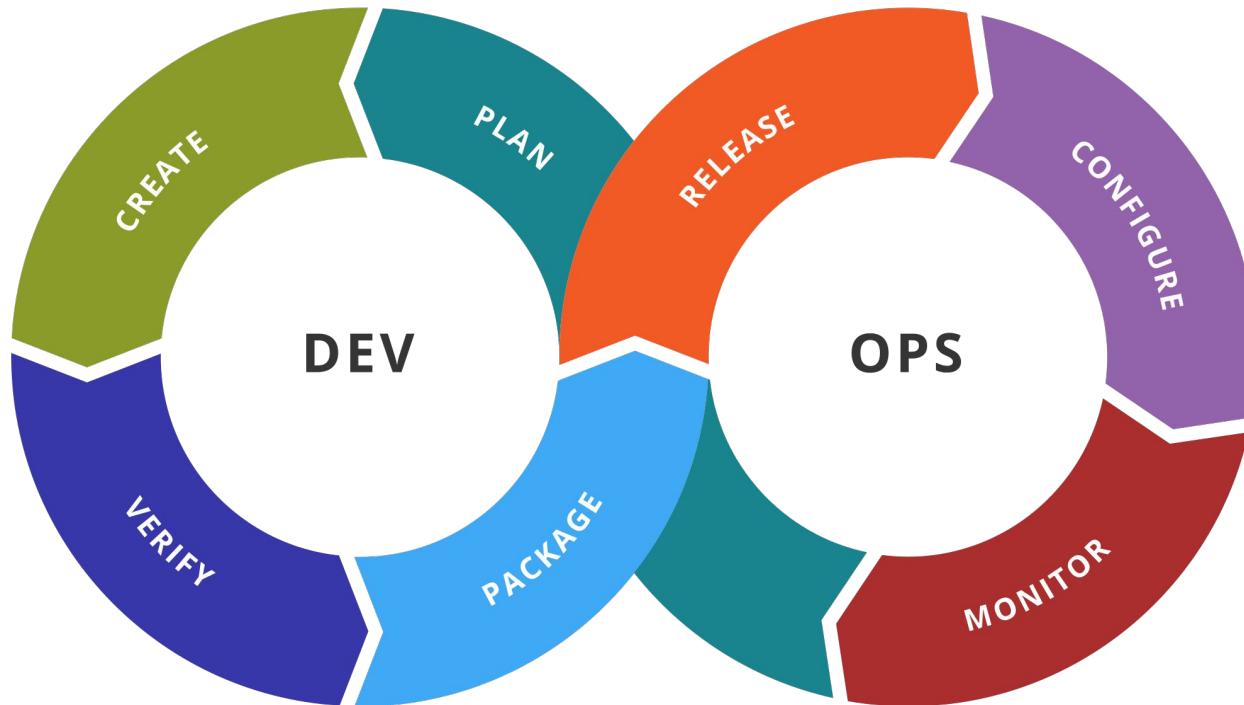
FLUXO



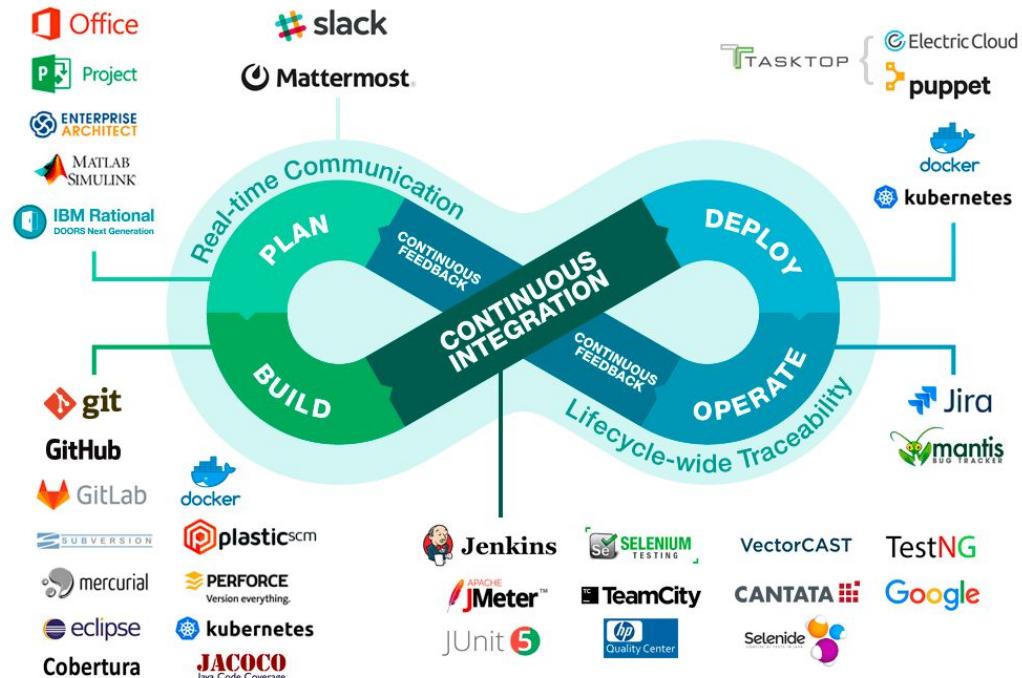
BENEFÍCIOS



FLUXO



FLUXO COM FERRAMENTAS





Processo de Desenvolvimento de Software

Engenheiro de Software



```
# -*- coding: utf-8 -*-
from odoo import http
from odoo.exceptions import ValidationError
from datetime import datetime
import calendar
import math
import pytz
import io, base64

class AdmissionExtensionOnlineController(http.Controller):
    @http.route('/get/type_wise_program', website=True, auth='none')
    def type_wise_program(self, **kwargs):
        if len(kwargs['types']) <= 0:
            return "None"
        types = kwargs['types']
        program_list = []
        domain = []

        if types == 'local_bachelor_program_hsc':
            domain = [('course_id.is_local_bachelor_program_hsc', '=',
        elif types == 'local_bachelor_program_a_level':
            domain = [('course_id.is_local_bachelor_program_a_level',
        elif types == 'local_bachelor_program_diploma':
            domain = [('course_id.is_local_bachelor_program_diploma',
        elif types == 'local_masters_program_bachelor':
            domain = [('course_id.is_local_masters_program_bachelor',
        elif types == 'international_bachelor_program':
            domain = [('course_id.is_international_bachelor_program',
        elif types == 'international_masters_program':
            domain = [('course_id.is_international_masters_program',
            domain.append(('state', '=', 'application'))
            if admission_register_list = http.request.session.get('admission_register_list'):
                for program in admission_register_list:
                    if program['program_type'] in types:
                        program['domain'] = domain
            else:
                domain = []
        else:
            domain = []
        return {
            'domain': domain,
            'types': types
        }
```

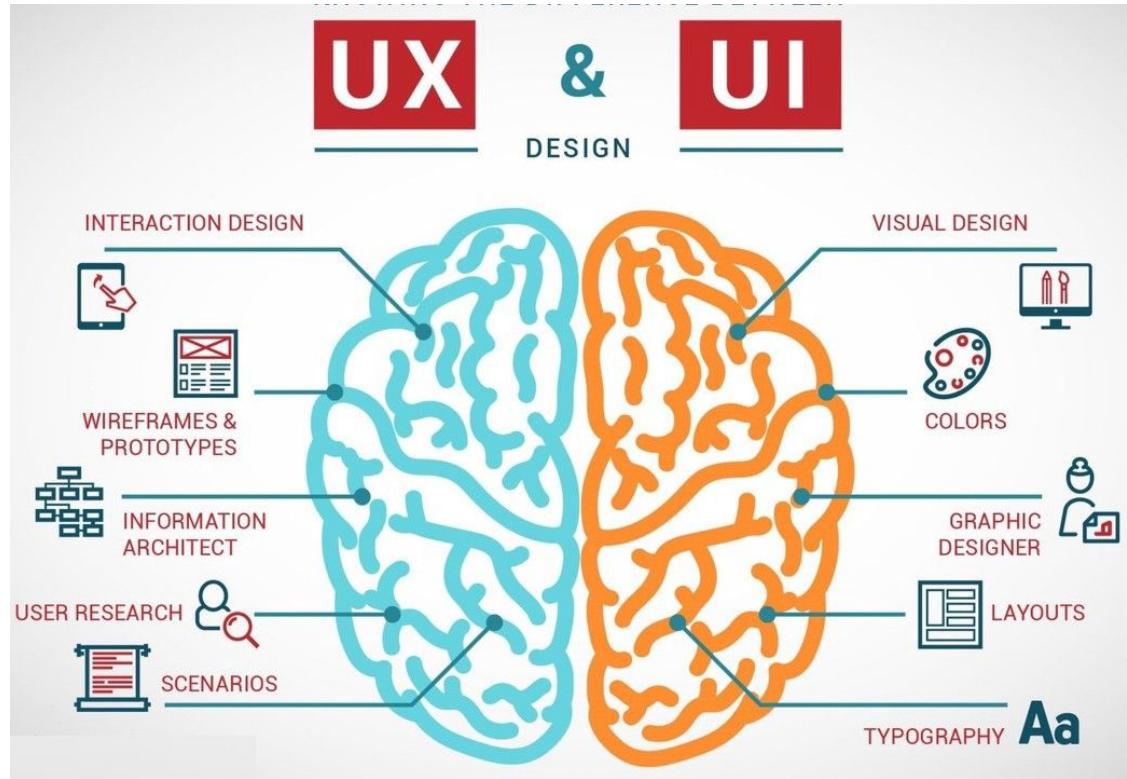
Engenheiro de Software

- Frontend
- Backend
- Fullstack
- Mobile

Engenheiro de Testes



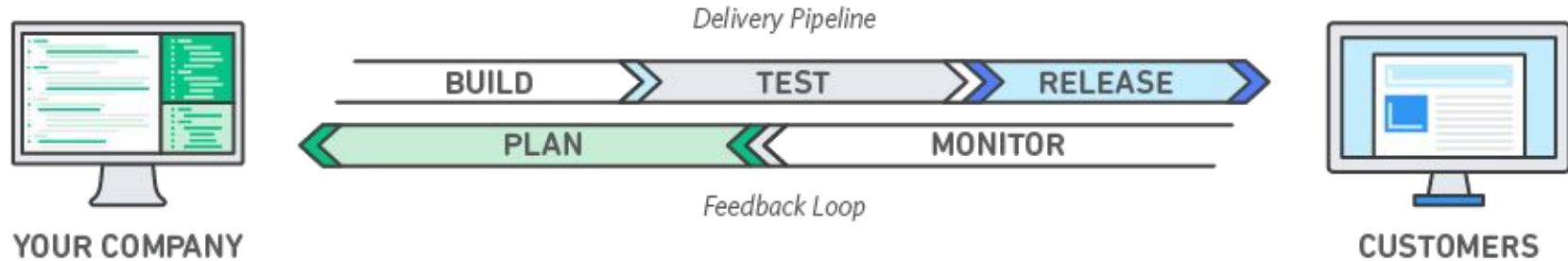
Design



Dentro de um processo de desenvolvimento de software quatro pontos podemos considerar como indispensáveis e serão estes que iremos abordar:

- Controle de versão
- Testes / Qualidade
- Construção do software
- Artefatos

O PROCESSO



O que é Controle de Versão?

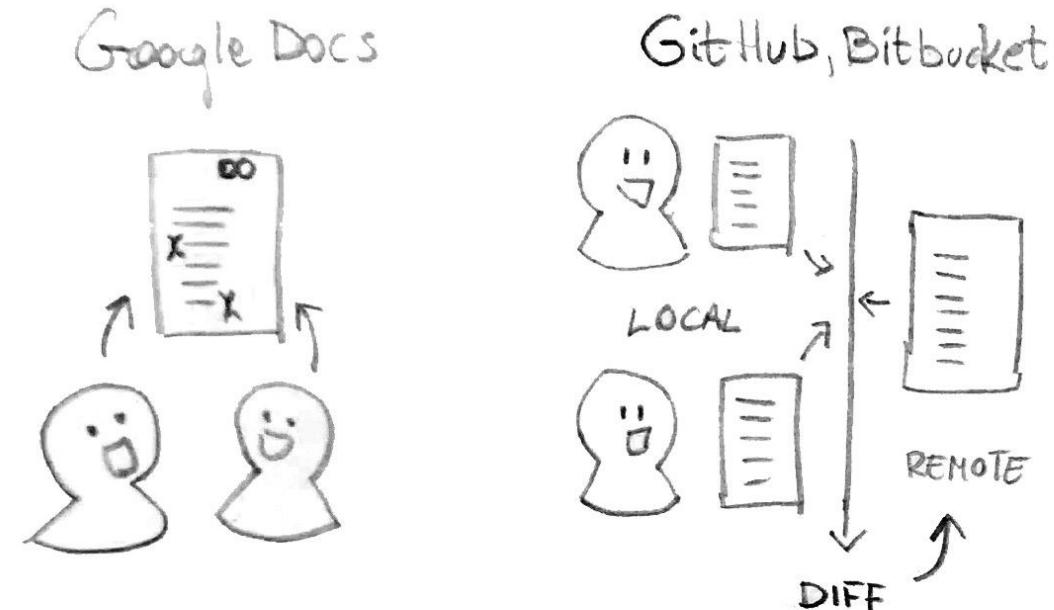
Nada mais é do que a **gestão do código fonte de um software**, onde é possível rastrear mudanças feitas durante todo o processo de desenvolvimento.

No mercado existem algumas ferramentas com esta finalidade, mas dentre elas a que está em uso em quase todo o mercado mundial é o **Git**.

Benefícios do Controle de Versão

- Histórico de alterações;
- Trabalho simultâneo;
- Ramificação;
- Rastreabilidade;
- Automação de Pipelines;
- Comunicação entre a equipe;
- Métricas.

CONTROLE DE VERSÃO



O que é Git?

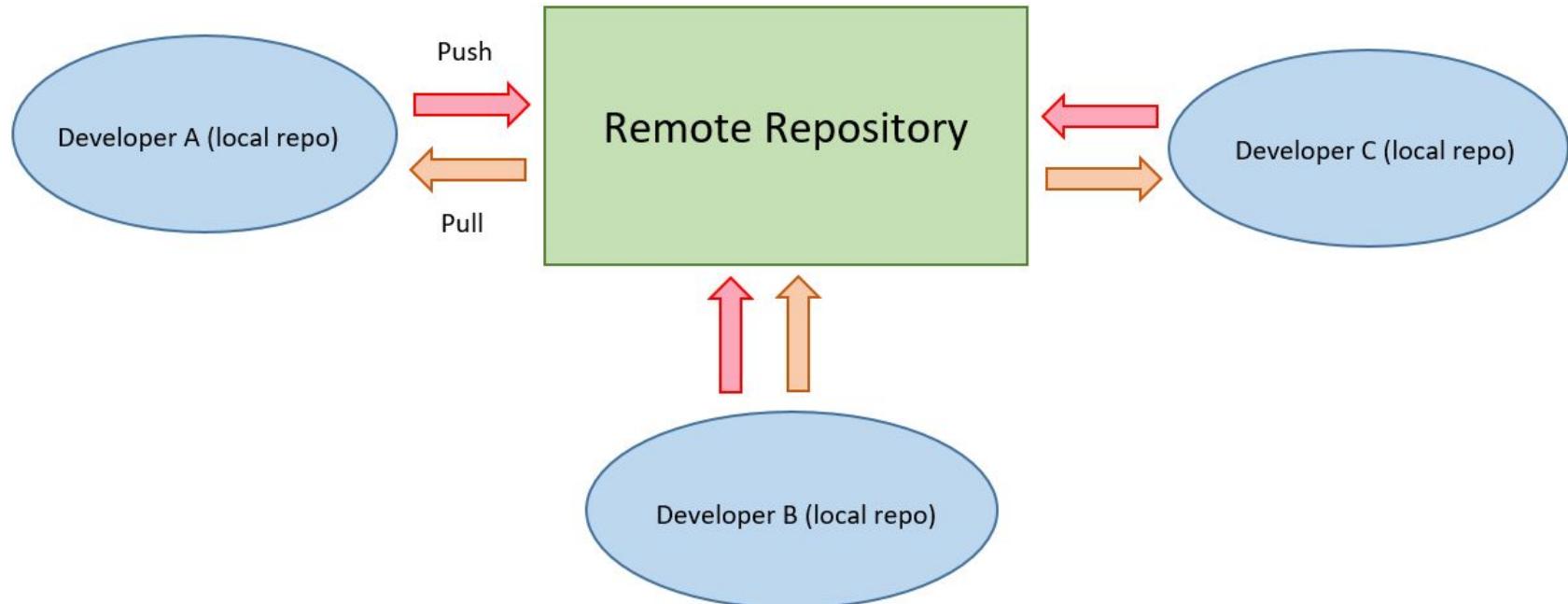
Um dos sistema de controle de versão mais utilizados no mundo e que se popularizou bastante muito, por conta do **GitHub** que é uma plataforma que permite os desenvolvedores hospedarem seus projetos e compartilhar com algumas ou qualquer pessoa.

Entendendo os conceitos

Para se trabalhar com Git é preciso entender alguns conceitos como:

- **Repositórios:** ambiente/projeto criado para armazenar o código fonte;
- **Branch:** ramificação/versão do projeto;
- **Merge:** unir modificações de uma branch a outra;
- **Fork:** copiar o projeto.

CONTROLE DE VERSÃO

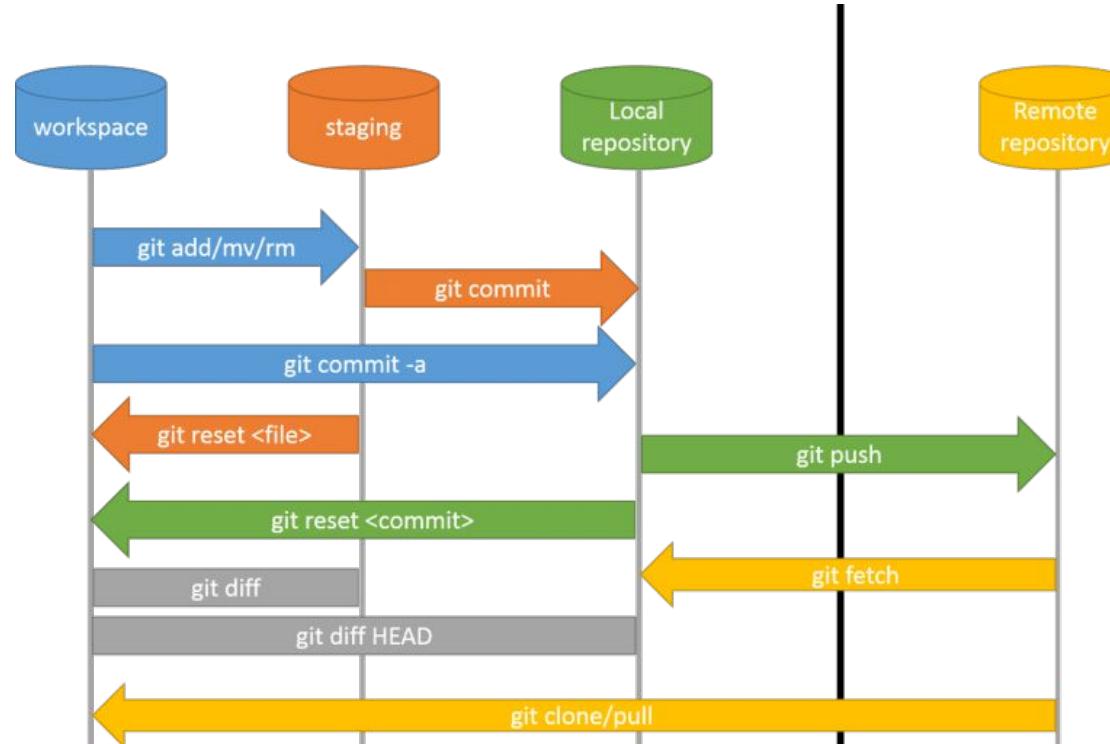


Principais comandos

Para se trabalhar com Git é preciso entender alguns conceitos como:

- **init**: inicia um *repositório novo* (local ou remoto);
- **clone**: clona o código de um *repositório*;
- **commit**: move os arquivos do *index* para o *repositório local*;
- **add**: adiciona o arquivo ao *index* e prepara o vínculo ao commit;
- **push**: envia do *repositório local* para o *remoto*;
- **pull**: traz do *repositório remoto* para o *local*;
- **merge**: uni alterações de um arquivo ao original de um projeto;
- **log**: visualiza histórico de commits;

CONTROLE DE VERSÃO



Testes / Qualidade

Por mais que se planeje a construção de um software, erros são passíveis de ocorrer. Pode ser um bug em um game, uma falha que feche uma app ou um erro que impossibilite você salvar um arquivo.

O teste de software serve justamente para tentar encontrar possíveis erros que uma app recém-desenvolvida possa apresentar, de modo a conseguir corrigi-lo antes que seja lançado no mercado, ficando disponível para uso do público.

Podem ser classificados em:

- **Testes Funcionais:** verificam as funcionalidades de um sistema e buscam erros de desempenho;
- **Testes Não-funcionais:** analisam a usabilidade, velocidade e outros aspectos que independem das ações executadas pelo sistema.

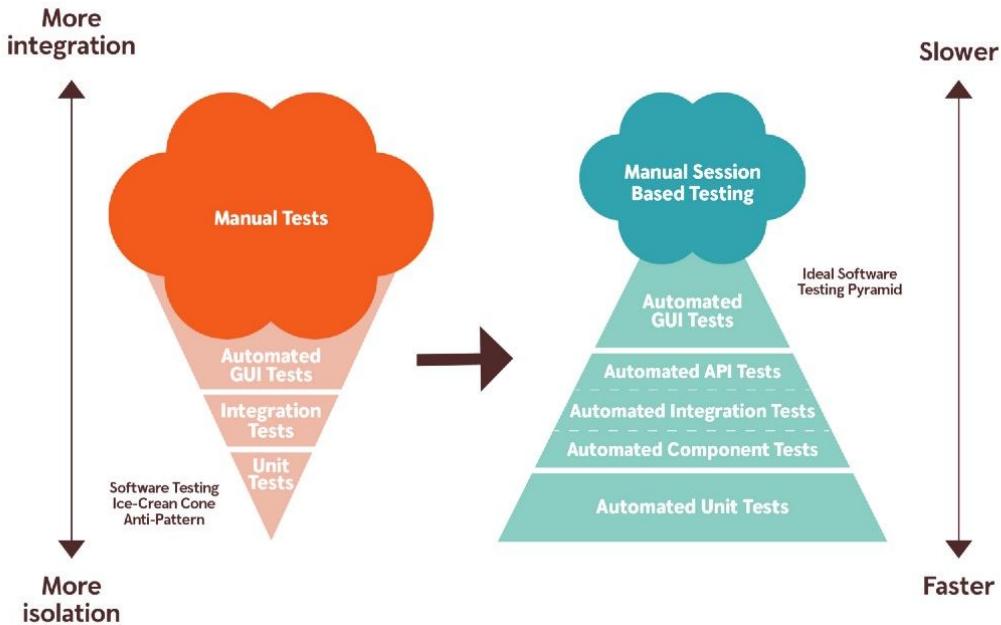
Tipos de Testes

- **Teste de Caixa Branca ou Estrutural:** analisa o fluxo de dados, a qualidade da estruturação do código-fonte, segurança, complexidade da manutenção, entre outros;
- **Teste de Caixa Preta ou Funcional:** analisa os requisitos do sistema, ou seja, se o software cumpre as funções que deve executar;
- **Teste de Regressão:** testa cada versão de um sistema, quando suas funcionalidades passam por mudanças ou são incorporadas novas tarefas.
- **Testes Unitários:** testa-se unidades menores de um software, de modo isolado, para ver se todas funcionam adequadamente;
- **Teste de Integração:** após a verificação das funções isoladas, o próximo passo é analisar sua integração e se todas funcionam nesta nova condição;
- **Testes de Carga:** é feito para avaliar os limites de uso do software, o quanto ele suporta em volume de informações, tráfego etc. sem que apresente erros;
- **Testes de Usabilidade ou Aceitação:** são realizados por um grupo de usuários que verifica o funcionamento do software.

Automação de testes

O processo de automação de testes consiste no uso de softwares específicos capazes de controlar e gerenciar determinados testes. Isso é possível a partir da aplicação de estratégias e algumas ferramentas que facilitam a comparação entre resultados previstos e resultados reais.

PIRÂMIDE DE TESTES



Ferramentas de Testes

Além de otimizar o trabalho da equipe de desenvolvedores, veja em que outros quesitos elas podem auxiliar:

- Conferem segurança e confiabilidade ao produto;
- Otimizam a gestão dos recursos ao longo do projeto;
- Evitam o excesso de trabalho manual;
- Identificam problemas de ordens diversas;
- Oferecem tempo hábil para fazer reparos;
- Garantem feedback em todo o processo de desenvolvimento;
- Podem render um grande volume de dados;
- Ajudam a checar resultados e configurações;
- Garantem uma rede de segurança ao sistema.

FERRAMENTAS DE TESTES



appium

<https://appium.io/>



Robotium

<https://github.com/RobotiumTech/robotium>



Selenium

<https://github.com/SeleniumHQ/>



<https://www.telerik.com/teststudio>

Análise de Código

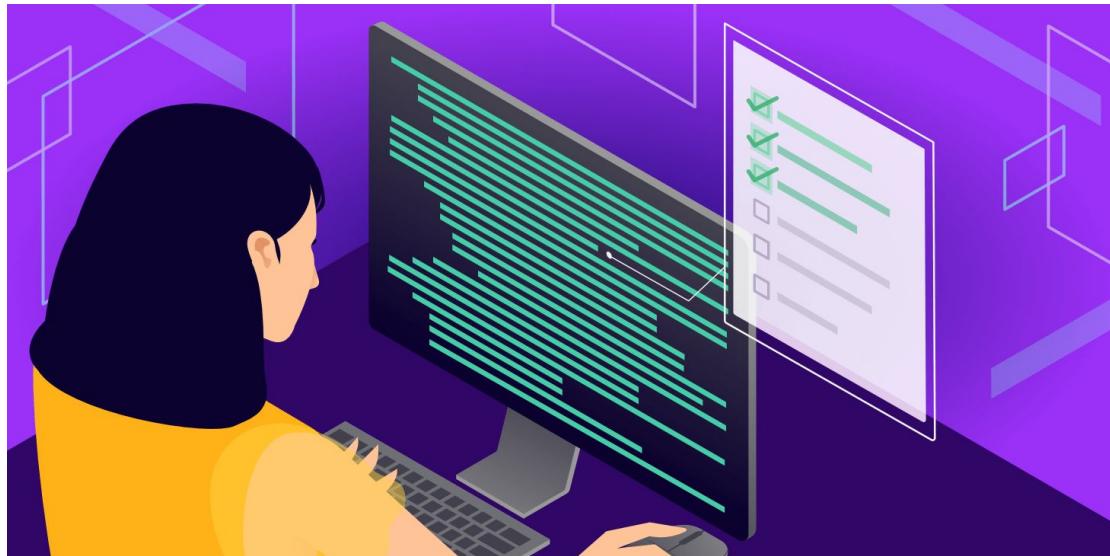
Quanto mais cedo um problema é encontrado no software, mais barata é a sua correção (um problema encontrado em desenvolvimento será muito mais barato de corrigir do que se o mesmo for encontrado em produção).

- **Análise estática;**
- **Revisão de código.**

Ferramentas para Análise de código

- Conseguem detectar e apontar oportunidades de melhoria de código.
- Elas podem ser integradas tanto a IDEs quanto ao processo de build
- Bem utilizadas, aceleram a capacitação do time para escrita de código limpo.

FERRAMENTAS DE ANÁLISE



Construção do Software

- Build;
- Versão "compilada" de um software ou parte dele que contém um conjunto de recursos que poderão integrar o produto final.

ARTEFATOS

Resultado de um conjunto de atividades proveniente das tarefas envolvidas no processo de desenvolvimento de um sistema de software. Isto é:

- Um subproduto concreto de um produto de software;
- Provê informações sobre e internamente ao software;
- Um documento;
- Um modelo;
- Um executável.

FERRAMENTAS





CI / CD

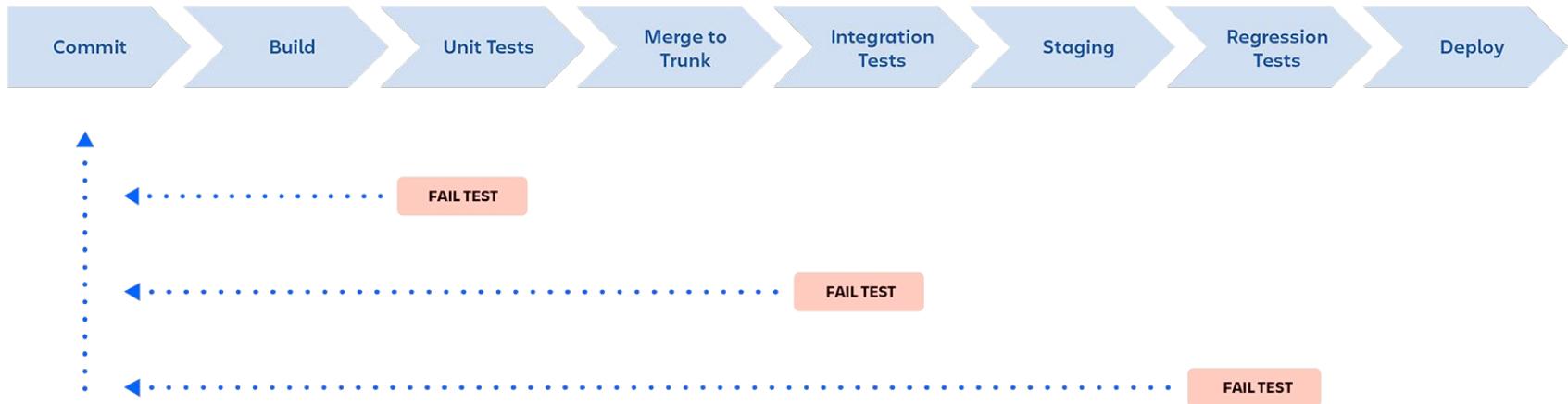
Pipelines

Tem como objetivo automatizar todo o processo de desenvolvimento até a entrega e implementação em produção de uma aplicação.

As pipelines são compostas por processos que agilizam todas as etapas do desenvolvimento de um software:

- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- *Continuous Testing*
- *Continuous Feedback*
- *Continuous Monitoring*

PIPELINE



CI / CD

São práticas que tem por finalidade agilizar a entrega de aplicações além de entregar também com mais frequência.

Nesta prática temos:

- **CI** - *Continuous Integration*
- **CD** - *Continuous Delivery*
- **CD** - *Continuous Deployment*

Continuous Integration

O **CI** (*Continuous Integration*) é o processo de automação que visa o desenvolvimento da aplicação, sendo ele o primeiro passo na construção de uma pipeline dentro de um projeto de desenvolvimento.

Continuous Delivery

O **CD** (*Continuous Delivery*) é processo de entrega automática das mudanças feitas no desenvolvimento ao repositório de produção sendo este o passo seguinte a integração contínua que precisa estar devidamente implementada na pipeline de desenvolvimento.

Continuous Deployment

O **CD** (*Continuous Deployment*) é o processo de implementação automática para produção, ou seja, é o complemento da entrega automática e o último passo de todo o processo.

Continuous Testing

O **CT** (*Continuous Testing*) é a implementação de testes automáticos para cada etapa do processo de desenvolvimento, diminuindo assim possíveis problemas de integração de código e também melhorando a segurança da aplicação que está em desenvolvimento.

Continuous Feedback

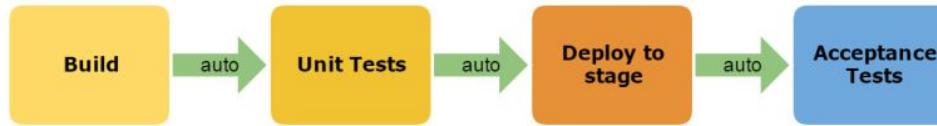
O **CF** (*Continuous Feedback*) é o retorno a do impacto daquele produto entregue, para desse modo, melhorar o código para as próximas entregas.

Continuous Monitoring

O **CM** (*Continuous Monitoring*) é a implementação de monitoramento para uma rápida detecção de vulnerabilidade, comportamentos errados da aplicação, entre outros possíveis problemas, visando garantir a saúde e performance da aplicação entregue.

Integração - Entrega - Implementação

Continuous Integration



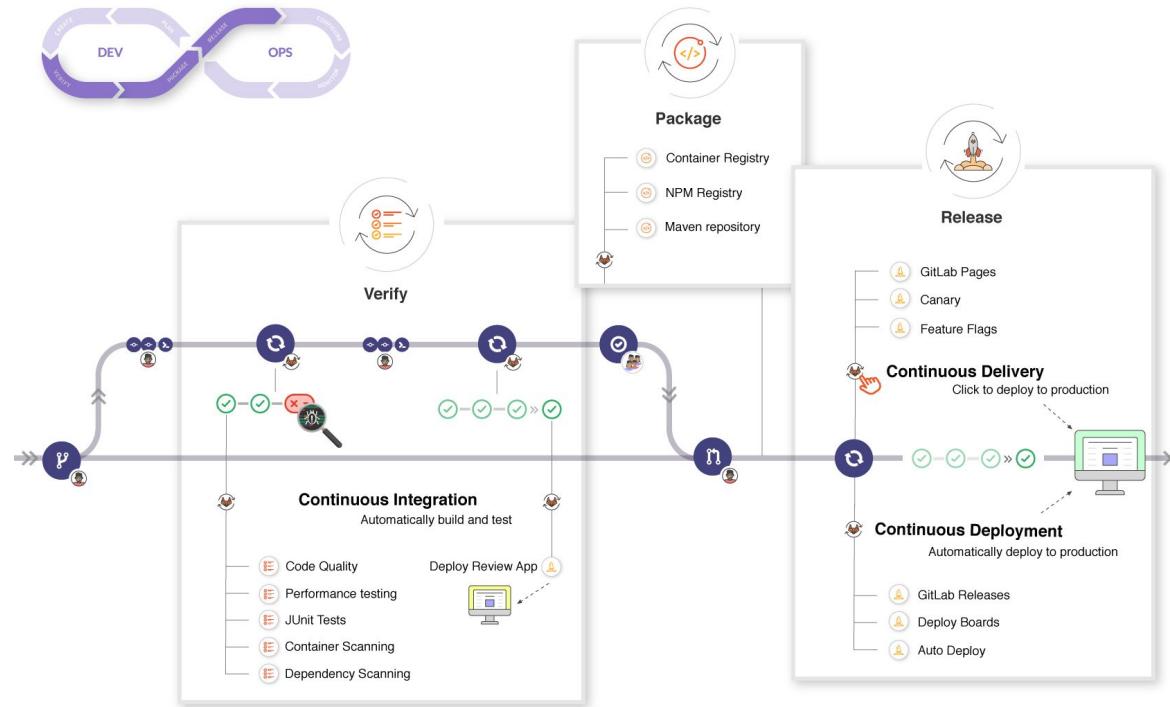
Continuous Delivery



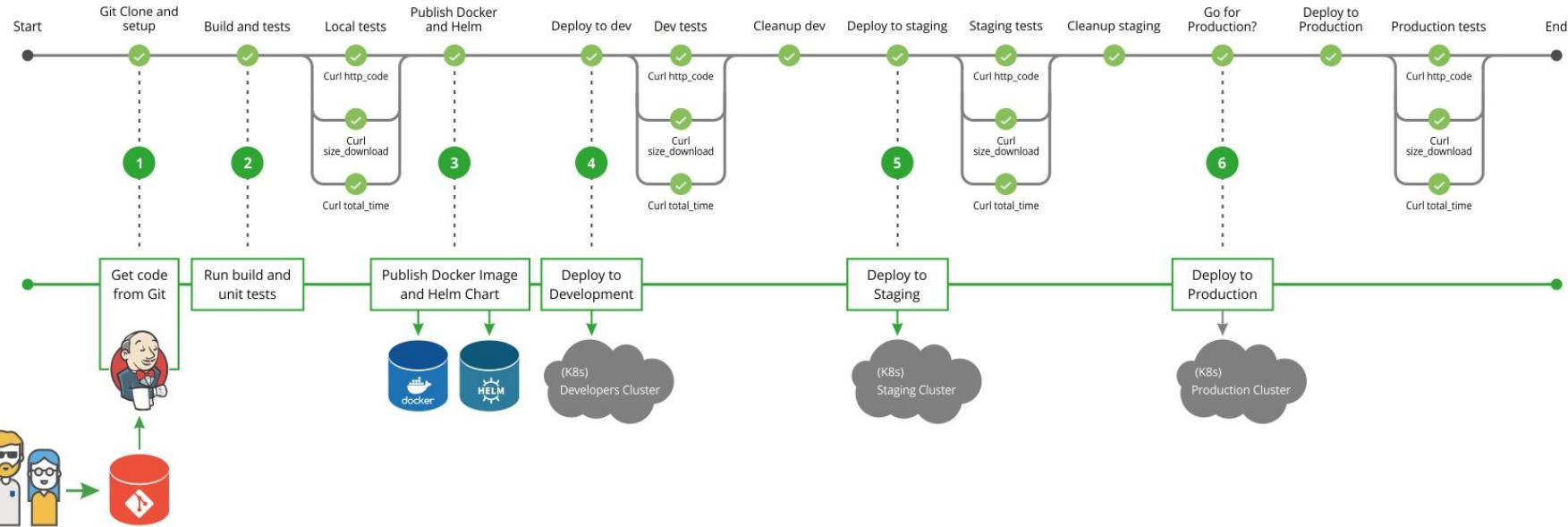
Continuous Deployment



Integração - Entrega - Implementação



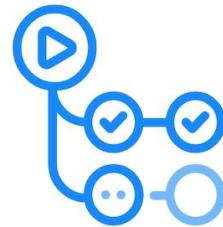
PIPELINE



FERRAMENTAS



Jenkins



GitHub Actions

circleci



AWS CodePipeline

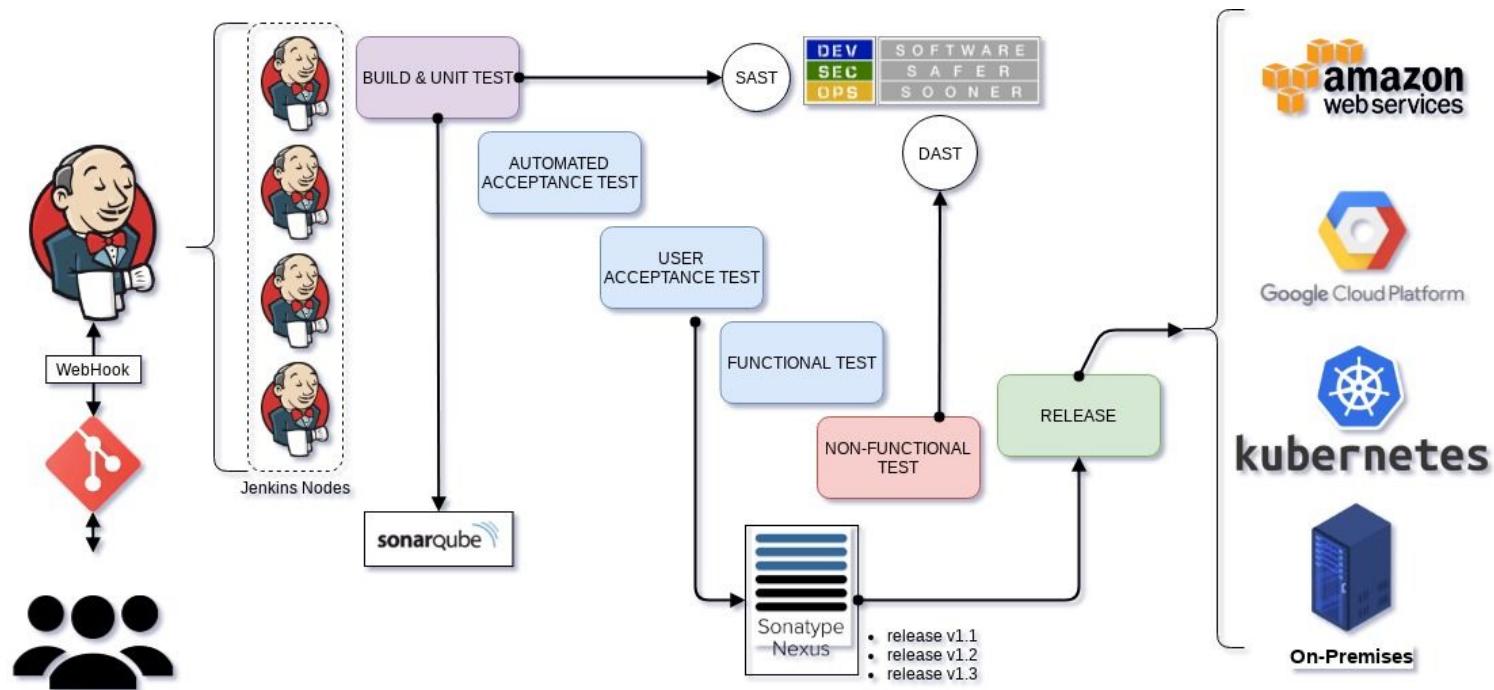


GitLab CI



Travis CI

FERRAMENTAS

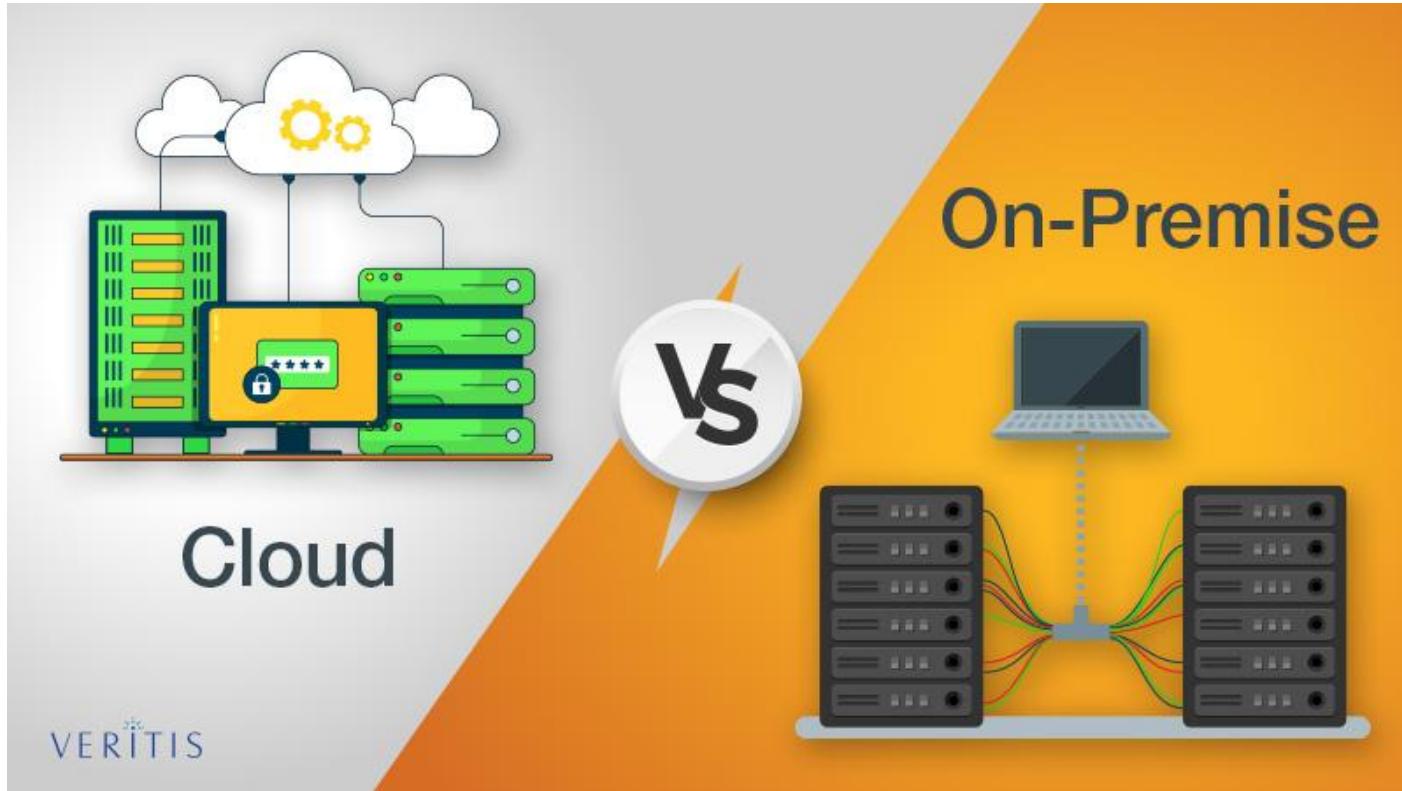




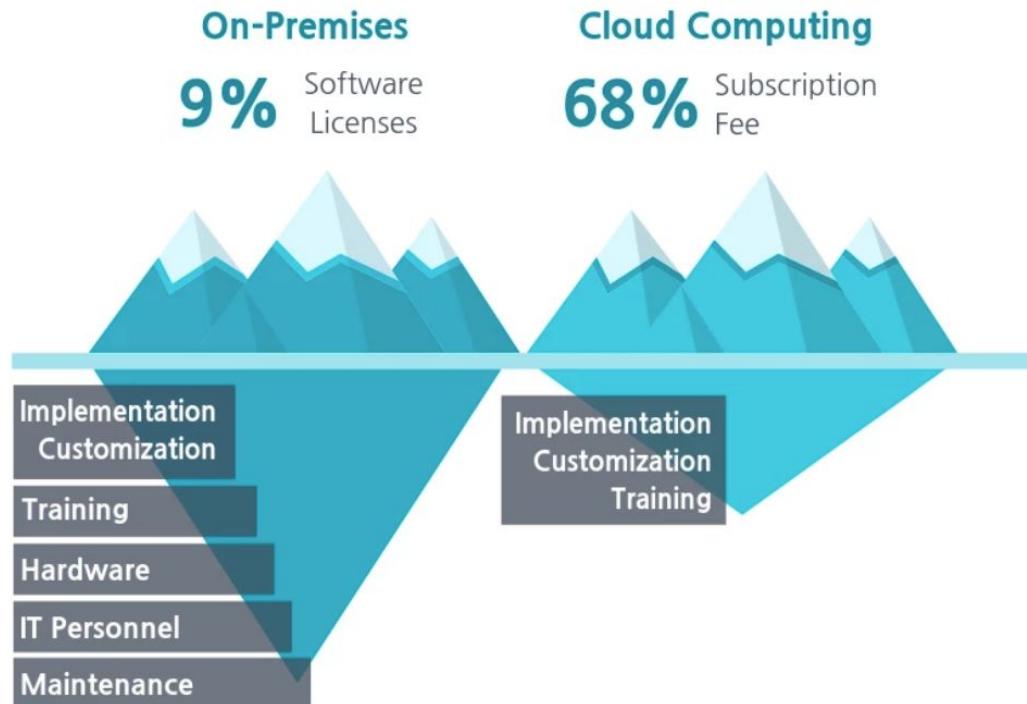
Infraestrutura On-Premise/Cloud

Infraestrutura On-Premise/Cloud

- **On-premise** refere-se a infraestruturas privadas mantidas pela própria empresa em suas instalações e que podem ser mantidas de modo independente.
- **Cloud (Nuvem)** refere-se a infraestruturas robustas que ficam instaladas em data centers de provedores contratados, onde o acesso é feito via Internet.



ON-PREMISE VS CLOUD



Tipos de Cloud

Dentro deste cenário de Cloud Computing podemos classificar a cloud em 2 tipos:

- **Pública:** é definida como uma série de serviços de computação oferecidos por terceiros à Internet pública, os quais são disponibilizados a qualquer pessoa que queira utilizá-los ou comprá-los.
- **Privada:** é um modelo de implantação sob demanda em que os serviços e a infraestrutura de computação em nuvem são hospedados de maneira privada, geralmente na própria intranet ou data center de uma empresa, por meio de recursos proprietários que não são compartilhados com outras organizações.



Provedores do Mercado

Nos dias atuais a gama de provedores de *Cloud* é gigantesca, empresas que antes investiam fortemente nos conhecidos softwares de prateleira, viram neste novo mundo uma forma de crescer e com isso aumentou ainda mais a diversidade de provedores.

Dentre os provedores existentes alguns se destacam, pela diversidade dos serviços prestados e/ou pelo tempo de existência no mercado.

PRINCIPAIS PROVEDORES

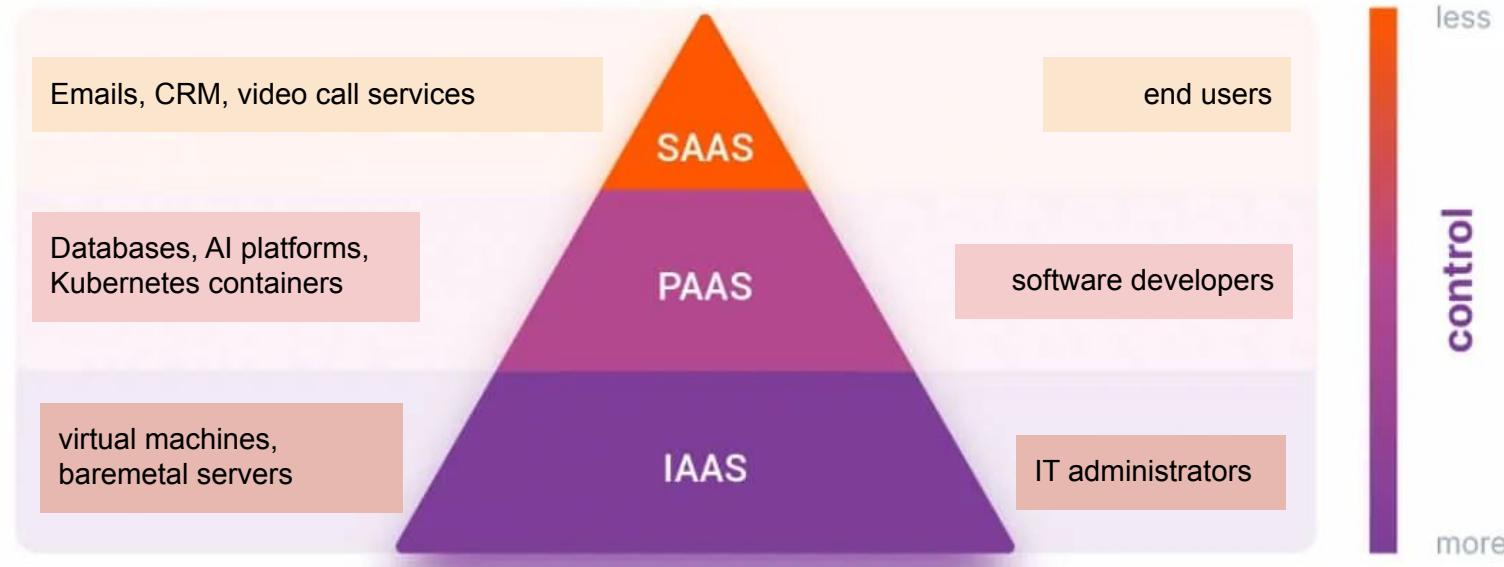


Modelos de serviço de Cloud

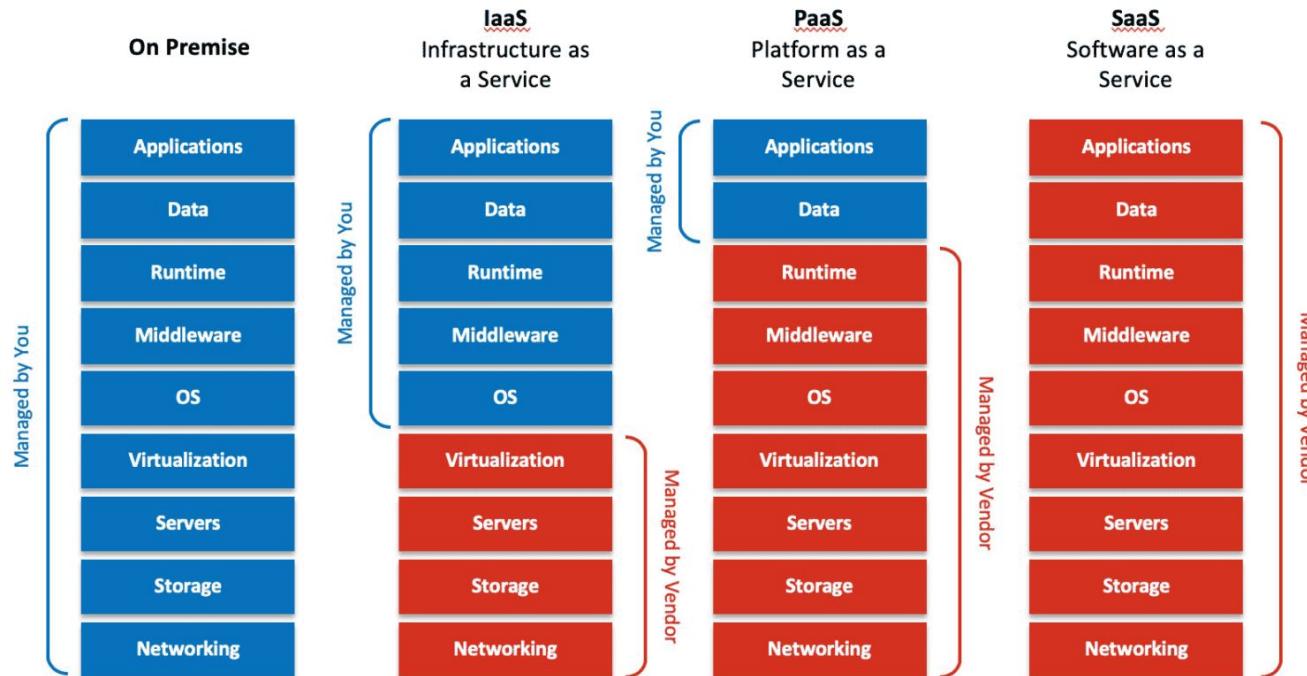
Além dos tipos de Cloud existem os serviços disponíveis na Cloud e como principais podemos citar:

- **IaaS** (*Infrastructure as a Service*)
- **PaaS** (*Platform as a Service*)
- **SaaS** (*Software as a Service*)

MODELOS DE SERVIÇO DE CLOUD



IAAS VS PAAS VS SAAS





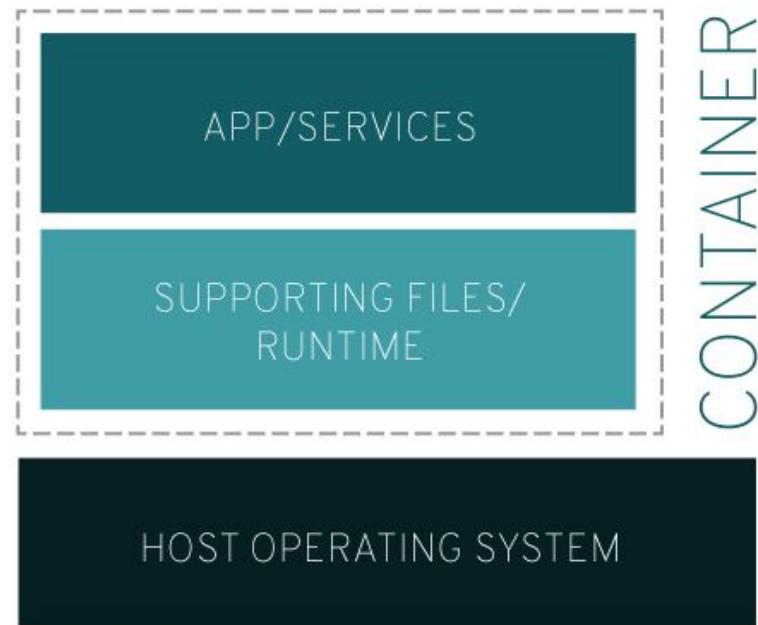
Containers e Orquestradores

O que é um container?

Seria o conjunto de um ou mais processos organizados de forma isolada do sistema onde, todos os arquivos necessários para execução encontram-se disponibilizados em uma imagem.

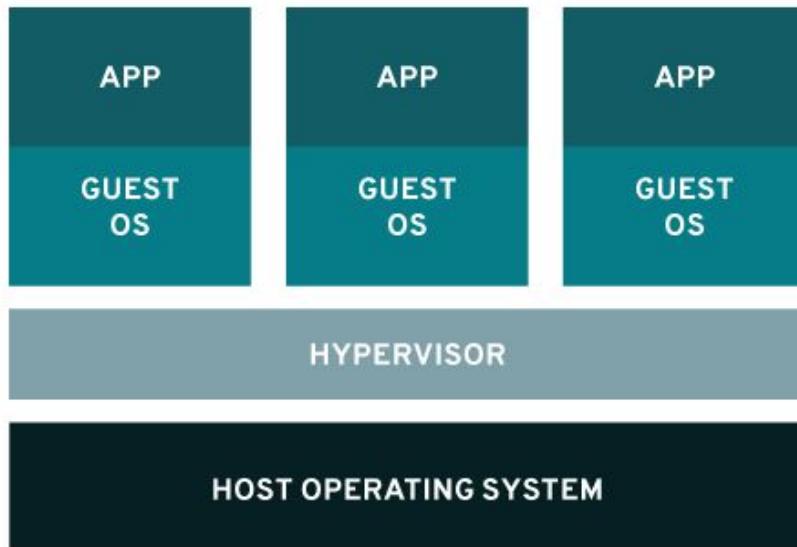
Os containers são executados de maneira nativa no sistema operacional, compartilhando-o com todos os outros containers. Assim, as aplicações e os serviços permanecem leves e são executados em paralelo com agilidade.

CONTAINER

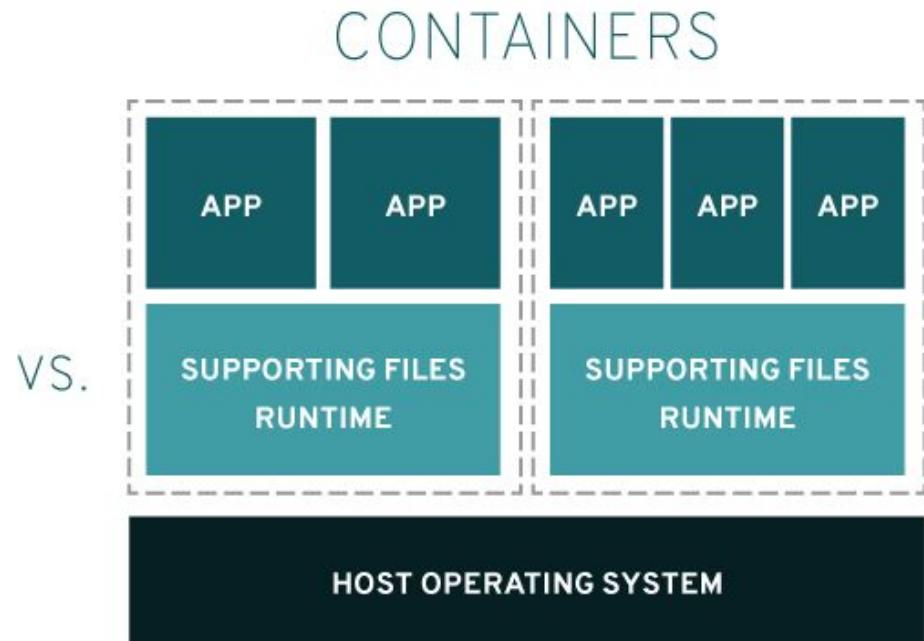


CONTAINER VS VIRTUAL MACHINE

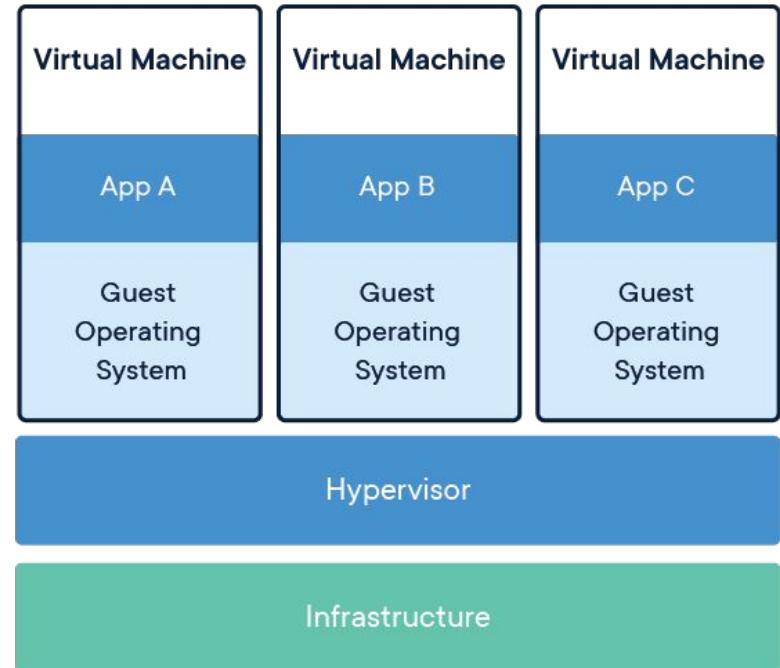
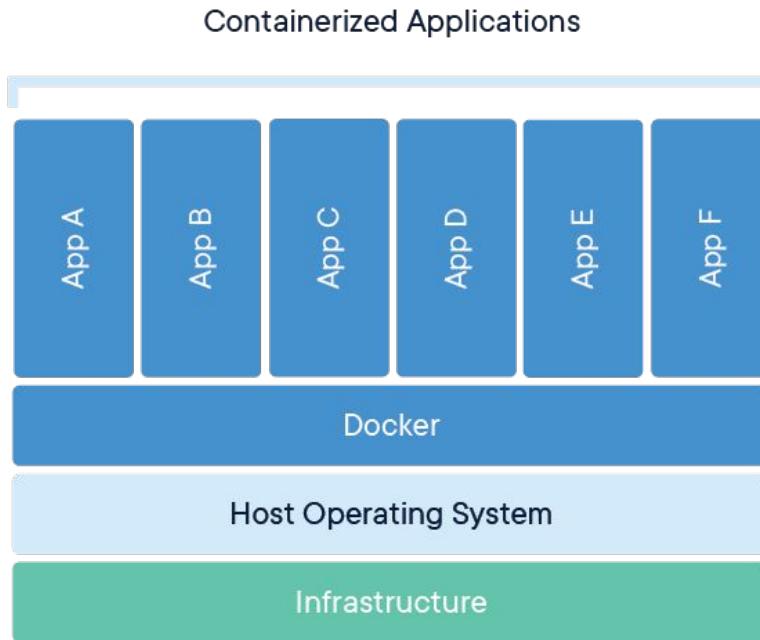
VIRTUALIZATION



CONTAINERS



CONTAINER VS VIRTUAL MACHINE

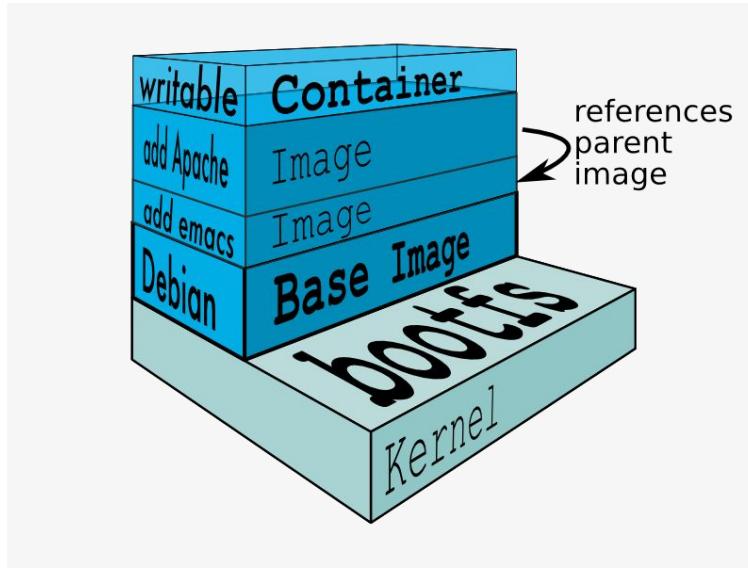


Imagens

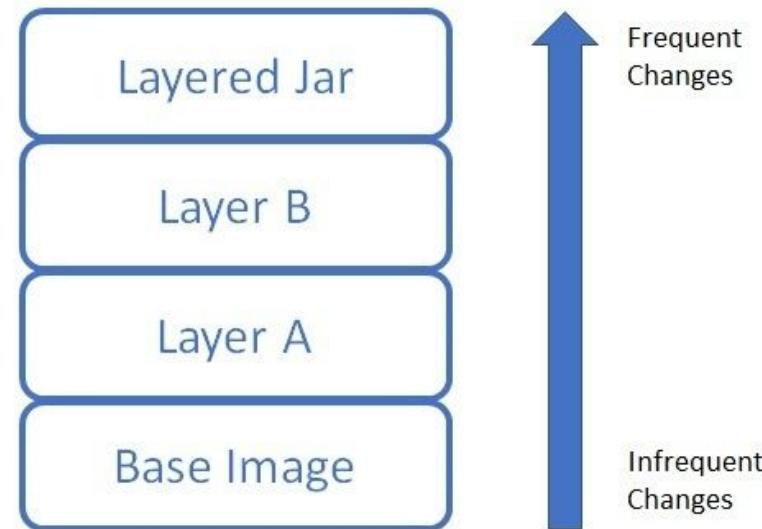
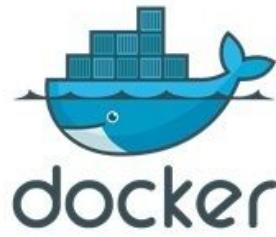
As imagens são a base da construção da aplicação que irá rodar em um container. Elas são compostas por sistemas de arquivos de camadas que ficam uma sobre as outras.

A construção destas imagens são feitas através de um arquivo de definição chamado **Dockerfile**.

DOCKER IMAGE



CAMADAS DA IMAGEM



DOCKERFILE

Dockerfile



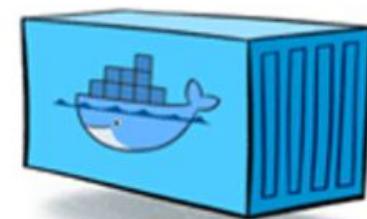
→ Build →

Image

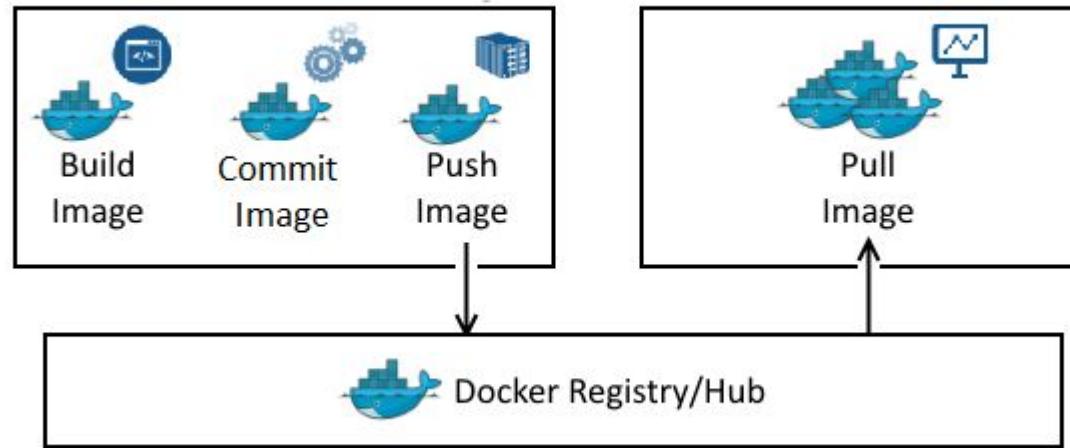


→ Run →

Container



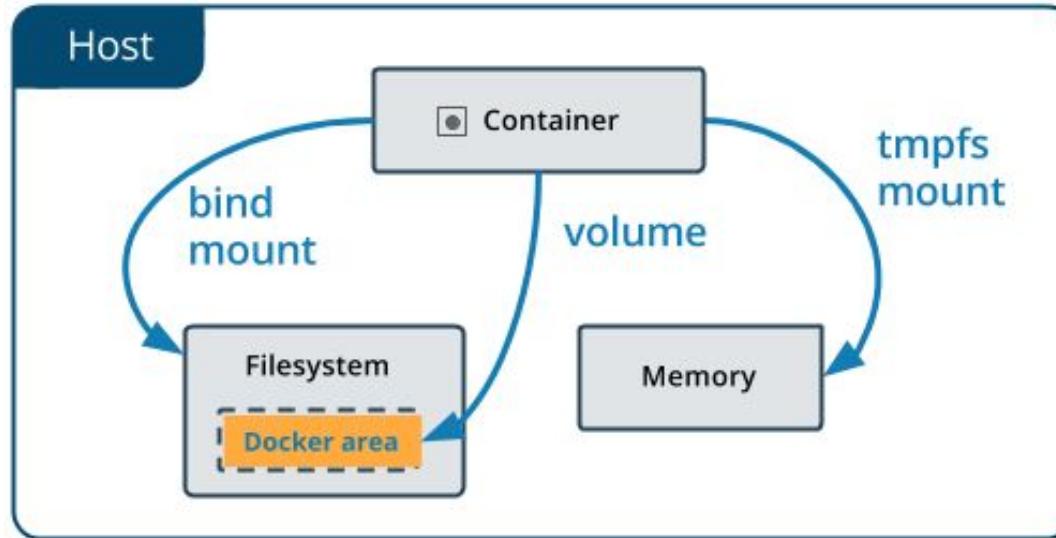
DOCKER REGISTRY



Volumes

É o mecanismo mais recomendado para a **persistência** de dados. O gerenciamento de um **volume**, ao contrário de um **bind mount**, é feito totalmente pelo docker. Lembrando que bind mounts são dependentes da estrutura de diretório e do sistema operacional da máquina host.

VOLUMES



Orquestração de containers

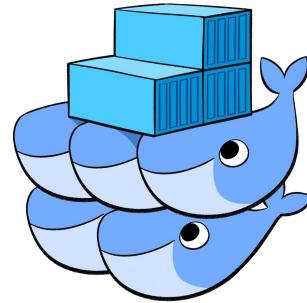
A orquestração automatiza a implantação, o gerenciamento, a escala e a rede dos containers. Com seu uso é possível:

- Provisionamento e implantação
- Configuração e programação
- Alocação de recursos
- Disponibilidade de containers
- Escala ou remoção de containers
- Balanceamento de carga e roteamento de tráfego
- Monitoramento de integridade dos containers
- Proteção das interações entre os containers
- Configuração da aplicação com base no container

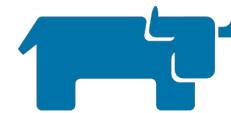
FERRAMENTAS DE ORQUESTRAÇÃO



kubernetes



MESOS

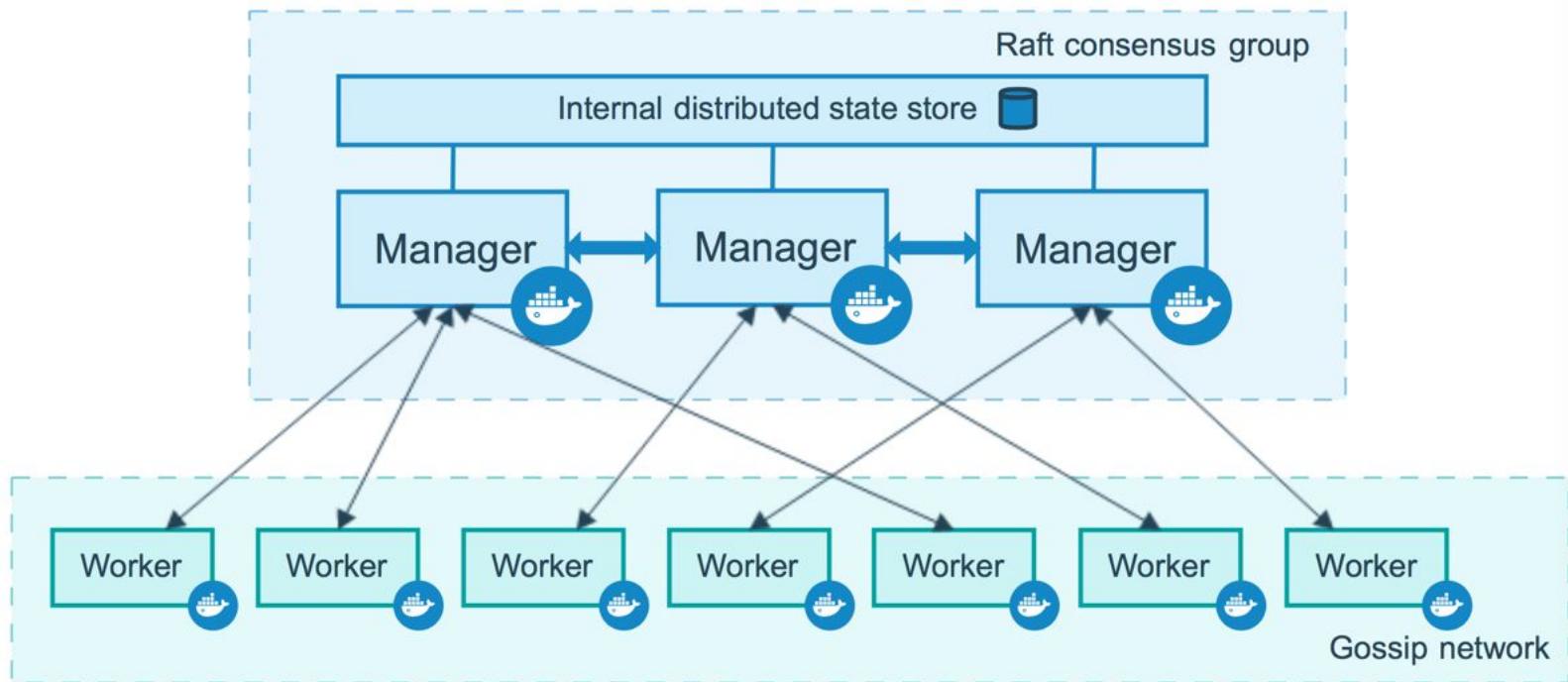


RANCHER

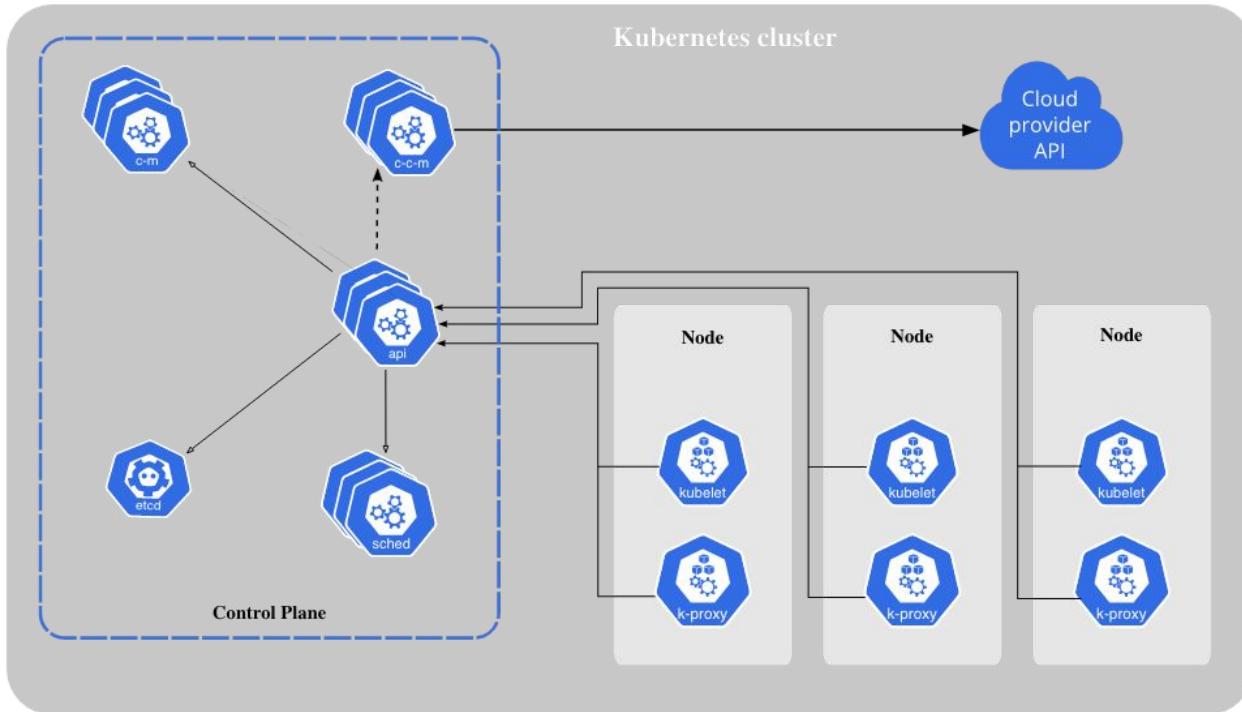


HashiCorp
Nomad

DOCKER SWARM



KUBERNETES





Automação

Automação

Automatizar é usar a tecnologia para realizar atividades com o mínimo possível de intervenção manual.

Com isso é possível acelerar processos e entregas, escalar e criar ambientes, criar fluxos de trabalho.

IaC

Infraestrutura como código é o gerenciamento e provisionamento de infra através de códigos.

Com a criação de arquivos configuração onde toda a construção/manutenção da infraestrutura é descrita, facilita tanto na manutenção do ambiente quanto na distribuição.

O controle de versão é uma parte importantíssima na IaC.

Benefícios

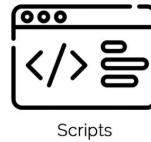
Dentre os muitos benefícios de se usar a IaC, podemos citar:

- Redução de custos
- Aumento na velocidade das implantações
- Redução de erros
- Melhoria na consistência da infraestrutura
- Eliminação de desvios de configuração

IAC WORKFLOW



Templates



Scripts



Policies



ANSIBLE



Terraform



CHEF



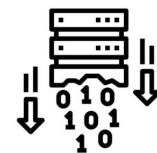
puppet



Network



Application



Storage

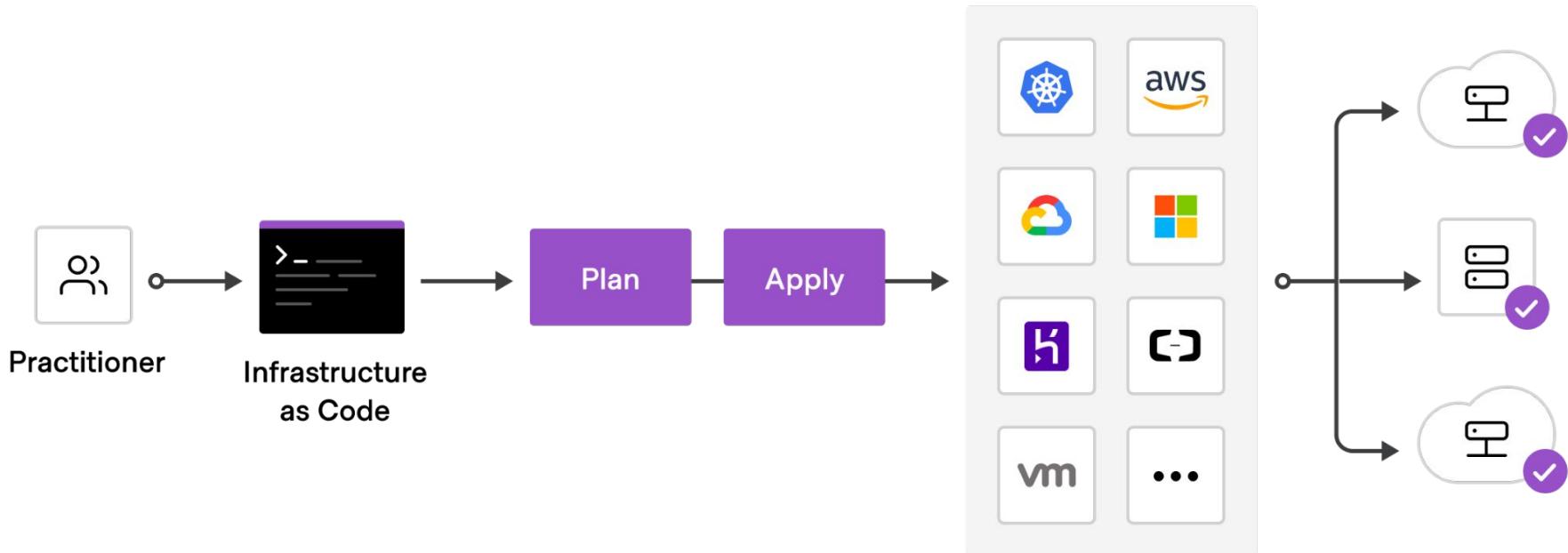


Security



Cloud
Infrastructure

INFRAESTRUTURA COMO CÓDIGO

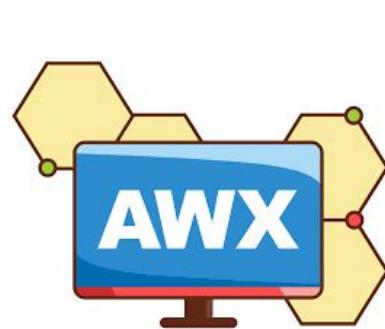


Gestão de configuração

A gestão de configuração é um processo de manutenção que certifica de que o sistema funciona como o esperado enquanto as mudanças são feitas.

Gerenciar configurações de sistemas de TI envolve a definição de um estado desejado, como a configuração do servidor, e então a criação e manutenção desses sistemas. Intimamente relacionado às avaliações da configuração e análise de desvios, o gerenciamento de configuração as utiliza para identificar os sistemas que serão atualizados ou reconfigurados ou que receberão a aplicação de patches.

FERRAMENTAS DE GESTÃO



Jenkins



Monitoramento e Observabilidade

Monitoramento e Observabilidade

A **Observabilidade** é a prática de instrumentalizar sistemas com ferramentas ou código, com o objetivo de coletar dados que esclareçam não só quando um erro ou incidente ocorreu, mas o porquê.

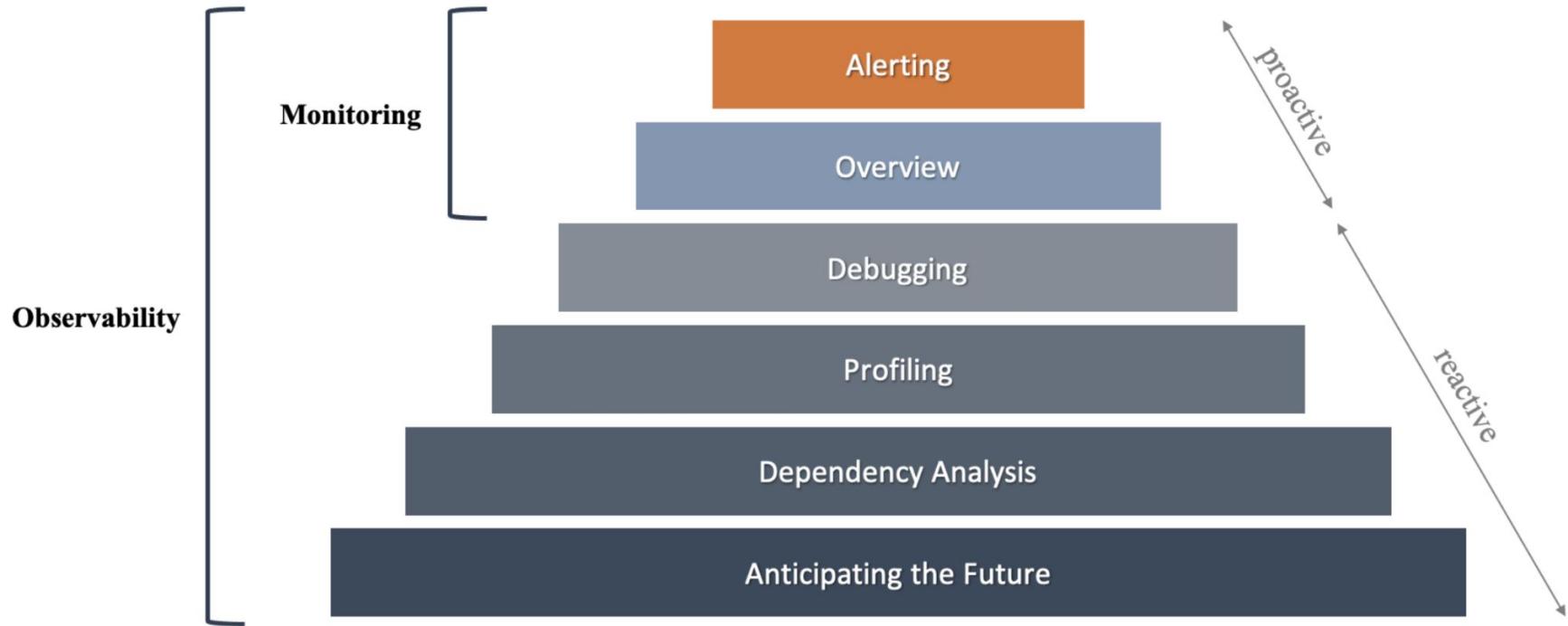
O **monitoramento** de um sistema, por sua vez, é a capacidade de inferir sua condição interna, indicando a qualidade do desempenho do sistema ao longo de um período. Ou seja, o monitoramento considera o estado do sistema como um todo. Cada serviço individual produz dados que alimentam essa visualização agregada, gerando alertas.

Monitoramento e Observabilidade

Embora o termo “**monitoramento**” às vezes seja definido como diferente da observabilidade, o monitoramento é uma atividade que torna um sistema observável, juntamente com outras atividades, como rastreamento e registro.

Frequentemente, você verá **monitoramento, rastreamento e registro** descritos como “**três pilares de observabilidade**”.

OBSERVABILIDADE E MONITORAMENTO



Logs

Um **log** é um registro imutável com carimbo de data/hora de eventos discretos que pode fornecer um relato abrangente do que aconteceu em um aplicativo em um momento específico. Os logs de eventos podem vir em formato de texto simples, estruturado ou binário, mas todos os formatos contêm uma carga útil.

Métricas

As métricas são a base do monitoramento. Eles são valores que expressam parte do estado interno de um sistema. Normalmente, as métricas são definidas como contagens ou medidas que são agregadas ao longo de um período de tempo.

Um sistema observável deve fornecer medições constantes para métricas predefinidas. As melhores métricas são aquelas que apresentam informações acionáveis sobre seu aplicativo, não necessariamente gráficos genéricos de CPU e memória.

Rastreamento

Um **rastreamento** descreve toda a jornada de uma solicitação conforme ela se move de um nó para outro em um sistema distribuído. Ele revela os efeitos não intencionais da solicitação e fornece visibilidade em sua estrutura. Para uma transação ou solicitação individual, um único rastreio exibe a operação conforme ela passa por um aplicativo.

Os rastreios permitem que você entre em detalhes de solicitações específicas para determinar quais componentes causam erros de sistema; para monitorar o fluxo através dos módulos; e para encontrar gargalos de desempenho.

Alertas

Quando algo dá errado, você precisa receber alertas oportunos. No entanto, a detecção muito sensível pode levar à fadiga do alarme, o que torna o gerenciamento de alertas fundamental.

SLA / SLI / SLO

As três siglas representam as promessas que são feitas aos usuários, os objetivos internos que ajudam a manter as promessas e as medições rastreáveis que dizem como se está indo.

O objetivo dos três é que todos, fornecedores e clientes, estejam alinhados sobre o desempenho do sistema.

SLA

Um **SLA (Service Level Agreement)** é um acordo entre o provedor e o cliente sobre métricas mensuráveis, como disponibilidade, capacidade de resposta e responsabilidades.

SLO

O **SLO (Service Level Objectives)** é um acordo dentro de um SLA sobre uma métrica específica, como disponibilidade ou tempo de resposta. Portanto, se o SLA for o acordo formal entre você e o cliente, os SLOs são as promessas individuais que você faz a esse cliente. Os SLOs são o que definem as expectativas do cliente e dizem às equipes de TI e DevOps quais metas precisam atingir e considerar como referência.

SLI

O **SLI (Service Level Indicators)** mede a conformidade com o SLO. Assim, por exemplo, se o SLA especificar que os sistemas vão estar disponíveis 99,95% do tempo, é provável que o SLO vá ter 99,95% de disponibilidade, e o SLI é a medição real da disponibilidade. Talvez seja 99,96%. Talvez 99,99%. Para se manter em conformidade com o SLA, o SLI vai precisar cumprir ou superar as promessas feitas nesse documento.

SLA / SLO / SLI

Aquisição de link de Internet

SLA	SLO	SLI						
<p>Link de 100Mbps, com tolerância de 10% de perda.</p> <p>Mínimo aceitável de velocidade de download: 90Mps</p> <p>Mínimo aceitável de velocidade de upload: 90Mps</p> <p>Ferramentas de medição: Fast.com ou Speedtest</p>	<p>Avaliação por média diária;</p> <p>Mínimo aceitável de velocidade de download: 90Mps</p> <p>Mínimo aceitável de velocidade de upload: 90Mps</p> <p>Ferramentas de medição: Fast.com ou Speedtest</p>	 <p>The screenshot shows a speed test result from fast.com. The main display shows a large '92 Mbps' with a green circular icon. Below it, the results are broken down into Latency (2 ms), Download (17 ms), and Upload (80 Mbps). At the bottom, there are sharing options (Compartilhar) and configuration links (Configurações, Resultados, Configurações).</p> <table border="1"> <thead> <tr> <th>PING ms</th> <th>DOWNLOAD Mbps</th> <th>UPLOAD Mbps</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>93.68</td> <td>93.79</td> </tr> </tbody> </table>	PING ms	DOWNLOAD Mbps	UPLOAD Mbps	1	93.68	93.79
PING ms	DOWNLOAD Mbps	UPLOAD Mbps						
1	93.68	93.79						

SLI / SLO / SLA

Product



SLIs

SLOs

SLAs

Tipos de monitoramento

- Tempo de atividade / monitoramento de disponibilidade;
- Monitoramento de desempenho do site
- Monitoramento de recursos
- Monitoramento de erros
- Monitoramento de log
- Monitoramento de banco de dados
- Monitoramento de segurança ou malware.

FERRAMENTAS

Gerenciamento e centralização de logs:

- Elastic Stack;
- Splunk;
- Graylog.

Monitoramento de infraestrutura:

- Grafana;
- Zabbix;
- Nagios;
- Prometheus.

Monitoramento de performance de aplicação:

- New Relic;
- Data Dog;
- App Dynamics;
- Elastic APM.



Práticas de Segurança

Práticas de segurança

Pensar em segurança é um dos pontos principais em qualquer desenvolvimento de software, bem como da Cultura DevOps, existe até mesmo um termo no mercado para isto DevSecOps. Dito isto, vamos listar algumas práticas importantes que devem ser validadas e fazendo sentido incorporadas ao processo de desenvolvimento.

- Padronize e automatize o ambiente.
- Cada serviço deve ter o mínimo possível de privilégios para reduzir as conexões e os acessos não autorizados.
- Centralize os recursos de controle de acesso e identidade de usuários.
- Criptografe os dados trocados entre aplicativos e serviços.

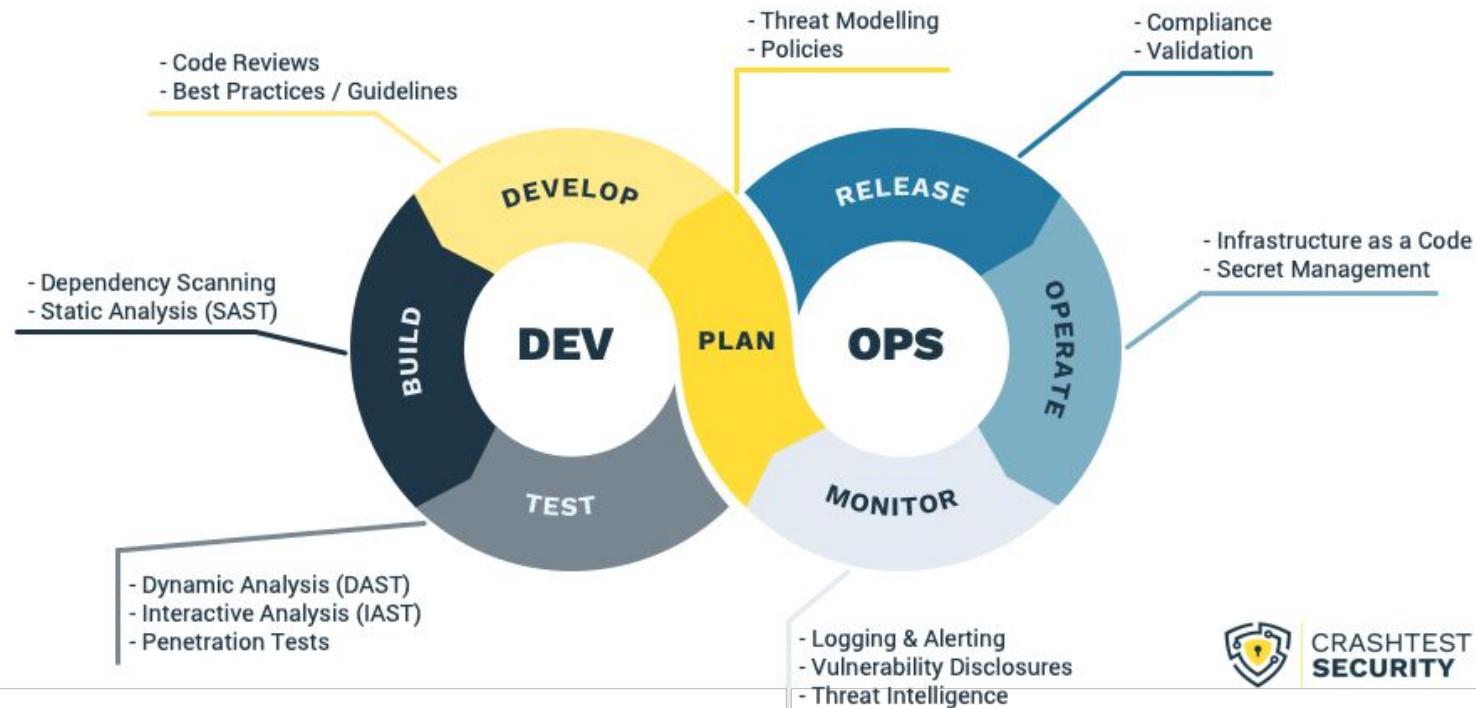
Práticas de segurança

- Automatize os testes de segurança no processo de integração contínua.
- Adicione testes automatizados para os recursos de segurança no processo de teste de aceitação.
- Automatize as atualizações de segurança, como patches, para identificar vulnerabilidades conhecidas.
- Automatize os recursos de gerenciamento das configurações de serviços e sistemas.

Análise de código e infraestrutura

- Categorização e priorização de planos de remediação
- Acompanhamento contínuo dos planos de remediação para melhoria de segurança
- Recomendação/implementação de alternativas para avaliação de segurança nas ferramentas de integração contínua
- Correção de vulnerabilidades
- Testes de segurança especializados
- Avaliações de vulnerabilidades em diferentes etapas do ciclo de desenvolvimento
- Validação de pacotes entregues por fábricas ou equipes de desenvolvimento

FLUXO DEVSECOPS



SAST VS DAST

SAST

- White box testing
- Requires Source Code
- Earlier detection
- Doesn't find environment issues
- Supports all software

VS

DAST

- Black box testing
- Requires Web Application in staging or production
- Later detection
- Finds environment issues
- Predominantly Web App testing.



FERRAMENTAS





C . E . S . A . R

Pessoas impulsionando inovação.
Inovação impulsionando negócios.

NOSSO CONTATO

<http://cesar.org.br>
negocios@cesar.org.br
+55 81 3425-4700

