Microsserviços e APIs

Igor Kalb

Sumário

- Quem sou;
- Agenda;
- Introdução ao desenvolvimento de software e metodologias agéis;
- O que são microsserviços?
- Arquitetura de microsserviços;
- Desafios de uma arquitetura distribuída;
- Como quebramos e definimos um microsserviço;
- Dinâmica;
- O que é uma API?
- Padrões de Comunicação entre serviços (REST, message, gRPC);
- Dinamica;

Quem sou?

Quem sou

- Formado em Análise e Desenvolvimento de sistemas pelo IFRS - Sertão em 2019;
- Especialista em Arquitetura de Software distribuído pela PUC - Minas;
- MBA em Cloud Computing pelo IGTI;
- Certificado pela Oracle, Azure, DevOps Institute e EXIN;
- Já atuei como Desenvolvedor backend, DevOps Engineer e Engenheiro de Software;
- Hoje atuo como Arquiteto de Solução pela Livelo

Agenda

Agenda

- Dia 1
  - Introdução ao desenvolvimento de software e metodologias agéis;
  - O que são microsserviços?
  - Arquitetura de microsserviços
  - Desafios de uma arquitetura distribuída;
  - Como quebramos e definimos um microsserviço;
  - Dinâmica;
  - O que é uma API?
  - Padrões de Comunicação entre serviços (REST, message, gRPC);
  - Resiliência de serviços
  - Dinamica;
  - Documentação de APIs

Agenda

- Dia 2
  - O que é DevOps?
  - DevOps e a arquitetura de microsserviços;
  - Entrega continua, integração continua e implantação continua;
  - Pipelines - exemplos e dinâmica;
  - Qualidade de uma arquitetura distribuída;
  - SRE;
  - Monitoria e Observabilidade;
  - Ferramentas para observabilidade;
  - Dinâmica;

Agenda

- Dia 3
  - Segurança de aplicações;
  - API Gateway;
  - Comunicação front-end x back-end;
  - Cloud e microsserviços - cloud native;
  - Containers e docker;
  - Kubernetes;
  - Prática, criando uma API Simples

Agenda

- Dia 4
  - Continuação da criação da API

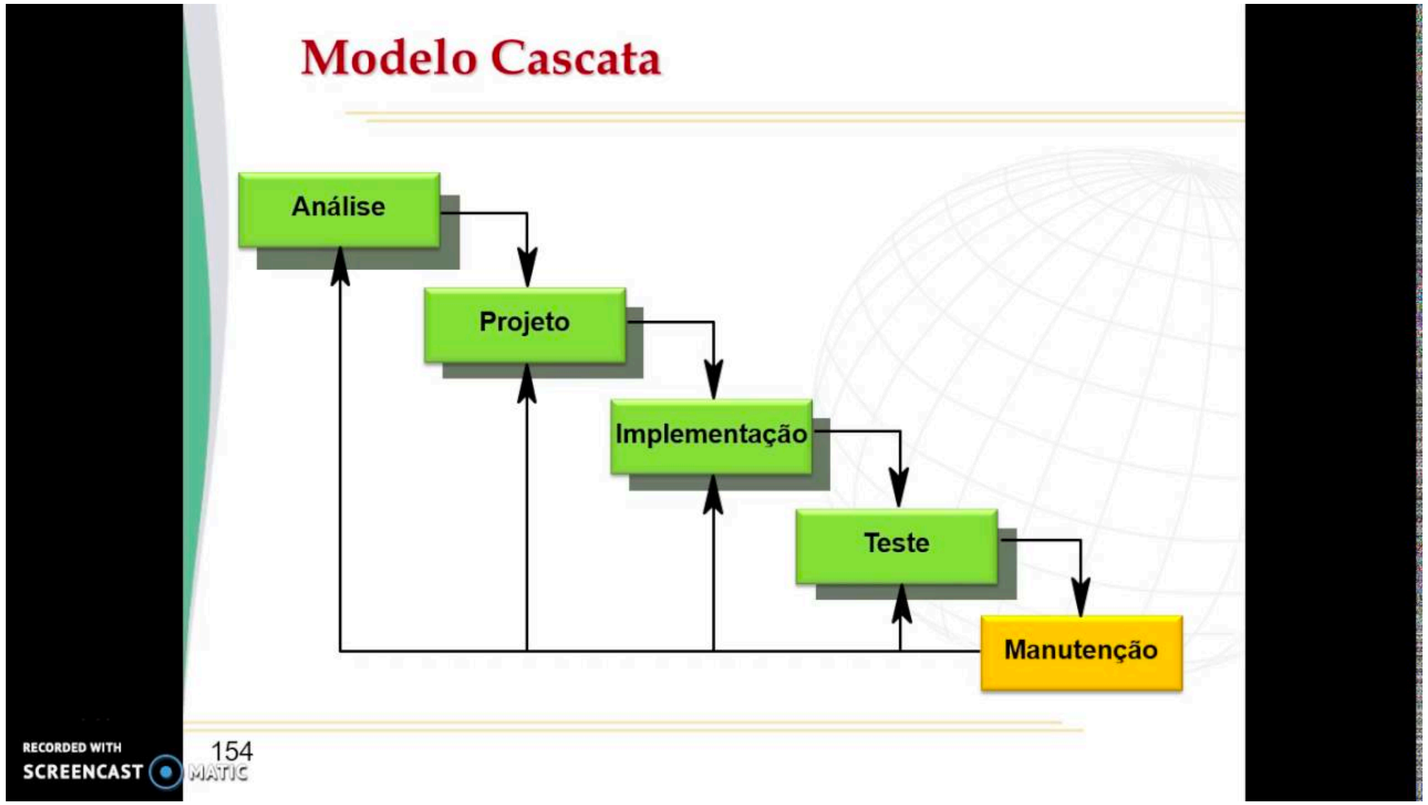
Dia 1

Introdução ao desenvolvimento de software e Metodologia Ágil

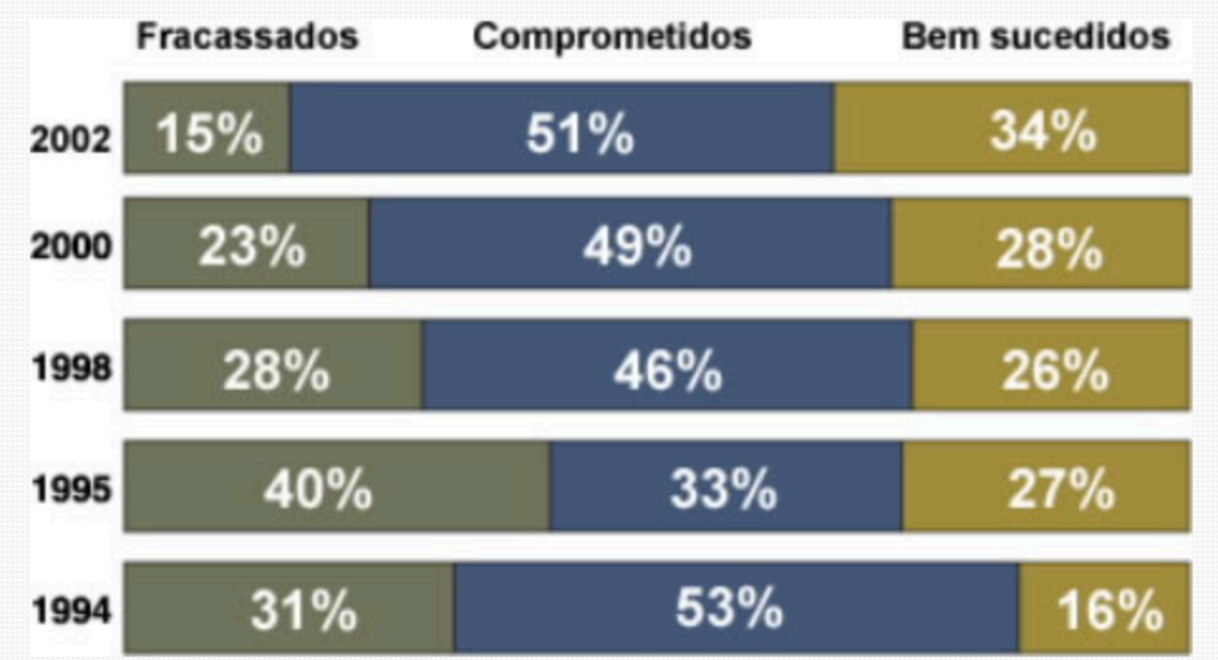
Introdução ao desenvolvimento de software

- A metodologia tradicional ou modelo cascata, o desenvolvimento de software era feito seguindo etapas;
- Cada etapa precisava ser concluida antes de ser iniciada a próxima;
- Possui uma falha bem grande no desenvolvimento de software que é a incapacidade de lidar com mudanças ou lidar com imprevistos e a falta de qualidade ao longo do projeto;
- Por essa falta de qualidade, os softwares não suportavam longo períodos rodando e acabavam fracassando.

Introdução ao desenvolvimento de software



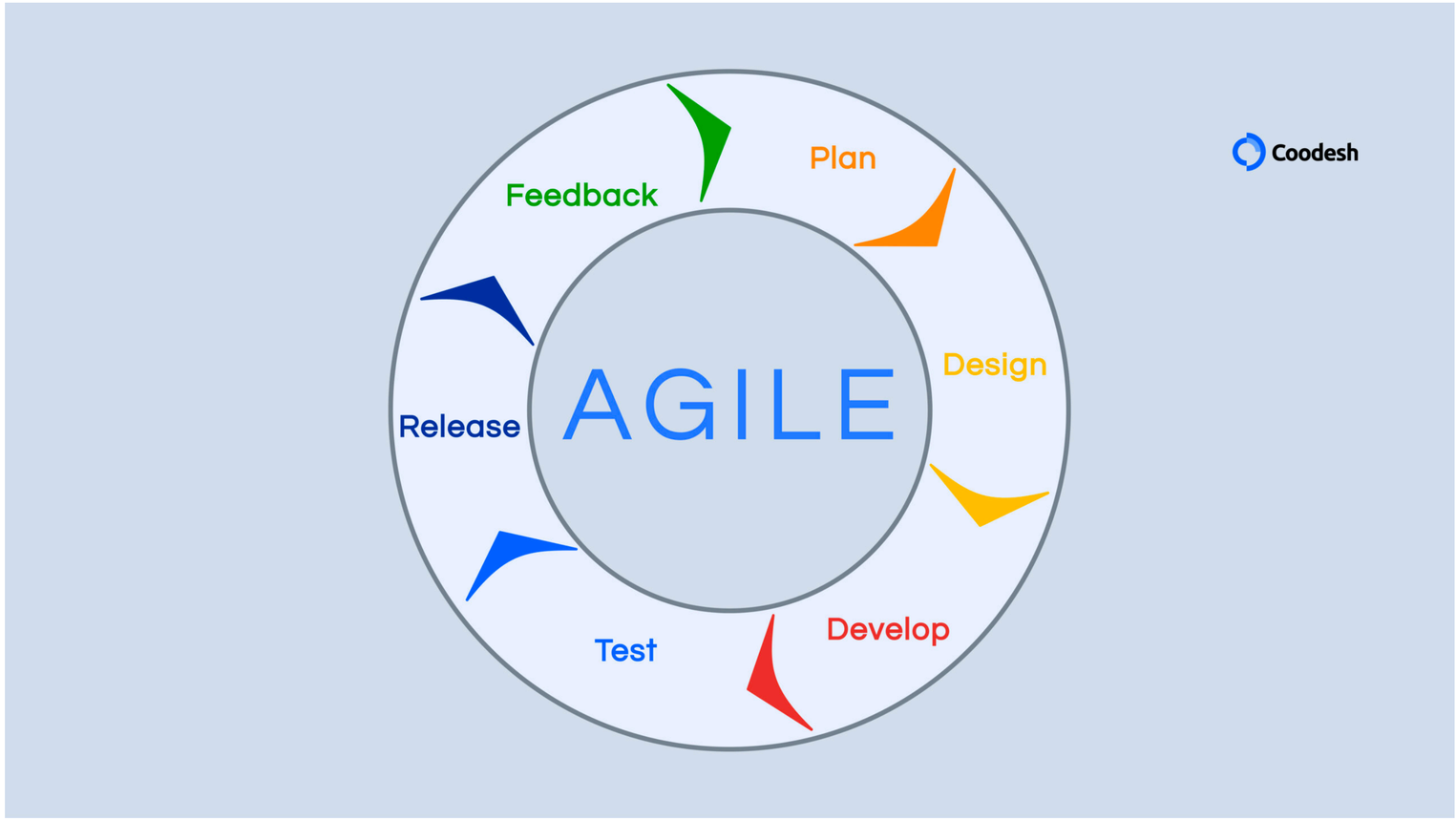
Introdução ao desenvolvimento de software



Metodologia ágil

- A metodologia Ágil veio para solucionar o problema da falta de reatividade a mudanças do negócio e a qualidade do software entregue;
- Escrita do manuscrito ágil em 2001;
- Obedece 4 principios básicos:
  - Indivíduos e interação entre eles mais que processos e ferramentas;
  - Software em funcionamento mais que documentação abrangente;
  - Colaboração com o cliente mais que negociação de contratos;
  - Responder a mudanças mais que seguir um plano;

Metodologia ágil



O que são microsserviços?

Monolito

- Grande peça de software;
- Composto por Front-end e back-end em uma única aplicação;
  - Arquiteturas MVC (Model-View-Controller);
- Aplicação que contem toda a regra de negocio do sistema;
- Problemas:
  - Um código errado e todo o negócio está fora do ar;
  - Consumo alto de memoria e CPU;
  - Uma única base de código, concorrência entre times;
  - Tempo alto para entrega de algo para o cliente;

Microsserviços

Em poucas palavras uma arquitetura de microsserviço é a construção de uma aplicação através de pequenos serviços, independentes de linguagem e que se comunicam entre si através de APIs, utilizando protocolos como HTTP, AMQP, gRPC etc.

Características Gerais

- Aplicações altamente coesas e de baixo acoplamento;
- Independente de linguagem (podem ser criados serviços usando JavaScript, GO, Java, Kotlin, Python etc)
- Facilidade na entrega do software (CI/CD)
- Maior mantenabilidade, escalabilidade e evolução do software;
- Dividido por capacidade/contexto de negocio (normalmente);

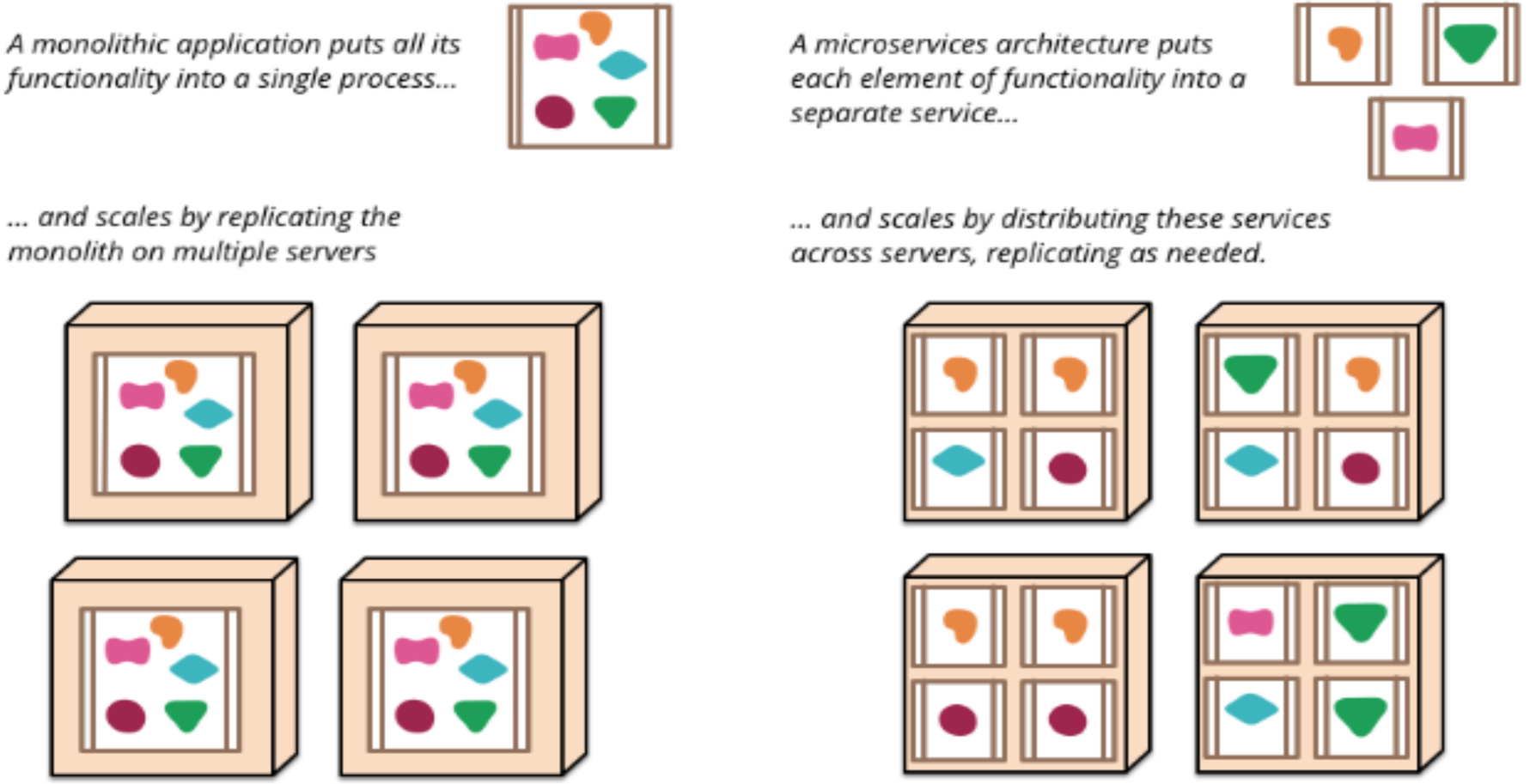


Figure 1: Monoliths and Microservices

Desafios de uma arquitetura distribuida

- Sistemas distribuídos são difíceis demais de lidar (chamadas entre serviços são lentas e podem falhar);
    - Dados estão espalhados em diferentes partes;
- Um serviço pode comprometer o sistema todo;
- Consistência eventual (Teorema CAP);
- Complexidade de operações (CI/CD tem que ser bem maduro para garantir deploy/saúde de diversos serviços)

Consistency & Availability
- MySQL
- PostgreSQL

Consistency & Partition tolerance
- HBase
- MongoDB
- Redis
- Memcache

Availability & Partition tolerance
- Riak
- Cassandra
- CouchDB
- DynamoDB

Como definimos um microsserviço?

- Primeiro precisamos fazer o levantamento de todos os requisitos funcionais, não funcionais e restrições arquiteturais;
- Feito isso, podemos agora separar os requisitos em modelos de negocio ou contextos específicos.
- Definindo o que cada microsserviço vai ser responsável, podemos olhar para os requisitos não funcionais e definir linguagem, arquitetura interna e etc.

Dinâmica

Estudo de caso

O seu Joaquim quer tornar a sua loja física em um e-commerce, ele contrata a turma aqui presente para o desenvolvimento dessa plataforma.
Ele precisaria que o sistema fosse capaz de realizar vendas online, calculo de frete, gestão de estoque, tracking de pedidos e todas as outras funcionalidades que um e-commerce possui.
O budget do projeto é ilimitado, já que o seu Joaquim ganhou na Mega-Sena, o único desejo dele para o projeto é que a construção do software seja feita in-house e que seja escalável, confiável e seguro para os usuários.

O que é uma API?

Definição

Uma API é um conjunto de definições e protocolos usado no desenvolvimento e na integração de aplicações. Às vezes, as APIs são descritas como um contrato entre um provedor e um usuário de informações, estabelecendo o conteúdo exigido pelo consumidor (a chamada) e o conteúdo exigido pelo produtor (a resposta).

Fonte: https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api

Formas de comunicação entre APIs

- Utilizando HTTP;
  - APIs construídas dentro do padrão REST;
- Por mensagem, utilizando o padrão AMQP;
- gRPC;
  - Google Remote Procedure Call;

REST

- REST é Representational State Transfer que, em português, é **“Transferência de Estado Representacional”**. Por sua definição, trata-se de um conjunto de princípios e definições necessário para a criação de um projeto com interfaces bem definidas.
- Possui vários formatos aceitos, por exemplo:
  - JSON (é o mais usado hoje em dia);
  - HTML;
  - XLT;

Fonte: https://coodesh.com/blog/dicionario/o-que-e-rest/

REST

- Possui diversos verbos para acessar os recursos, exemplo:
  - POST (Utilizado para criação de dados)
  - PUT (Utilizado para atualização total)
  - DELETE
  - PATCH (Utilizado para atualização parcial)
  - GET (Utilizado para buscar informações)

Responses de uma API REST

- Aplicações REST possuem código que são devolvidos indicando sucesso, erro, redirect, não autorizado e etc.
- Por exemplo:
  - Códigos 2xx - Indicam normalmente sucesso na requisição;
  - Códigos 3xx - Indicam redirecionamento;
  - Códigos 4xx - Indicam erro no client;
  - Códigos 5xx - Indicam erro no servidor

REST

- Para uma API ser considerada RESTFul, ela precisa obedecer os seguintes padrões:
  - Ter uma arquitetura cliente/servidor, formada por clientes, servidores e recursos, com solicitações gerenciadas por HTTP;
  - Não manter estado entre requisições;
  - Armazenar dados em cache para otimizar as interações entre cliente e servidor;

REST

- Exemplo de API REST:
  - https://developer.spotify.com/console/

REST

- Como definimos uma API utilizando o padrão REST?
  - Primeiro, criamos um versionamento para a mesma, o versionamento é utilizado da seguinte forma: /v1, /v2, /v3 …
  - Depois definimos o recurso que queremos acessar;
  - Após isso, definimos o que queremos fazer com esse recurso para assim, escolher o verbo HTTP correto (Exemplo: vamos retornar dados? Vamos atualizar algum recurso? Vamos criar um recurso?);

Dinamica REST

- Dinamica:
  - Crie 3 APIs para os microsserviços criados anteriormente.

AMQP

- Diferente do protocolo HTTP, o protocolo AMQP serve para mensagens;
- Ele trabalha como um processo async e menos custoso comparado ao HTTP;
- Possui padrões específicos como: filas, rotas, exchanges e etc;
- Ferramentas que implementam o AMQP:
  - RabbitMQ;
  - ActiveMQ;

Documentação de APIs

- Para APIs utilizando o padrão REST, normalmente é utilizado o a especificação OpenAPI com o Swagger;
- Swagger é uma plataforma que utiliza notação YAML para documentação de APIs, definindo payloads, requests, tokens, responses e etc;
- Para mensagens, existe o AsyncAPI, que utiliza o mesmo padrão do Swagger e a especificação OpenAPI;

Resiliencia de serviços

- Nenhum serviço é 100% confiável, nem mesmos o que nós criamos;
- Serviços podem falhar, por isso é preciso implementar maneiras que essas falhas sejam isoladas e não afetem nosso serviço;
- Lidar com erros em microsserviços é a chave para sucesso dos mesmos;

Resiliencia de serviços - Padrões

- Alguns padrões para manter a resiliência de serviços:
  - Circuit breakers;
  - Fallbacks;
  - Rate-limit;
  - Bulkheads;

Dia 2

O que é DevOps?

Historia do surgimento do DevOps

- O termo foi cunhado em meados de 2008 em uma palestra sobre infraestrutura ágil;
- O DevOps vem com o intuito de integrar os times de desenvolvimento e os times de operação (Administradores de sistemas);
    - Fim do muro das lamentações;
- É uma metodologia, uma cultura, que vem para auxiliar a metodologia ágil;

Pilares do DevOps

- Cultura
  - As equipes precisam ter colaboração, mudança de comportamento, flexibilidade, troca de ideias, manter uma relação saudável entre as areas e principalmente, trabalhar juntos.
- Automação
  - Toda atividade manual e repetitiva precisa e deve ser automatizada, processos manuais tendem a ser demorados e sujeitos a falhas. É necessário tornar o processo mais determinístico e uniforme.
- Medição/Avaliação
  - É importante medir o progresso da abordagem para saber se as equipes estão melhorando ou progredindo;
- Compartilhamento
  - Colaboração entre os times de Devs e Ops, tentando sempre alcançar um processo simplificado que facilita a integração entre eles;

DevOps e a arquitetura de microsserviços

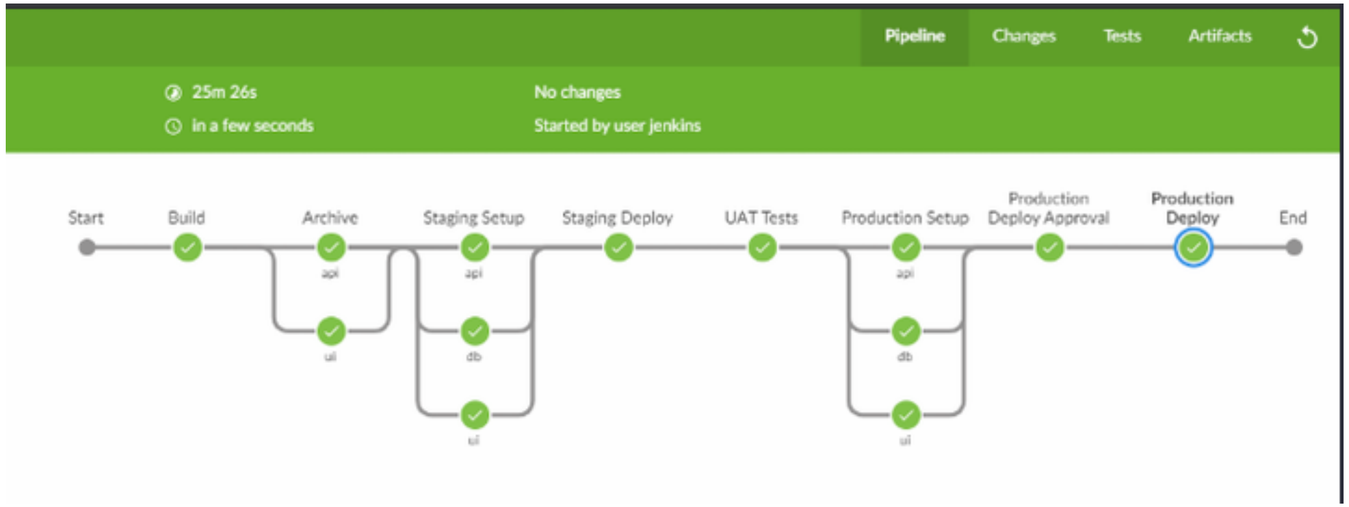
DevOps tem uma sinergia muito grande com microsserviços pelos seguintes motivos:

- Um microsserviço é uma aplicação pequena, isso limita o WIP e acelera a entrega da esquerda para a direita;
- Um microsserviço é um serviço isolado, isso significa que uma parada do sistema não acarreta em todo o sistema fora;
- Microsserviços são escaláveis;
- São mais leves e performáticos do que sistemas monolíticos;

Pipelines

- Para levar um serviço em produção, pegando todo o conceito de DevOps (e as três maneiras que ele descreve), é utilizado um **Pipeline;**
- Um Pipeline é um conjunto de steps que realizam o build, testes, controle de qualidade, segurança e a entrega em ambientes, ou seja, uma automação do processo de entrega do software;

Exemplo de pipelines



Fonte: https://www.rohitsalecha.com/post/practical_devops_continous_delivery_Jenkins/

Qualidade em uma arquitetura distribuida

- Como garantimos qualidade em uma arquitetura que está espalhada em nosso ecossistema?
  - Testes unitários em abundância (investimento maior nesse ponto);
  - Gates de qualidade (SonarQube por exemplo);
  - Ambientes de testes para os desenvolvedores;
  - Testes de contrato (Contract Testing);

Monitoria e Observabilidade

O que é observabilidade e monitoria?

- Observabilidade:
  - Capacidade de medir os estados internos examinando suas saídas. Um sistema só é considerado “observável” se o seu estado atual pode ser estimado apenas pelas suas saídas, também conhecidas como sensores;
- Monitoria:
  - O que você faz depois que um sistema é observável.

Fonte: https://www.splunk.com/en_us/data-insider/what-is-observability.html

Exemplos de observabilidade

- Consumo de CPU e memória da aplicação;
- Regras de negocio especificas:
  - Por exemplo, quantidade de pedidos que desceram em um período X de tempo;
- Quantidade de erros no sistema;
- Quantidade de códigos 2XX, 3XX, 4XX e 5XX de um sistema;
- Se a aplicação está ativa ou não;

Ferramentas para observabilidade

- Datadog;
- Splunk;
- EFK
  - ElasticSearch
  - FluentD
  - Kibana
- Dynatrace;

Dinâmica

- O que podemos observar de nossas aplicações?
  - Façam o levantamento de 3 pontos que precisamos observar em nossas aplicações.

Segurança de aplicações

Segurança de aplicações

- Como garantir a segurança de aplicações distribuídas?
  - Adicionando mecanismos de autenticação e autorização em nossas aplicações (AuthN e AuthZ);
- Existem vários padrões para autenticação de aplicações, umas das mais utilizadas é o JWT;
- JWT garante a autorização e autenticação da pessoa que está tentando acessar nossos serviços;

Segurança de aplicações

- Um exemplo de utilização:
  - Implementando autenticação e autorização utilizando Keycloak;
  - É feito a autenticação pelo keycloak, caso ocorra sucesso, o mesmo irá retornar um token assinado e com as credenciais da pessoa que solicitou;
  - Com esse token, usamos para acessar nosso sistema;

Segurança de aplicações

- Um exemplo prático;

Segurança de aplicações - API Gateway

- Uma API Gateway, é um gateway que fica entre o front e o nosso back-end, sendo assim mais uma camada de segurança de nossos serviços;
- Permite centralizar todas as requisições (já que temos vários endpoints e diversos serviços expostos);
- Permite isolar serviços que não queremos que sejam expostos, e os que estão expostos, possuem uma camada extra de segurança;

Microsserviços e Cloud Computing

Microsserviços e Cloud Computing

- Microsserviços possuem um encaixe perfeito com a computação em nuvem;
  - São leves;
  - Escaláveis;
- Abordagem cloud-native;

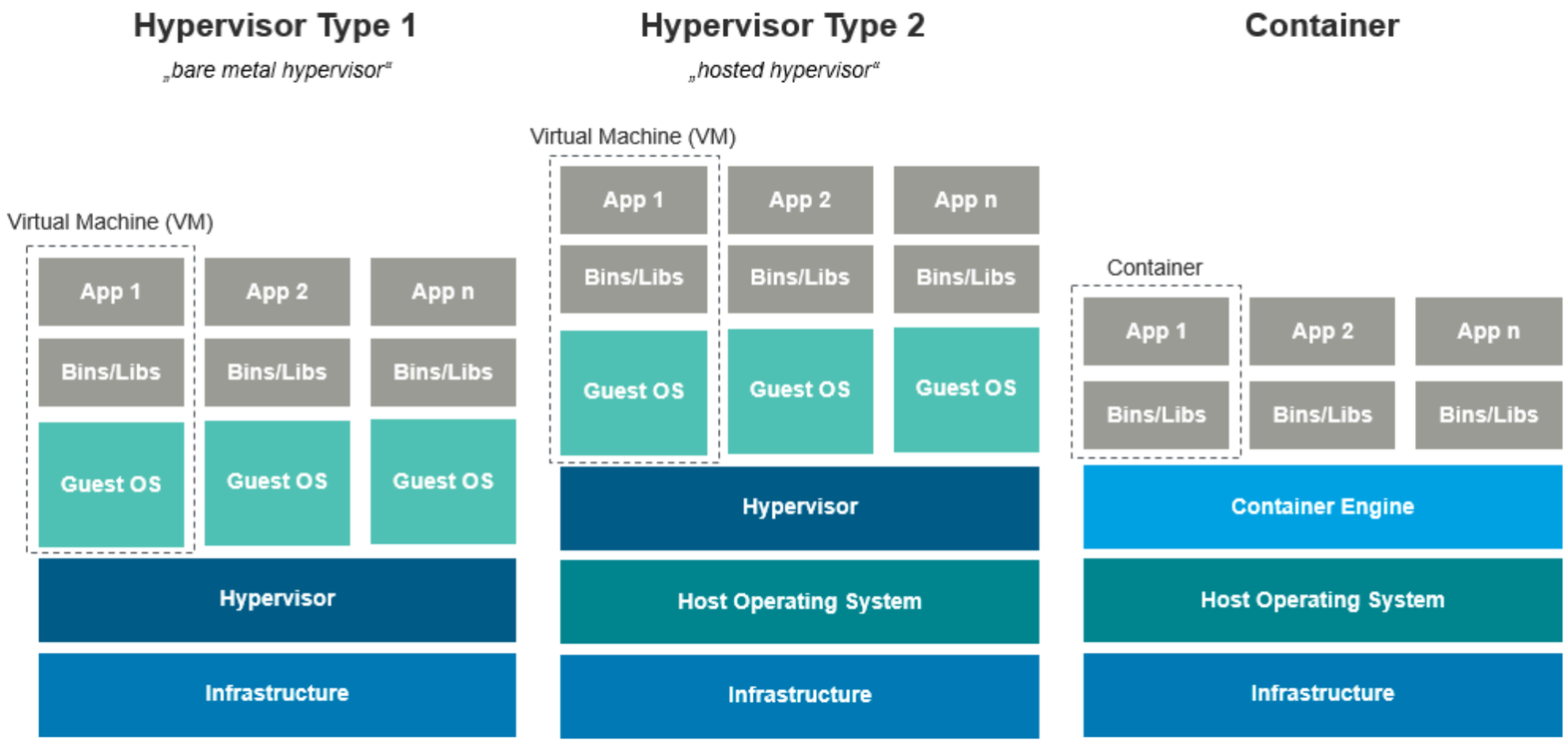
Microsserviços e Cloud Computing

- Porém nem todo microsserviço está apto para ir para a nuvem;
- Isso se deve ao fato de, talvez, o serviço não estar preparado para a nuvem;
  - Serviços mal otimizados;
  - Serviços que não absorvem o melhor da computação em nuvem (escalabilidade, observabilidade e resiliência)
  - Alguns serviços foram construídos voltados para data-centers locais;

Infraestrutura para microsserviços

- Existem várias formas de rodar microsserviços (EC2, VMs, data-center local ou até mesmo clusters Kubernetes);
- Ambas as formas, acabam utilizando o conceito de container para rodar os serviços;
- Docker é uma engine muito popular para rodar containers;

O que é um container?



Mais informações: https://www.docker.com/resources/what-container/

Kubernetes

- Para lidar com tantos containers e fornecer uma forma de resiliência, padronização, escalabilidade e segurança foi criado o Kubernetes;
- Kubernetes foi criado pela Google e doado para a comunidade (hoje ele é mantido pela Linux Foundation);
- Kubernetes se tornou um padrão quando se trata de cloud-computing e microsserviços em larga escala;

Kubernetes

- Kubernetes permite a:
  - Padronização na criação de containers;
  - Orquestração de containers;
  - Comunicação entre serviços dentro e fora do cluster;
  - Configuração de containers (ConfigMaps);
  - Escalabilidade de serviços (HPA ou VPA);
  - Disponibilidade (Garante que o serviço está de pé e o mínimo de pods bate com a descrição do definition-file);
  - Saude de serviços (Realiza a higienização de pods que estão down, criando novos pods);

Dinâmica

- Crie um microsserviço simples;
- Documente pelo menos 1 API do seu serviço usando Swagger;