



MICROSERVIÇOS

Vinicius Soares - Aula 01

Professores

VINICIUS SOARES

Professor Convidado

Entusiasta da Computação Distribuída, Vinicius Soares é Head de Tecnologia em uma das principais empresas do sul do país, ajudando clientes e empresas a alcançarem seus resultados de forma rápida e assertiva. Apaixonado por Java, Arquitetura de Sistemas e Computação em Nuvem, Vinicius possui sólida experiência liderando equipes de Arquitetura usando SOA e Microserviços com tecnologias Open-sources. Compartilha suas experiências através de conteúdo online e eventos nacionais e internacionais como Devoxx, TDC e Campus Party. Como empreendedor, já ajudou mais de 1000 pessoas a se qualificarem para o mercado de TI e atuarem de forma representativa na área.

LUIS FERNANDO PLANELLA GONZALEZ

Professor PUCRS

Doutor Ciências da Computação (PUCRS, 2018). Desenvolvedor e arquiteto Java com experiência profissional desde 1999, certificado pela Sun como programador e desenvolvedor de componentes web na plataforma Java. Entusiasta de software livre.

Ementa da disciplina

Estudo sobre a arquitetura de microserviços. Estudo sobre os conceitos de particionamento de serviços, replicação e distribuição, comunicação assíncrona via filas e Soluções serveless..



Microserviços

Vinicius Soares



Vinicius Soares

Quem sou eu

- Head de tecnologia
- Entusiasta Cloud
- Empreendedor

CONTEXT



O que SOA não é?



- Não é uma tecnologia
- Não é um produto
- Não é um WebService
- Não é um software
- Não é um framework
- Não é uma metodologia
- Não é uma solução de negócio

**SOA é uma abordagem arquitetural corporativa que permite
a criação de serviços de negócio interoperáveis que podem
ser facilmente reutilizados e compartilhados entre aplicações
e empresas**

Visão Geral

- Uma arquitetura baseada em reusabilidade com serviços bem definidos e provados por componentes de TI;
- Seus componentes possuem baixo acoplamentos;
- Provê plataforma, tecnologia e linguagens independentes;

Visão Geral



Benefícios

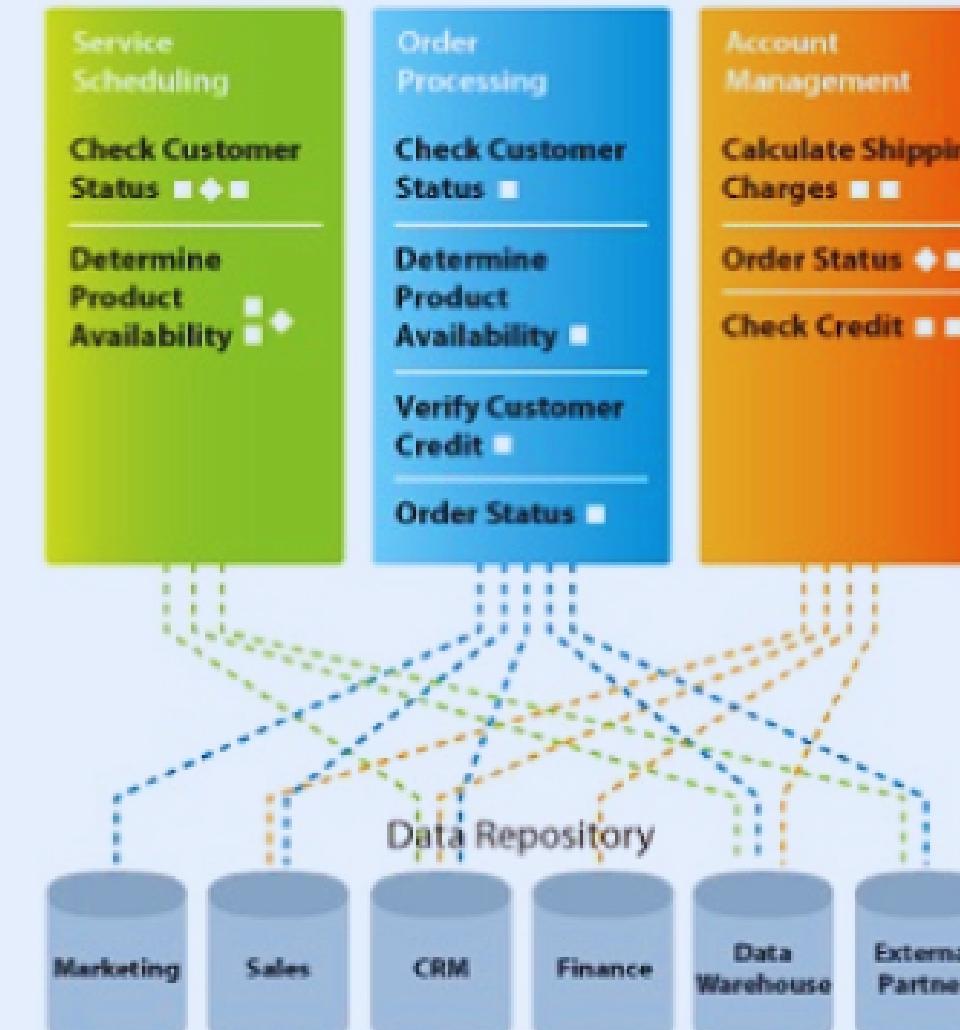


- Desacoplamento:
 - Interações inteligentes, flexibilidade, alinhamento com negócio;
- Reutilização de serviços:
 - Produtividade, manutenibilidade;
- Infraestrutura de plataforma:
 - Padronização corporativo (Log, Governança, etc.);

Before SOA

Closed - Monolithic - Brittle

Application Dependent Business Functions



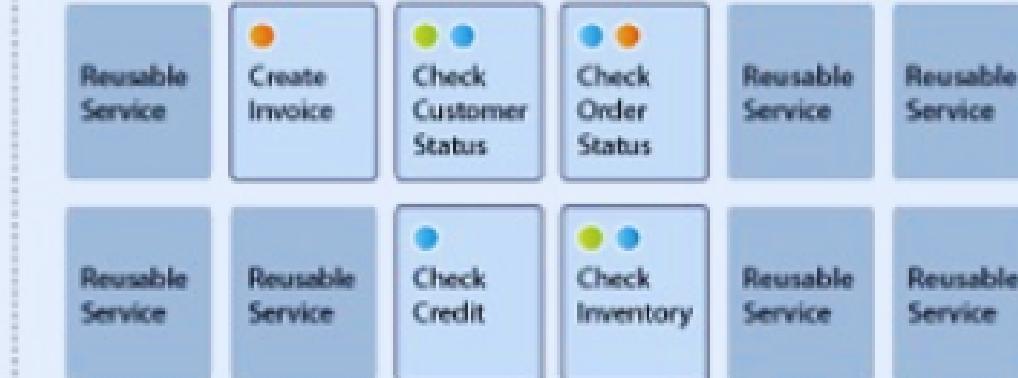
After SOA

Shared services - Collaborative - Interoperable - Integrated

Composite Applications



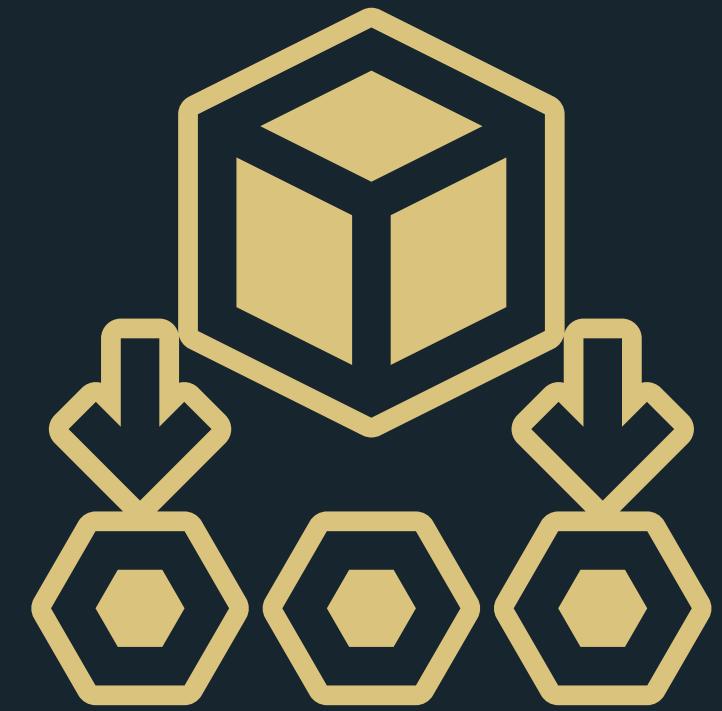
Reusable Business Services



Data Repository



MONOLITO X MICROSERVICES





MONOLITO

Uma arquitetura monolítica típica de um sistema complexo pode ser representada pela figura abaixo, onde todas as funções do negócio estão

IMPLEMENTADAS EM UM UNICO PROCESSO



Vantagens

- Fácil de desenvolver;
- Fácil manutenção;
- Apenas um deploy;
- Tráfego de rede baixo;



Problemas

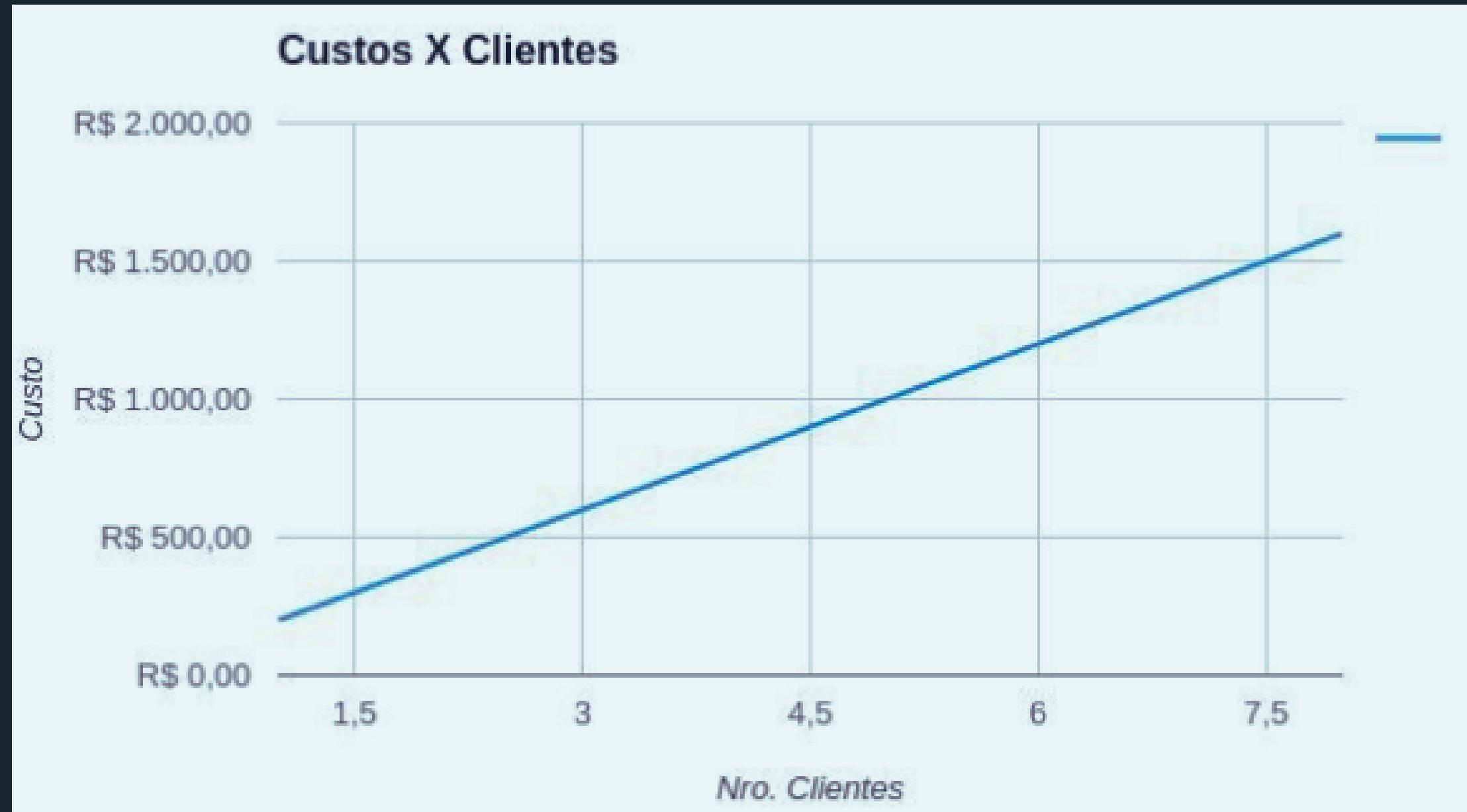
- Aumento de complexidade e tamanho ao longo do tempo;
- Alta dependência de componentes de código;
- Escalabilidade do sistema é limitada;
- Falta de flexibilidade;
- Dificuldade para colocar alterações em produção;



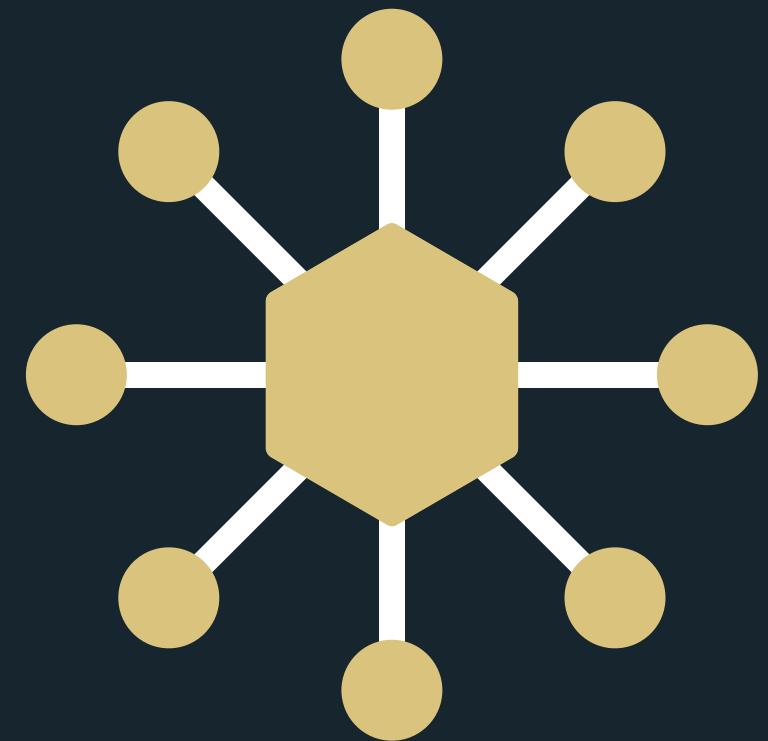
O monolito tem vantagens e desvantagens

Uma das grandes desvantagens é que é muito difícil **ESCALAR** o sistema. Muito comum que cada cliente que você tiver, vai ter que replicar/duplicar o servidor, isso duplicando o custo.

Quanto mais clientes tiver, mais servidor vai ser preciso, mais caro vai ficar.

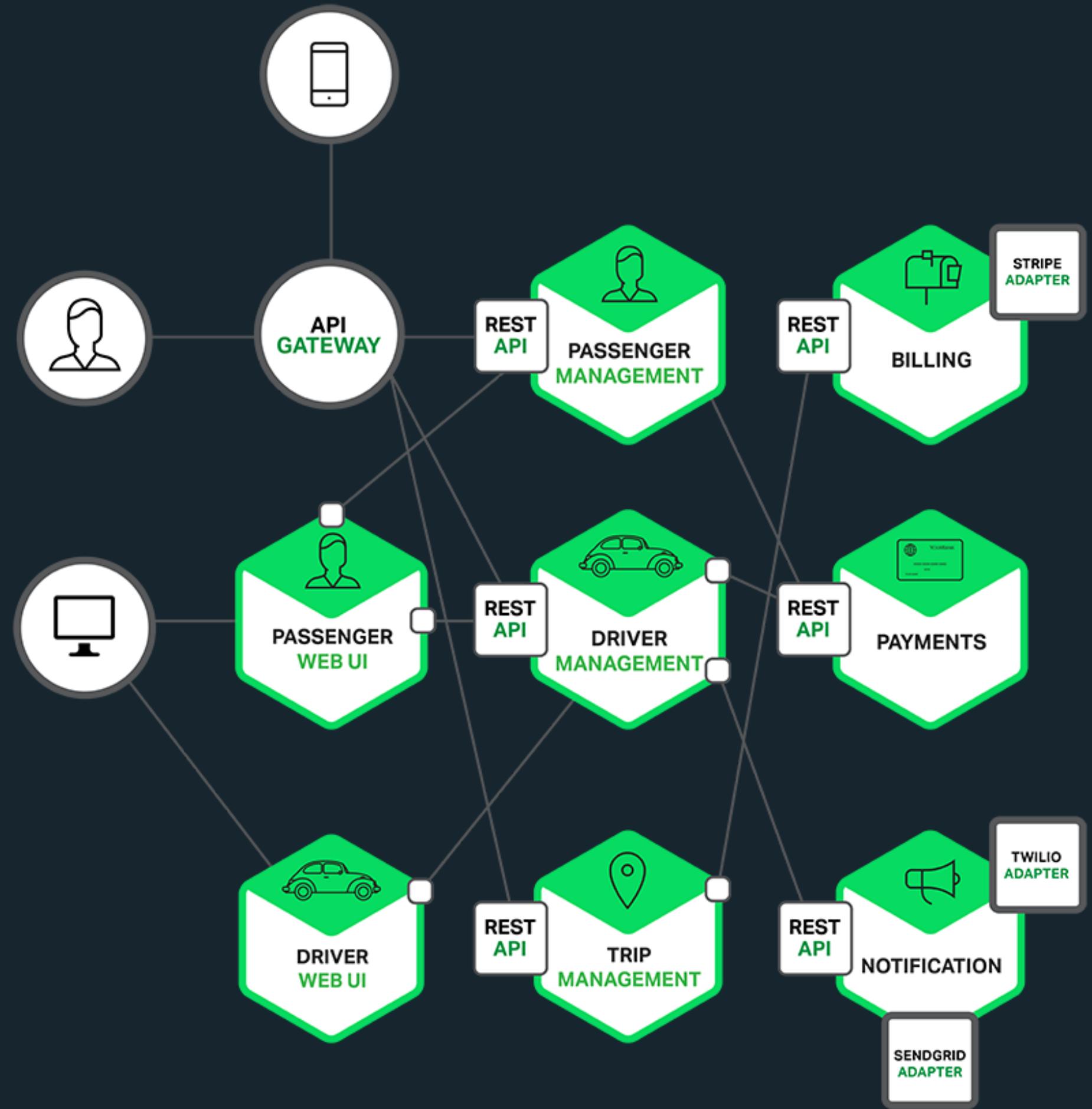


MICROSSERVIÇO



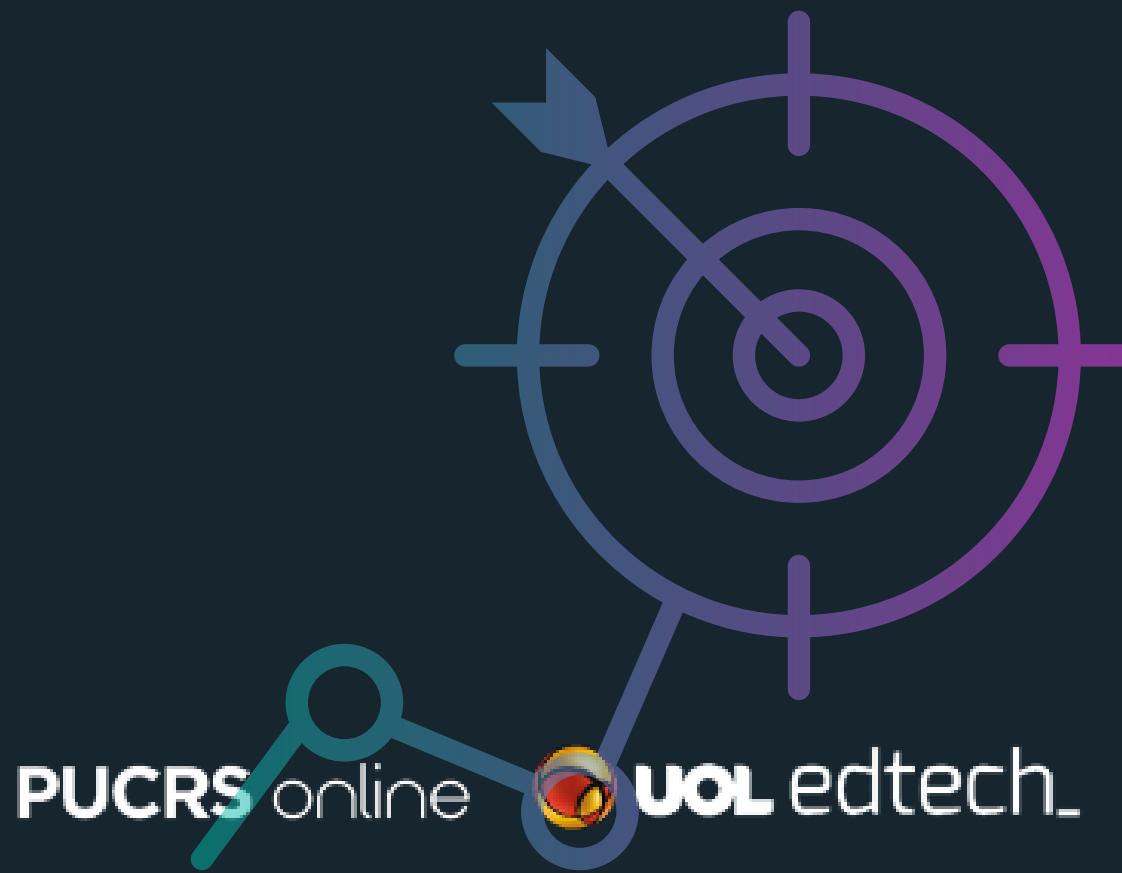
“um serviço com um único propósito e que execute bem a sua tarefa dentro de um nível de granularidade e suporte as mudanças do sistemas que são consideradas importantes tanto em tempo de projeto quanto em tempo de execução. O foco principal é tentar construir software que pode se adaptar e isto só é possível de ser feito se as partes forem pequenas suficientes para se ajustar às diferenças nas mudanças de sua arquitetura.”

RUSS MILE



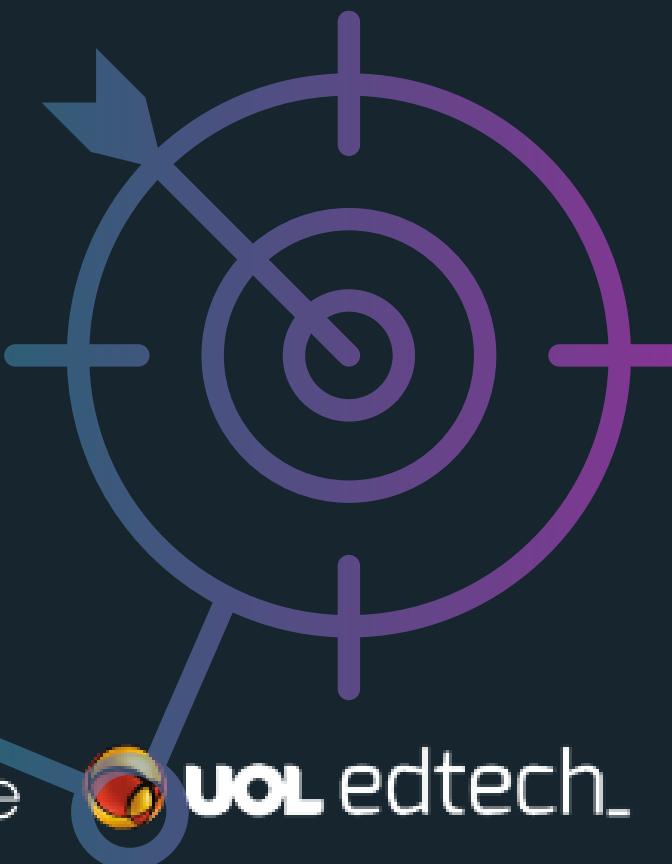
Vantagens

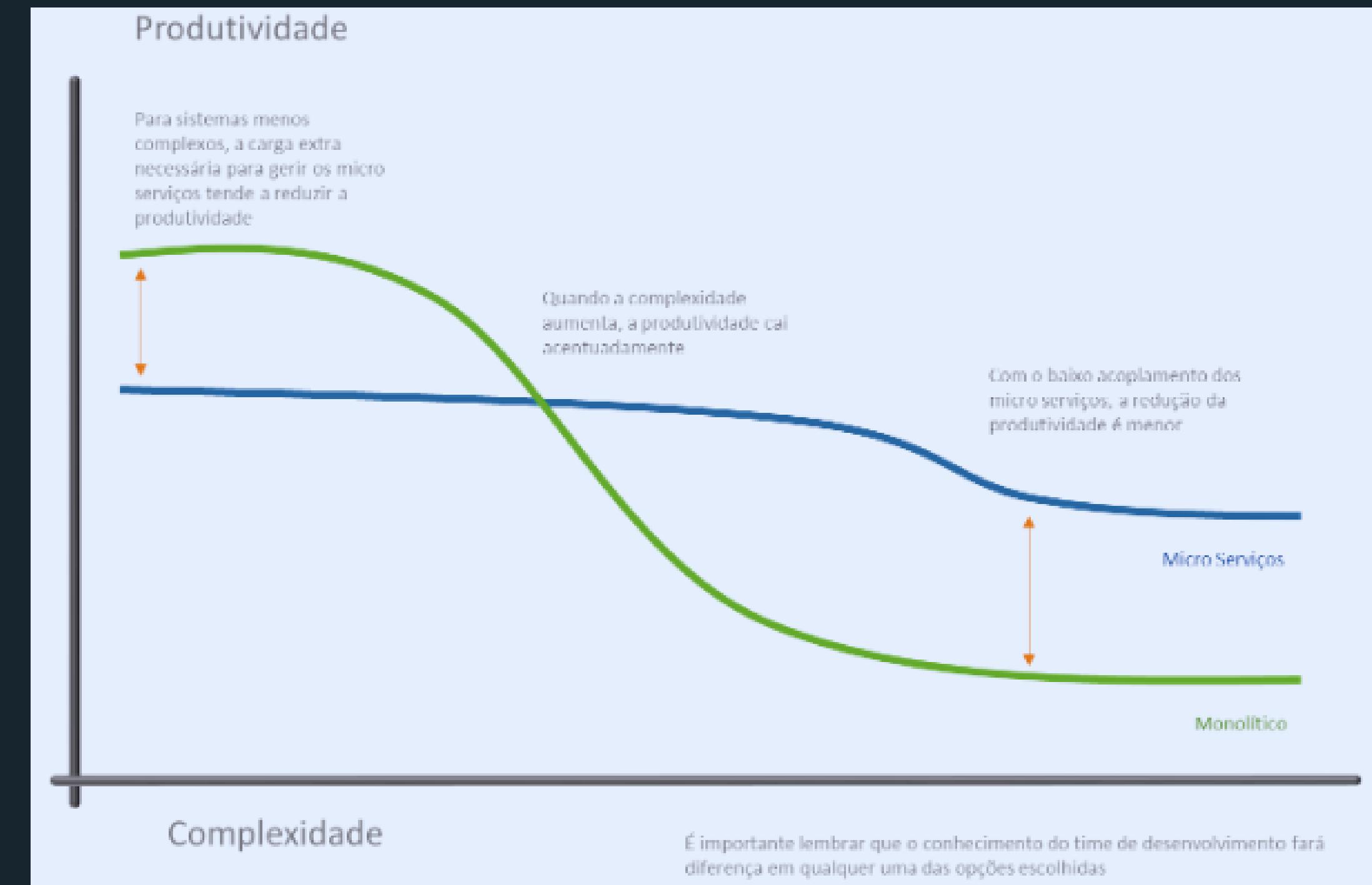
- Manutenção e evolução dos serviços mais estáveis;
- Serviços com baixo nível de acoplamento e interdependência;
- Escalabilidade do sistema;
- Redução de custos;
- Flexibilidade de tecnologia;
- Facilidade de colocar alterações em produção;
- Resiliencia;



Vantagens

- Aumento da produtividade;
- Implementação de entrega contínua;
- Monitoramento e automação de processos;
- Foco na entrega de valor ao cliente;

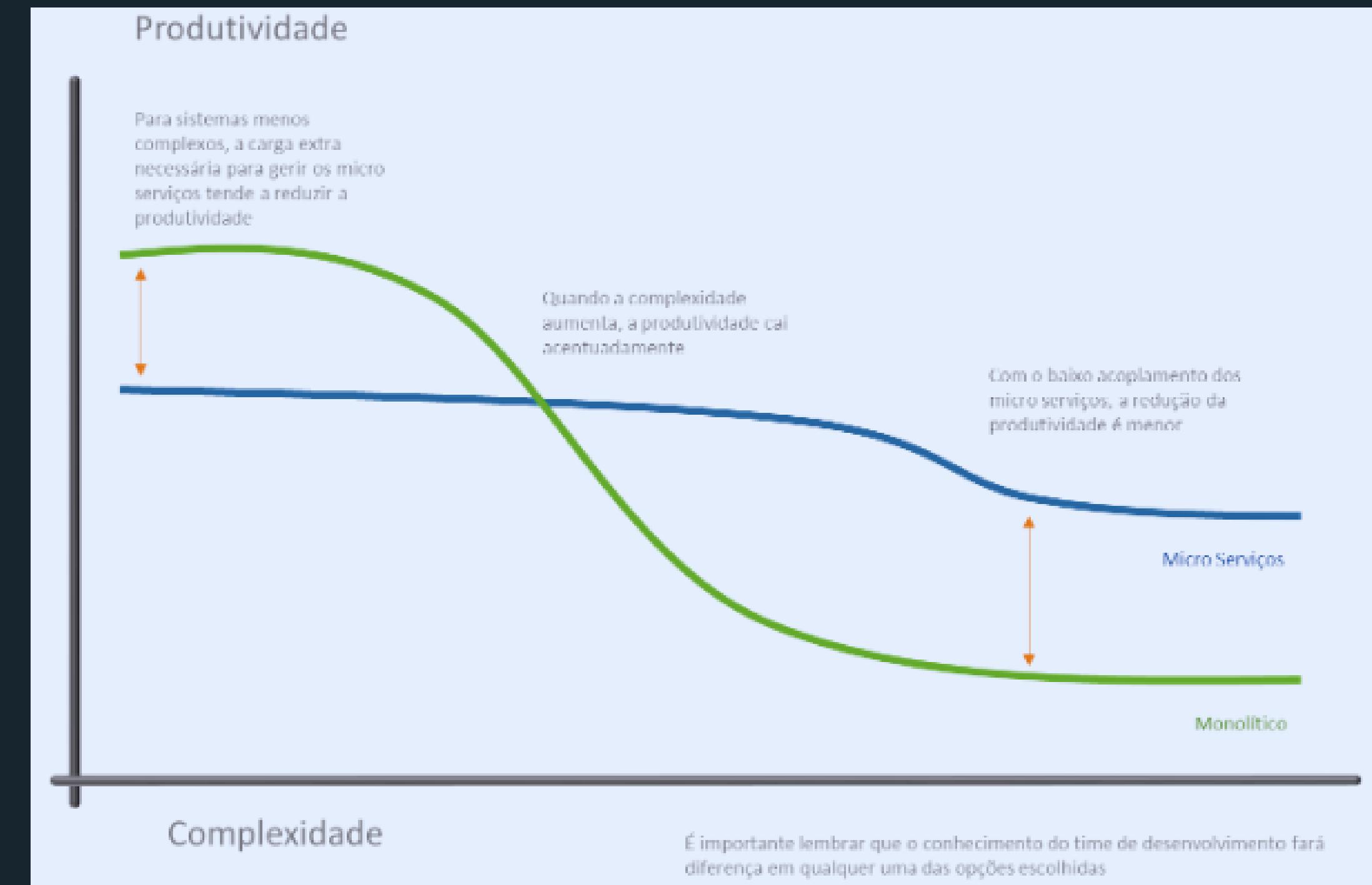




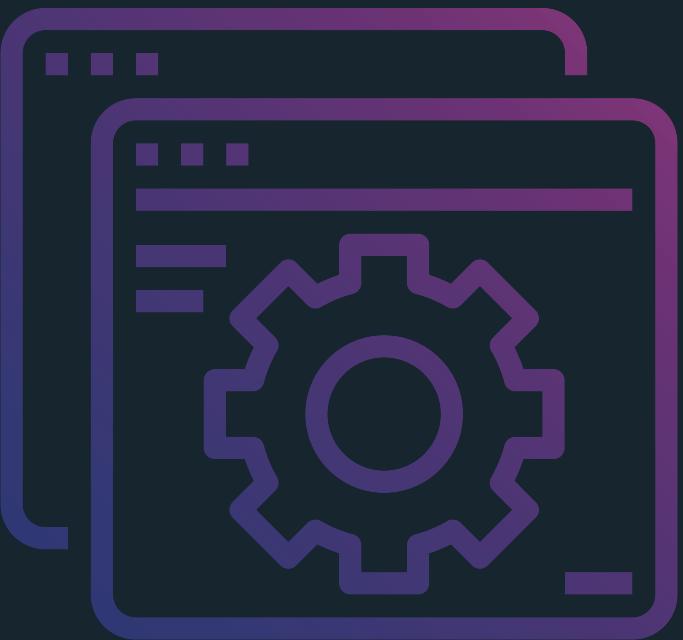
Riscos

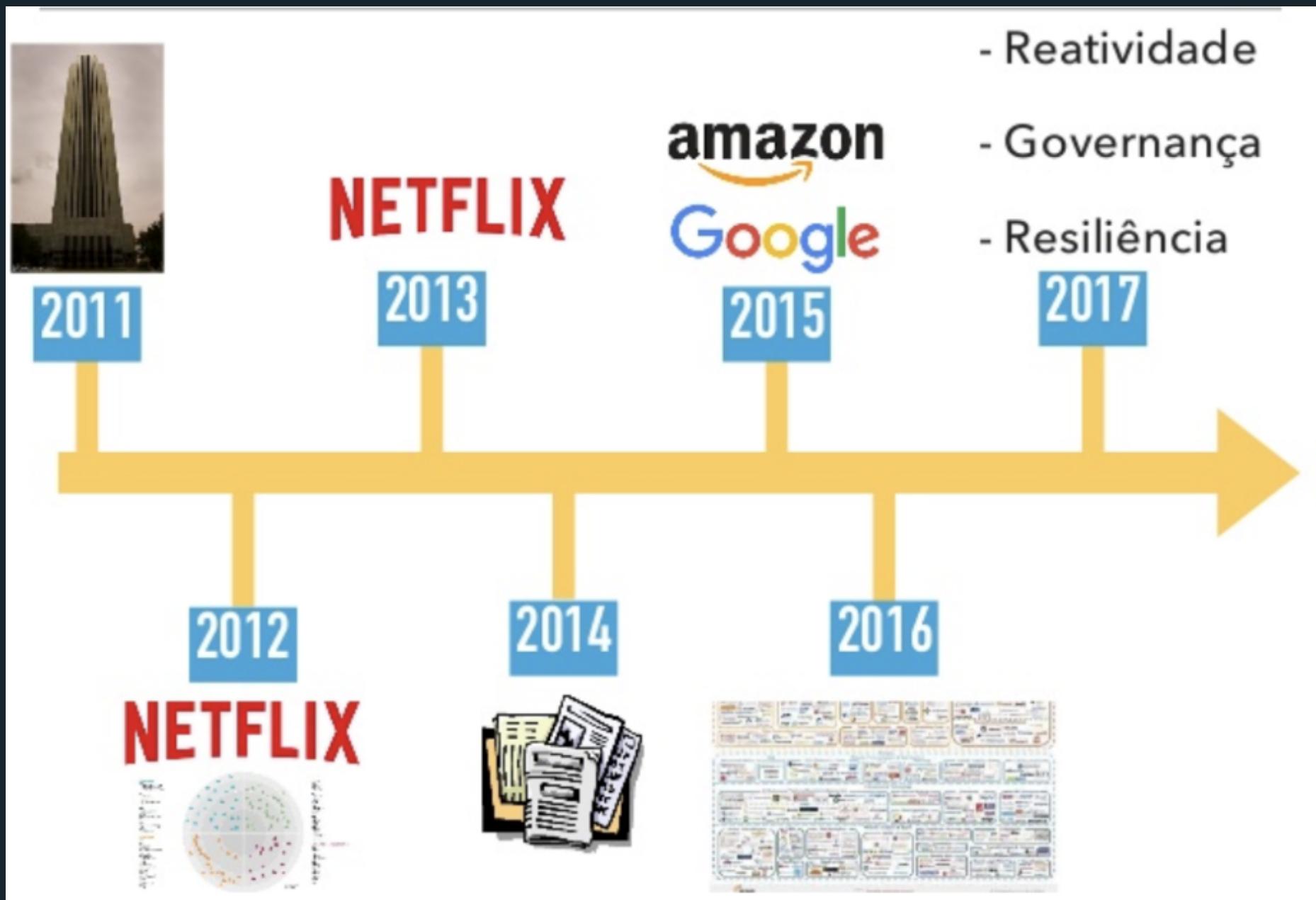
- Aumento da complexidade da coordenação;
- Comunicação entre os microsserviços;
- Governança;





CARACTERÍSTICAS





Características

- Um conjunto de pequenos autônomos que trabalham juntos. "Newman, Sam";
- Software modularizado em pequenos serviços que se comunicam através de uma forma padronizada;
- Se comunicam através de uma API Restfull (HTTP / Json);

Características Técnicas

- Out-of-process:
 - Possibilidade de execução fora dos processos.
- Chamadas remotas:
 - Microsserviços são acessados por chamadas remotas.
- Independente de linguagem de programação:
 - São agnósticos a linguagem de programação, ou seja, você pode ter serviços escritos em node, java, python, etc.

Características Técnicas

- Baixo acoplamento:
 - Você é dono somente do seu domínio de negócio. Não necessitando de outro serviço para gerar novas versões e ou evoluir seu produto;
- Escalabilidade horizontal e vertical:
 - Você pode aumentar o número de réplicas (scale horizontal) e/ou aumentar a capacidade computacional de seu serviço (scale vertical);

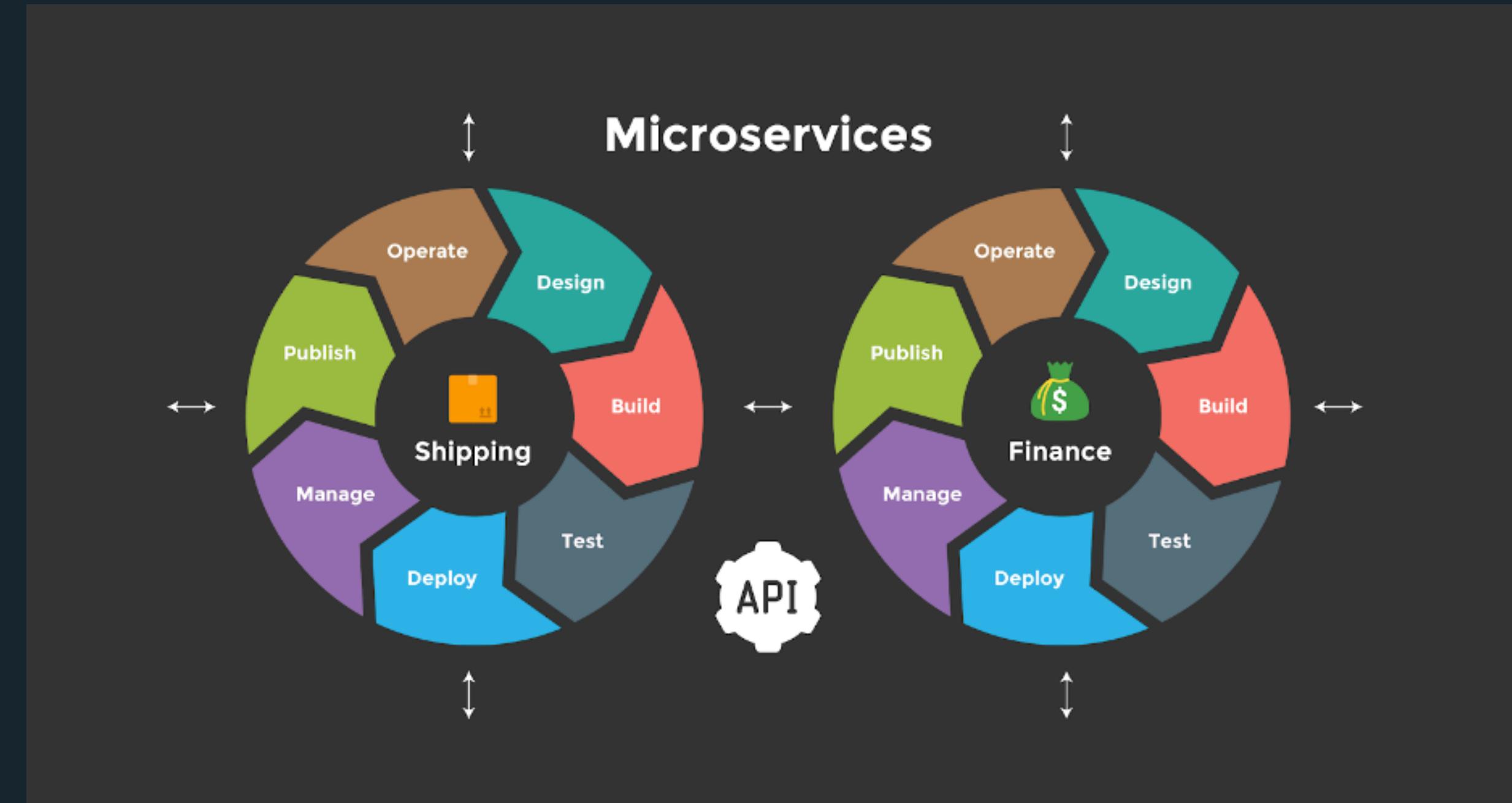
Características Organizacionais

- Agilidade:
 - Trabalhando em conjuntos de negócio menores você garante agilidade no desenvolvimento de software.
- Equipe pequena e focada:
 - Utilizando o conceito de two-pizzas team você garante foco e produtividade do time de desenvolvimento.

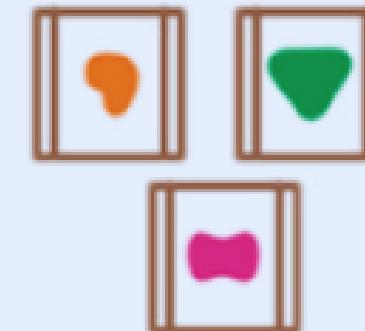
Características Organizacionais

- Entregas rápidas:
 - Por ser altamente testável, um micro serviço pode estar disponível rapidamente em produção com garantias de segurança e qualidade.
- Combinação de tecnologias:
 - Times podem ser multidisciplinares em tecnologias para desenvolver os microsserviços.

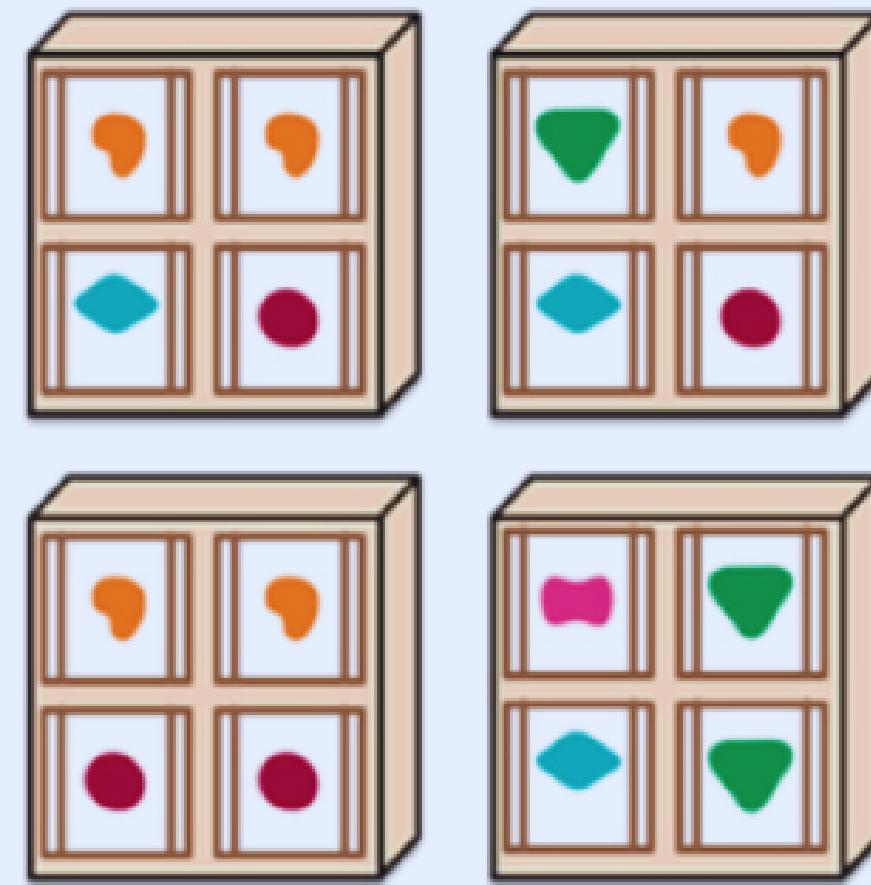
Características Organizacionais



A arquitetura de micro-serviços coloca cada elemento de funcionalidade em um serviço separado...



...e escala distribuindo os serviços entre os servidores, replicando por demanda.



Práticas recomendadas

- Modele os serviços em torno de domínio da empresa.
- Descentralize tudo. Equipes individuais são responsáveis por projetar e criar serviços. Evite compartilhar esquemas de dados ou códigos.
- O armazenamento de dados deve ser privado para o serviço que é o proprietário dos dados. Use o melhor armazenamento para cada serviço e tipo de dados.

Práticas recomendadas

- Os serviços comunicam-se por meio de APIs bem projetadas. Evite o vazamento de detalhes da implementação. As APIs devem modelar o domínio, não a implementação interna do serviço.
- Evite acoplamento entre serviços. Causas de acoplamento incluem protocolos de comunicação rígidos e esquemas de banco de dados compartilhados.

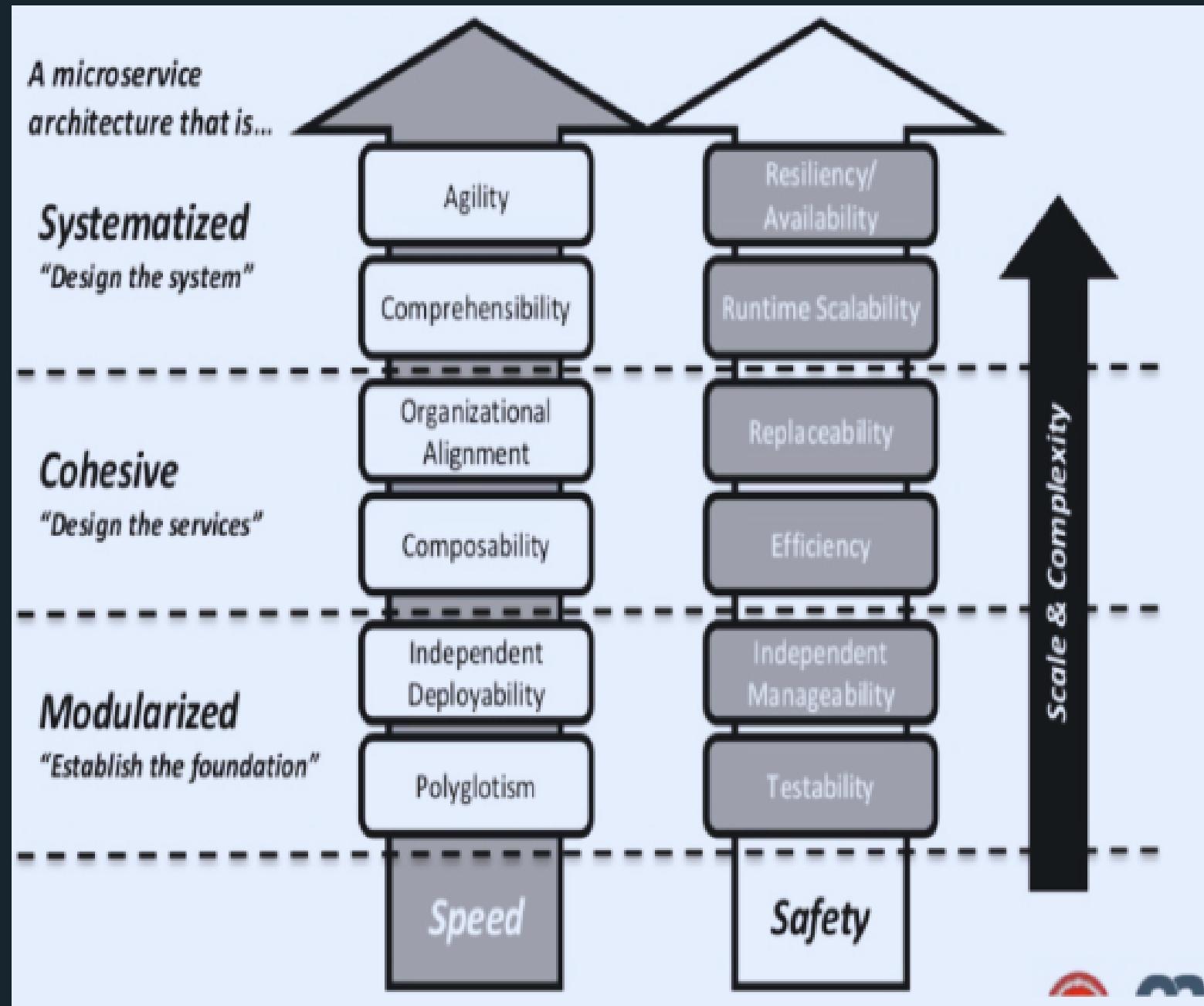
Práticas recomendadas

- Descarregue preocupações abrangentes, como autenticação e terminação SSL, para o gateway.
- Mantenha o conhecimento de domínio fora do gateway.
- Os serviços devem ter um acoplamento flexível e alta coesão funcional.
- Isole falhas. Use estratégias de resiliência para impedir que falhas em um serviço distribuam-se em cascata.

Capabilities

- Agilidade
- Compreensibilidade
- Alinhamento Organizacional
- Composição
- Deploy independente
- Múltiplas linguagens

VELOCIDADE



SEGURANÇA

**"Everything fails
All the time"**

Werner Vogels

Falácias

- A rede é confiável;
- Latência é zero;
- A banda de internet é infinita;
- A rede é segura;
- A topologia de rede não muda;
- Existe somente UM administrador;
- Custo de transporte é zero;
- A rede é homogênea;



IaaS

Infraestrutura como Serviço

Características

- Infrastructure as a Service significa entregar computação de infraestrutura sob demanda.
- É um dos 3 modelos de serviços da computação em nuvem.
- IaaS provê:
 - Servidores: Computação e máquinas;
 - Storage;
 - Rede
 - Sistemas Operacionais;

Características

- O usuário ao invés de adquirir softwares ou máquinas, espaço em data centers ou equipamentos de rede ele praticamente aluga espaços para estes recursos em uma infraestrutura externa.

Características

- IaaS pode ser obtida em:
 - Nuvem pública:
 - É considerada nuvem pública uma infraestrutura que consiste de recursos compartilhados, liberados sob demanda baseado na internet.
 - Nuvem privada:
 - Incorpora a maioria das features de uma nuvem pública como virtualização porém fica dentro de uma rede privada.
 - Nuvem Híbrida:
 - Mistura de uma nuvem privada com uma nuvem pública, geralmente conectadas através de um túnel VPN.

Características

- Recursos são distribuídos como serviço;
- Possibilita escalabilidade dinâmica;
- Custos variados;

Quando utilizar

- Quando a demanda for volátil, ou seja, você tem a possibilidade diminuir e aumentar sua capacidade computacional de acordo com a necessidade.
- Empresas sem capacidade de investimento em hardware.
- Empresas com crescimento rápido e necessidade de escala rápida de sua infraestrutura.
- Por estratégias de negócios rápidas.

Quando não utilizar

- Quando a legislação não permitir guardar os dados fora da infraestrutura interna da empresa, ou a terceirização do armazenamento não é permitida.
- Não é aconselhável quando os níveis de desempenho necessários para aplicação tenham limite de acesso ao provedor da cloud.



PaaS

Plataforma como Serviço

Características

- PaaS pode ser considerada IaaS adicionada uma camada middleware e/ou componentes prontos.
- Uma camada de abstração entre seu aplicativo em nuvem e seu provedor de IaaS.
- É um ambiente de execução escalável e com alta disponibilidade para aplicações customizadas.

Características

- PaaS, é uma categoria de computação em nuvem que fornece uma plataforma e um ambiente para permitir que os desenvolvedores criem aplicativos e serviços pela Internet.
- Fornece fundamentalmente escala elástica do seu aplicativo.

Benefícios

- Infraestrutura na nuvem, escalável e com alta disponibilidade nativa.
- Alta produtividade no desenvolvimento e manutenção de aplicações sob demanda.
- Resumindo: baixo custo (TCO), confiabilidade e diminuição do tempo de entrega.

Vantagens

- Desenvolvimento 100% focado no negócio:
 - Por direcionar a arquitetura lógica e administrar a arquitetura física de forma bem transparente, é possível desenvolver uma aplicação que vá demandar uma requisição por minuto ou 10 mil requisições por segundo da mesma forma, com o mesmo nível de preocupação do ponto de vista técnico: apenas a lógica de negócio.

Vantagens

- Produtividade:
 - O simples fato de não se gerenciar balanceamento de carga, replicação, cluster, instalando e configurando middlewares (servidores de aplicação, banco de dados, etc.) já é um grande ganho. Além disso, os grandes fornecedores estão criando camadas de componentes prontos para uso, APIs e aceleradores de desenvolvimento nessas plataformas.

Decisões

- A plataforma por trás do PaaS foi criada justamente para trazer benefícios e acelerar o desenvolvimento. Existe esforço dos fornecedores para deixar essa camada o mais padrão possível, mas ainda existe uma boa parte que é proprietária.
- Ao adotar PaaS, é natural que se adote também essa camada proprietária, caso contrário, poderia se trabalhar direto na infraestrutura.

Decisões

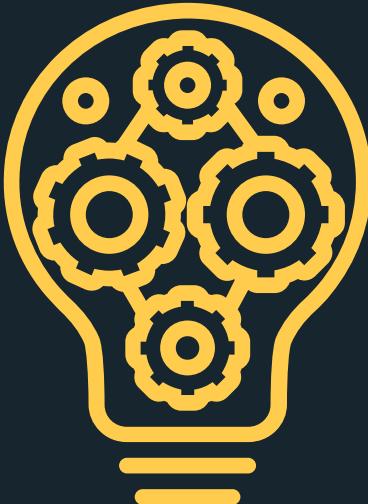
- É esta camada que permite dar um salto de produtividade e lidar com escalabilidade e disponibilidade de forma transparente.
- A decisão a ser tomada é: menos custo e mais entregas contra o efeito “lock-in” das aplicações construídas nessa abordagem!.

Restrições

- Plataformas são muito eficientes para construção de novas aplicações.
- A migração de aplicações já existentes para elas é um processo custoso ou mesmo inviável (dependendo da tecnologia atual e da plataforma almejada).

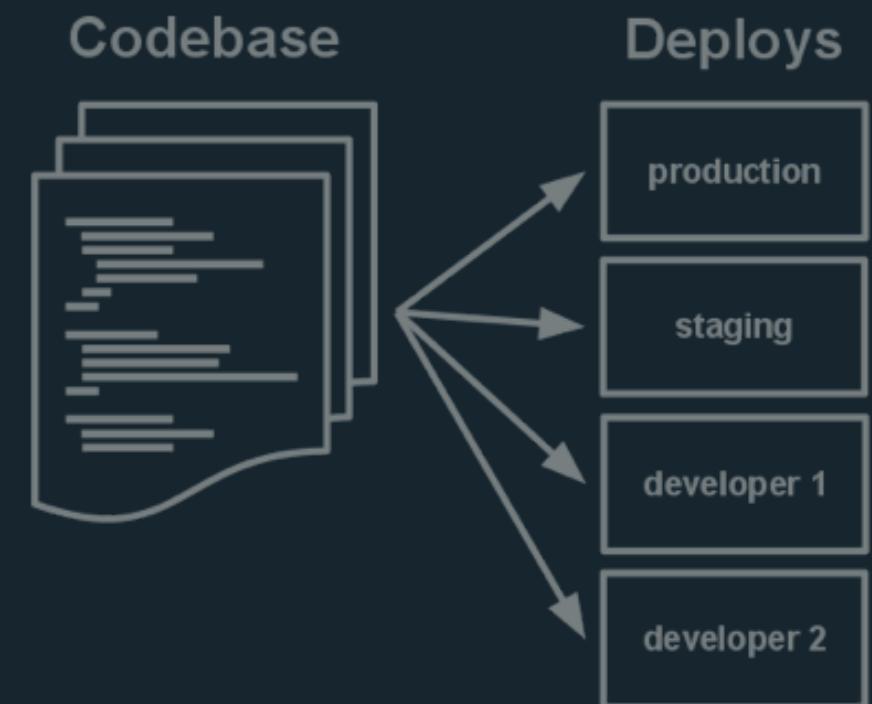
THE TWELVE

Factor for App



1 - Base de Código

- Somente uma base de código por aplicação;
- Vários deploys por aplicação;
- Desenvolvedor possui uma cópia local do repositório;

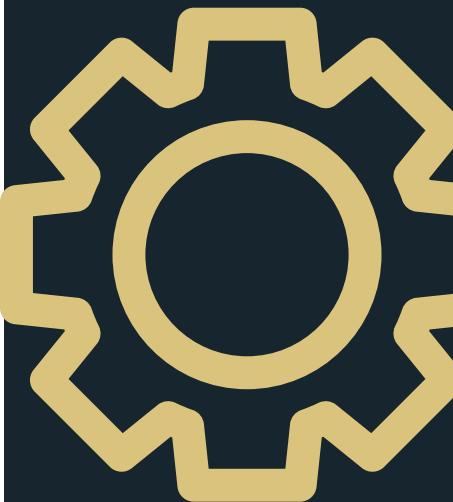


2 - Dependências

- Declare e isole explicitamente as dependências;
- Uma aplicação 12 fatores nunca confia na existência implícita de pacotes em todo o sistema;
- Uma declaração de dependência explícita é que simplifica a configuração da aplicação para novos desenvolvedores;
- Na prática:
 - Tenha sempre um gerenciador de dependências configurado para seu projeto (maven, gradle, npm, pip e etc);

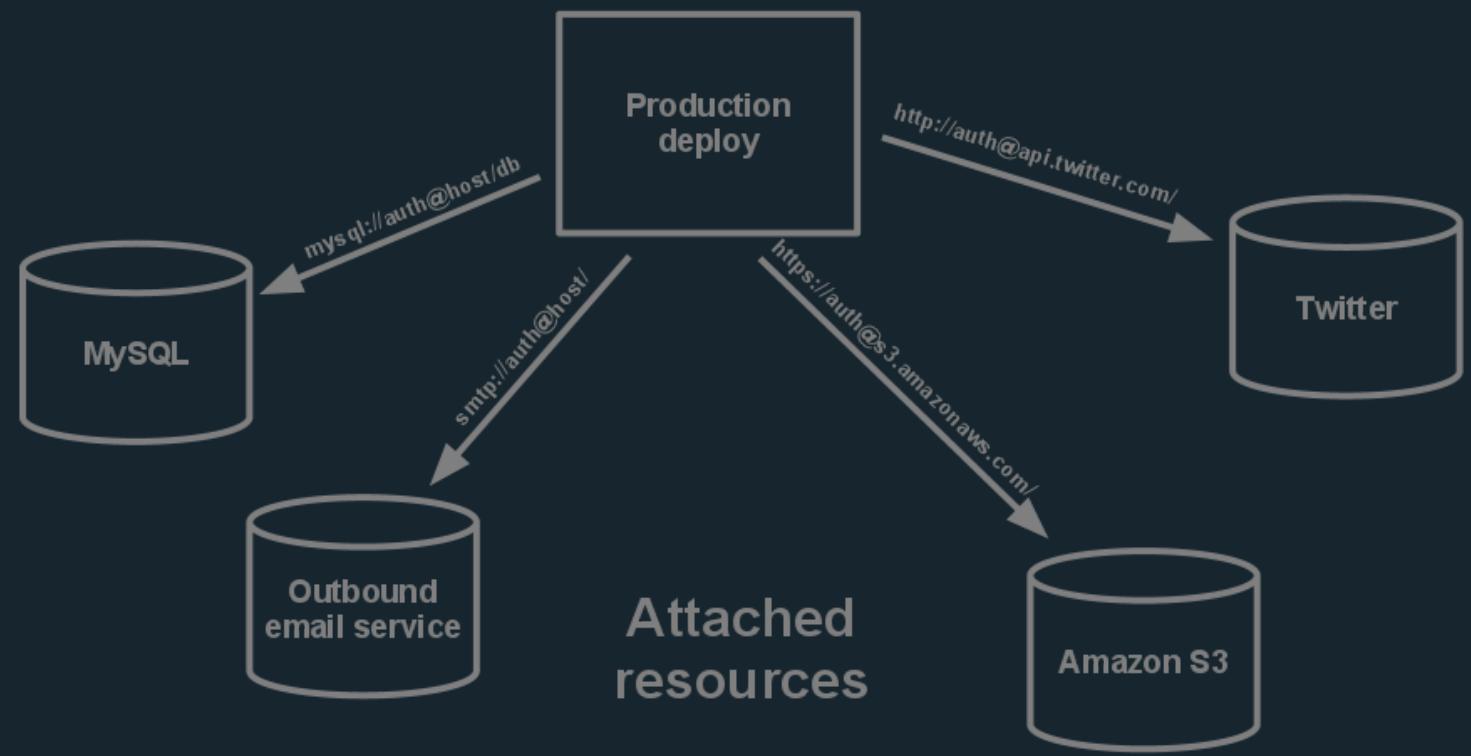
3 - Configurações

- A configuração de uma aplicação é tudo que é provável variar entre deploys(homologação, produção, desenvolvimento, etc).
- Uma aplicação 12 fatores armazena configuração em variáveis de ambiente ou algum recurso de configuração distribuída.
- Necessitamos de facilidade na troca de ambientes sem ter a necessidade de alterar o codebase.



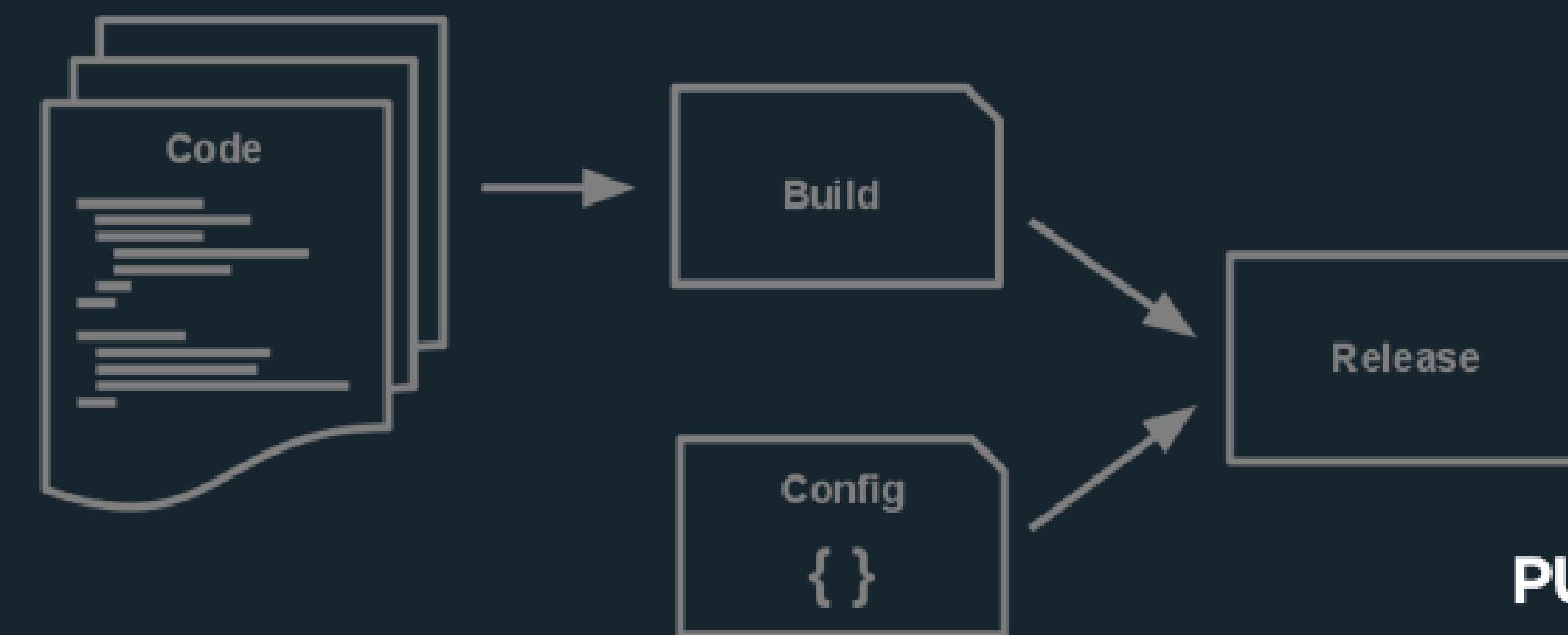
4 - Serviços de Apoio

- Trate serviços de apoio como recursos anexados;
- Serviço de apoio é qualquer serviço que o App consuma via rede como parte de sua operação normal:
 - Ex: Banco de Dados, Mensageria, Cache.
- Não se deve fazer distinção entre serviços locais e terceiros;



5 - Construa, lance, execute

- Uma base de código é transformada em um deploy através de 3 estágios:
 - Construção: Converte o repositório em um pacote executável;
 - Lançamento: Combina o artefato construído com a configuração do deploy.
 - Execução: Roda o app no ambiente de execução através dos processos específicos do APP.
- O app 12 fatores utiliza separação estrita entre os estágios de construção, lançamento e execução.



6 - Processos

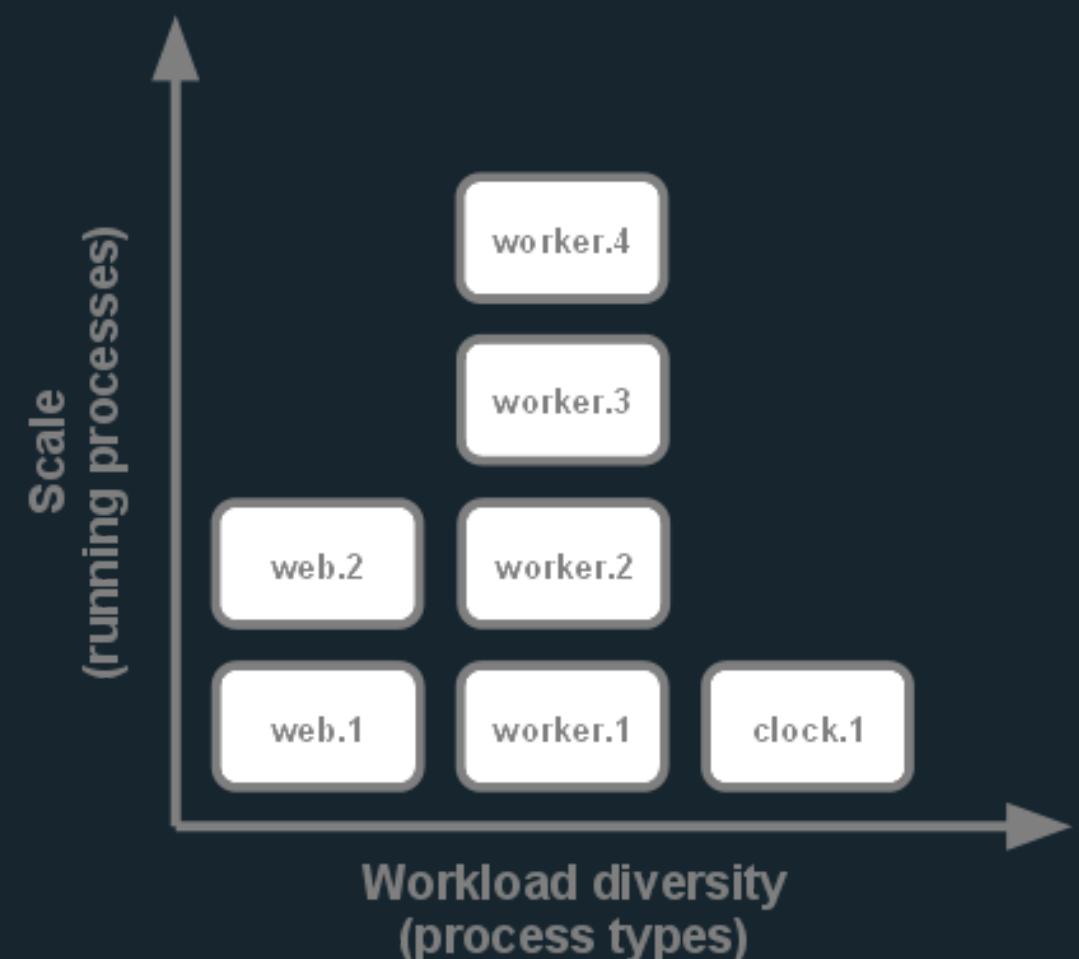
- Você não deve introduzir estado em seus serviços; os aplicativos devem ser executados como um processo único e sem estado.
- Os processos dos Doze Fatores são stateless e não compartilham nada. Esse fator está no núcleo da arquitetura de microserviços.

7 - Vínculo de porta

- Seu serviço deve estar visível para outras pessoas via ligação de alguma porta.
Se você criou um serviço, verifique se outros serviços podem tratar isso como um recurso, se assim o desejarem. O aplicativo de doze fatores é completamente independente de outros recursos.

8 - Concorrência

- Divilde seu aplicativo em pequenos pedaços, em vez de tentar aumentar seu aplicativo (executando uma única instância na máquina mais poderosa disponível). Aplicativos pequenos e definidos permitem a expansão conforme necessário para lidar com as cargas variadas. O processo deve ser dimensionado individualmente, com o Fator 6 (sem estado), torna-se transparente este tipo de abordagem.



9 - Descartabilidade

- Os processos devem consumir menos tempo. Certifique-se de poder correr e parar rapidamente. E que você pode lidar com falhas. Sem isso, o dimensionamento automático e a facilidade de implantação e desenvolvimento estão sendo diminuídos. Você pode conseguir isso com contêineres.

10 - Paridade entre desenvolvimento e produção

- Mantenha o desenvolvimento, a homologação e a produção o mais semelhante possível, para que qualquer pessoa possa utilizá-lo da mesma forma. A implantação contínua precisa de integração contínua com base em ambientes correspondentes para limitar desvios e erros. Isso também incentiva implicitamente uma cultura DevOps na qual o Desenvolvimento e as Operações de Software são unificados.

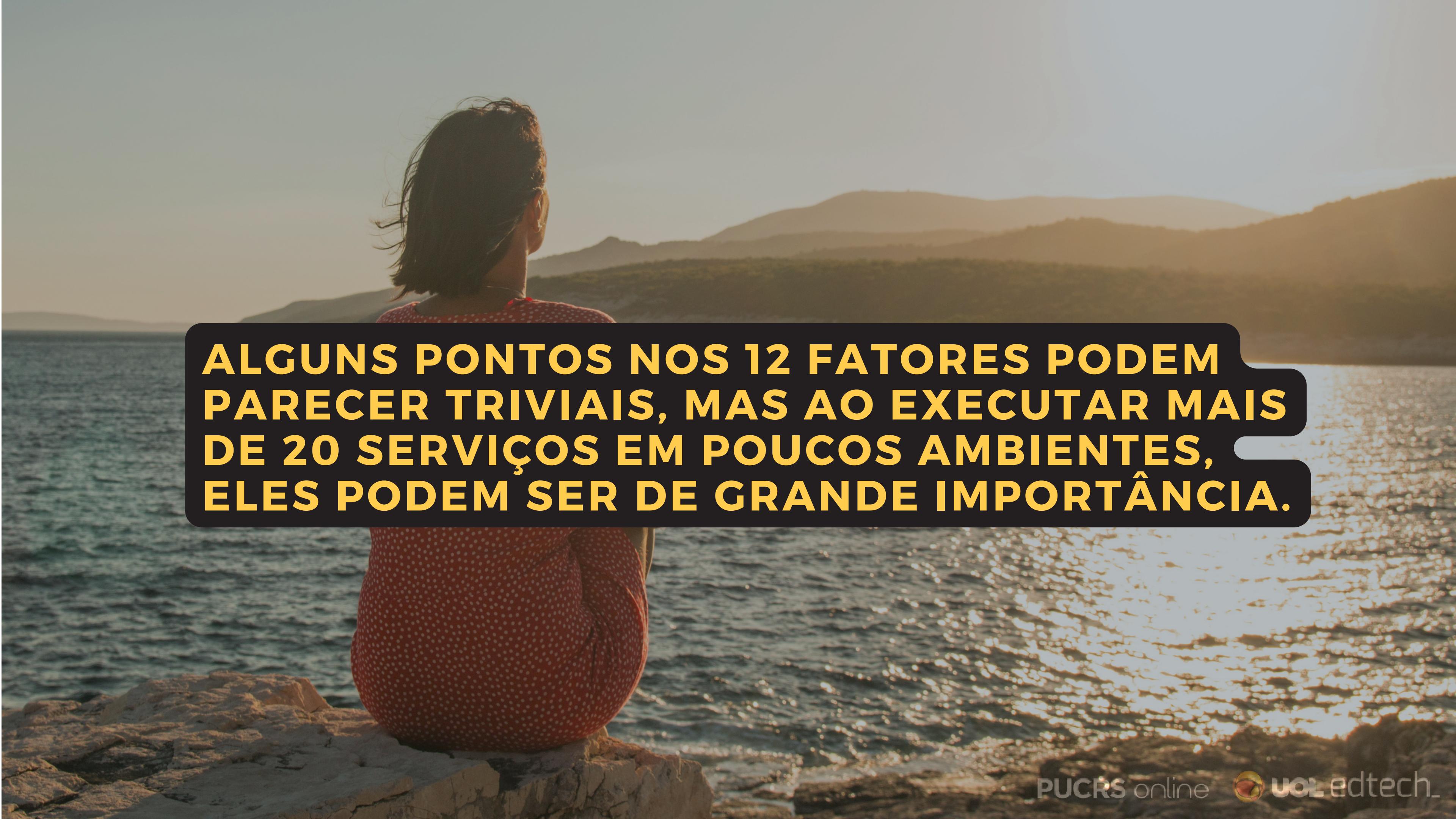
A containerização é uma grande ajuda aqui.

11 - Logs

- Trate os logs como fluxos de eventos. O registro é importante para validar erros e também verificar a integridade geral do seu sistema. Ao mesmo tempo, seu aplicativo não deve se preocupar com o armazenamento dessas informações. Esses logs devem ser tratados como um fluxo contínuo capturado e armazenado por um serviço separado.

12 - Processos Administrativos

- Execute tarefas administrativas / gerenciamento como processos pontuais - tarefas como migração de banco de dados ou execução de scripts pontuais no ambiente. Para evitar mexer com o banco de dados, use as ferramentas criadas ao lado do aplicativo e isole completamente sua aplicação por exemplo.



**ALGUNS PONTOS NOS 12 FATORES PODEM
PARECER TRIVIAIS, MAS AO EXECUTAR MAIS
DE 20 SERVIÇOS EM POUcos AMBIENTES,
ELES PODEM SER DE GRANDE IMPORTÂNCIA.**

APLICAÇÕES

Cloud Native



Definição - CNCF

- Containerizado;
- Gerenciado dinamicamente;
- Orientado a microserviços:
 - Automação;
 - Registro e Descoberta;
 - Rastreamento distribuído / Observabilidade;
- Cloud:
 - Elasticidade;
 - Modelo on-demand;

Definição - Pivotal

- Devops:
 - Processos;
 - Ferramentas;
 - Cultura;
- Entrega contínua:
 - Automação;
- Microsserviços:
 - Automação;
 - Registro e Descoberta;
 - Rastreamento distribuído / Observabilidade;
 - Anti-fragilidade / Engenharia de caos;

Definição - Pivotal

- BOSH:
 - Suporte a múltiplas clouds (evitar ficar preso a um provider);
 - Separação clara entre sistemas;
 - Provisionamento rápido;
 - Escalabilidade;
 - Monitoramento de saúde;
 - Controle de falhas;
 - Deploy canário;

**Quando falamos sobre
CLOUD NATIVE falamos sobre
ABSTRAÇÕES**

**NÃO ESTAMOS FALANDO
SOBRE
KUBERNETES
OU
CONTAINERS**

Características

- Arquiteturas cloud native aprimoram nossa capacidade de praticar DevOps e Entrega Contínua (Continuous Delivery), e elas exploram as características da infraestrutura na nuvem (Cloud Infrastructure).
- "Cloud-native" é um adjetivo que descreve as aplicações, arquiteturas, plataformas / infraestrutura, e processos que, em conjunto, tornam "econômico" trabalhar de forma a melhorar nossa capacidade de responder rapidamente às mudanças e reduzir a imprevisibilidade.

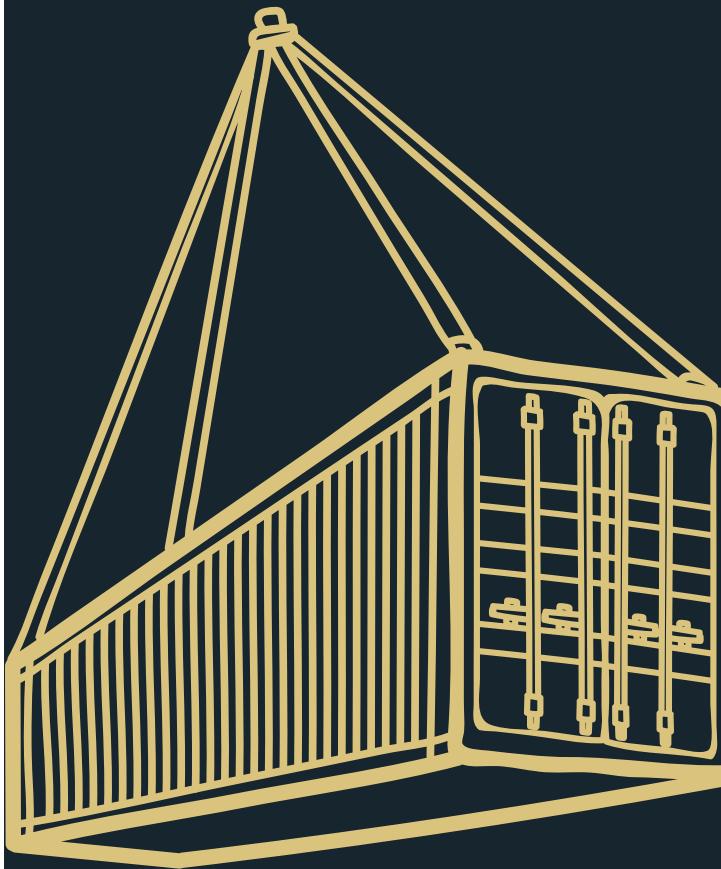
Características

- Arquiteturas cloud-native aprimoram nossa capacidade de praticar DevOps e Continuous Delivery, e elas exploram as características da infraestrutura na nuvem. Eu defino arquiteturas cloud-native como tendo as seguintes seis qualidades:
 - Modularidade (através de Microservices);
 - Capacidade de observação;
 - Implementabilidade;
 - Testabilidade;
 - Descartabilidade;
 - Substituível;

Características

- Criado para ser escalável;
- Tolerante a falhas;
- Decomposto em serviços;
- Envia o máximo trabalho para a plataforma se possível;
- Automatizado;

CONTAINERS



MAS NA MINHA
MÁQUINA FUNCIONA

Características

- Pequenos sistemas Linux minimalistas;
- Compartilhado Kernel do Host;
- Processos trabalhando isoladamente;
- Commits e versionamento de containers;
- Compartilhamento de ambientes customizados;

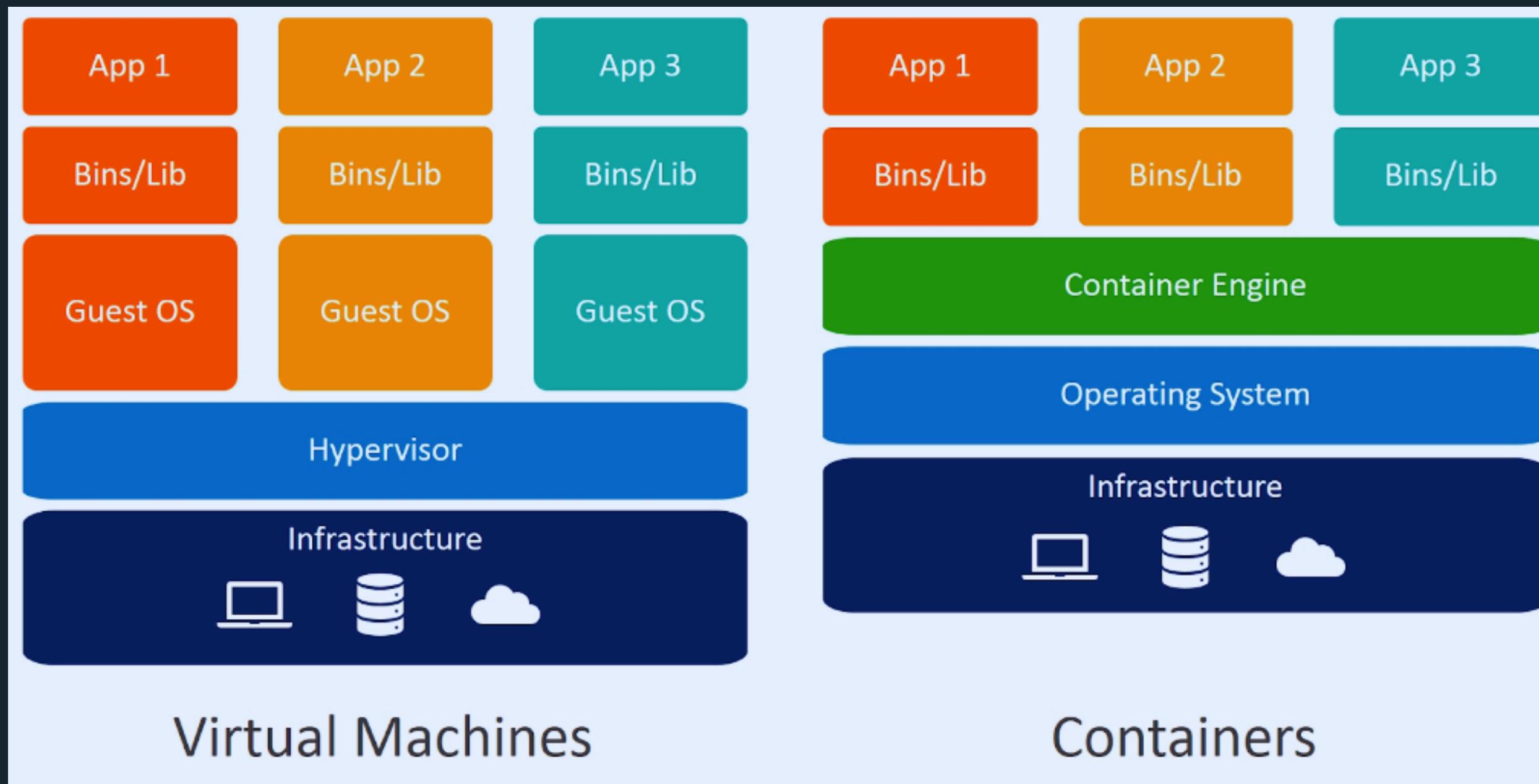
Características

- Por que os contêineres são bons para micro serviços?
 - Projetado para executar um aplicativo por contêiner;
 - Separação natural da carga de trabalho;
 - Muito leve;
 - Ótimo para dimensionar rapidamente;
 - Melhor uso de recursos;
 - Os contêineres compartilham o SO host e, quando apropriado, Binários e bibliotecas;

Características

- Formatos padrão de contêiner, como o Docker, são distribuições cruzadas entre linux compatível;
- Incrivelmente fácil de mover sua carga de trabalho • Equilibre melhor os recursos do sistema;
- Permita que os desenvolvedores trabalhem em um ambiente de produção simulado;
- Remove o problema "funcionou na minha máquina";

Características

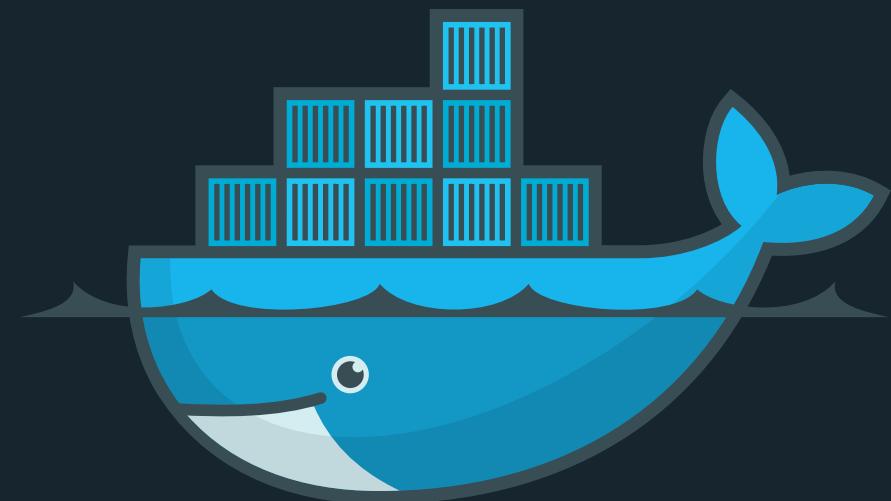


Características



Docker

Tecnologia Open Source que permite criar, executar, testar e implantar aplicações distribuídas dentro de containers de software. Ele permite que você empacote um software de uma padronizada para o desenvolvimento de software, contendo tudo que é necessário para a execução: código, runtime, ferramentas, bibliotecas, etc. Docker permite que você implante aplicações rapidamente, de modo confiável e estável, em qualquer ambiente.

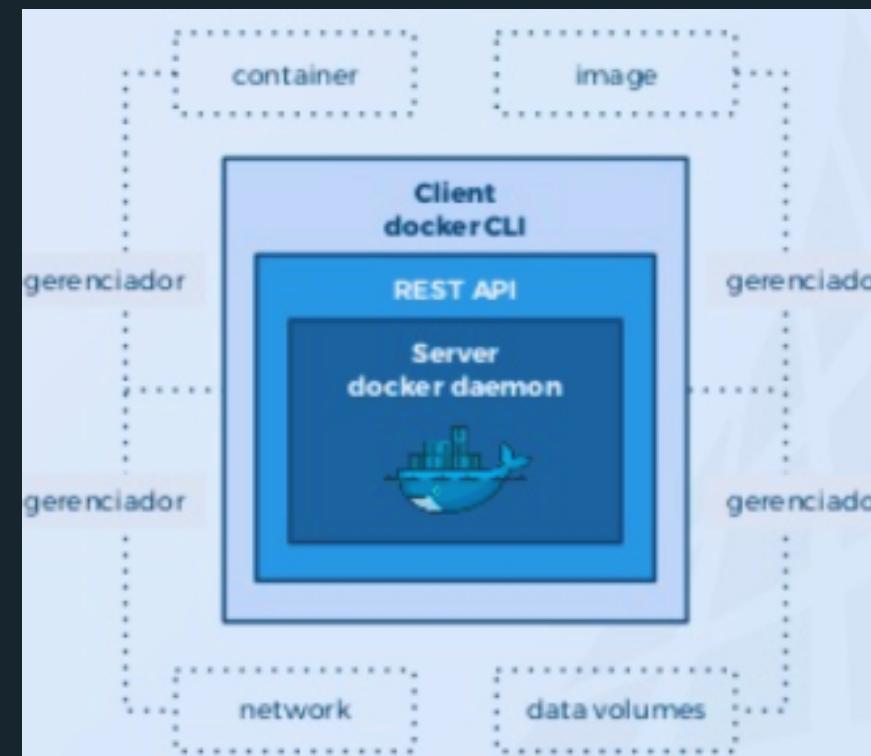


Docker

- Existem mais de 500 mil aplicações Dockerizadas, um crescimento de 3100% ao longo de 2 anos;
- Mais de 4 bilhões de containers já foram puxados até hoje;
- Docker é apoiado por uma grande e crescente comunidade de colaboradores e usuários;
- A adoção do Docker aumentou mais de 30% no último ano;
- Cerca de 30% dos containers Dockers estão rodando em produção. \29% das empresas que já ouviram falar em Docker planejam usá-lo;

Docker

A parte cliente fala com o Docker daemon, que faz o trabalho pesado de construção, execução e distribuição de seus containers e imagens Docker, também controla os recursos executados. O cliente Docker e Docker daemon, podem ser executados no mesmo sistema, também é possível conectar um cliente Docker a um Docker daemon remoto. O cliente Docker e daemon se comunicam através de uma API REST, através de sockets UNIX ou uma interface de rede, para execuções de comandos ou scripts.

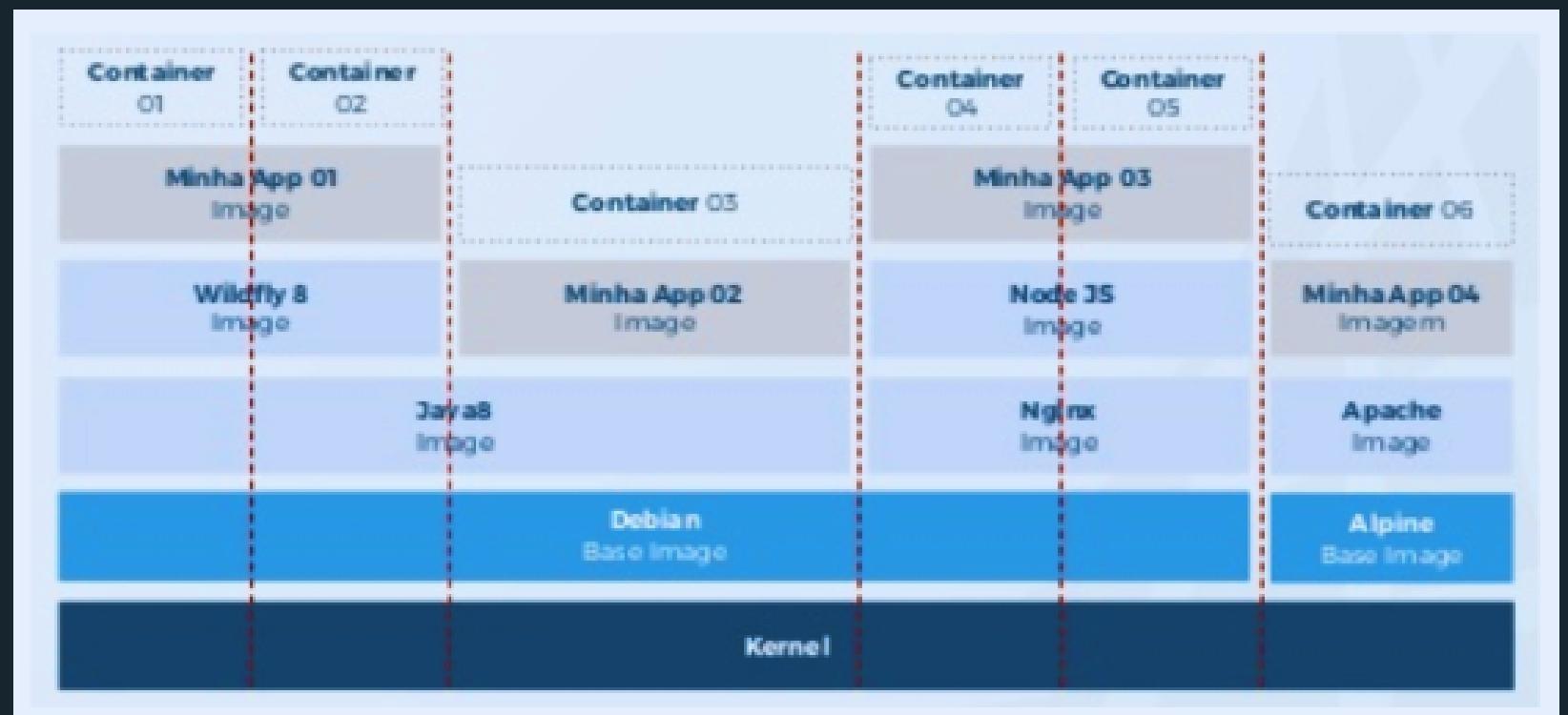


Container

- Containers tem como base sempre uma imagem, pense como na seguinte analogia do mundo Java, uma imagem é uma classe e um container é como um objeto instância dessa classe, então podemos através de uma imagem “instanciar” vários containers, também através de recursos chroot, Cgroups é possível definirmos limitações de recursos recursos e isolamento parcial ou total dos mesmos.
- Algumas características dos containers
 - Portabilidade de aplicação;
 - Isolamento de processos;
 - Prevenção de violação externa;
 - Gerenciamento de consumo de recursos;

Imagen

Imagens são templates para criação de containers, como falado no slide anterior, imagens são imutáveis, para executá-las é necessário criar uma instância dela (container), também vale ressaltar que as imagens são construídas em camadas, o que facilita sua reutilização e manutenção. Em resumo uma imagem nada mais é do que um ambiente totalmente encapsulado e pronto para ser replicado onde desejar.



Dockerfile

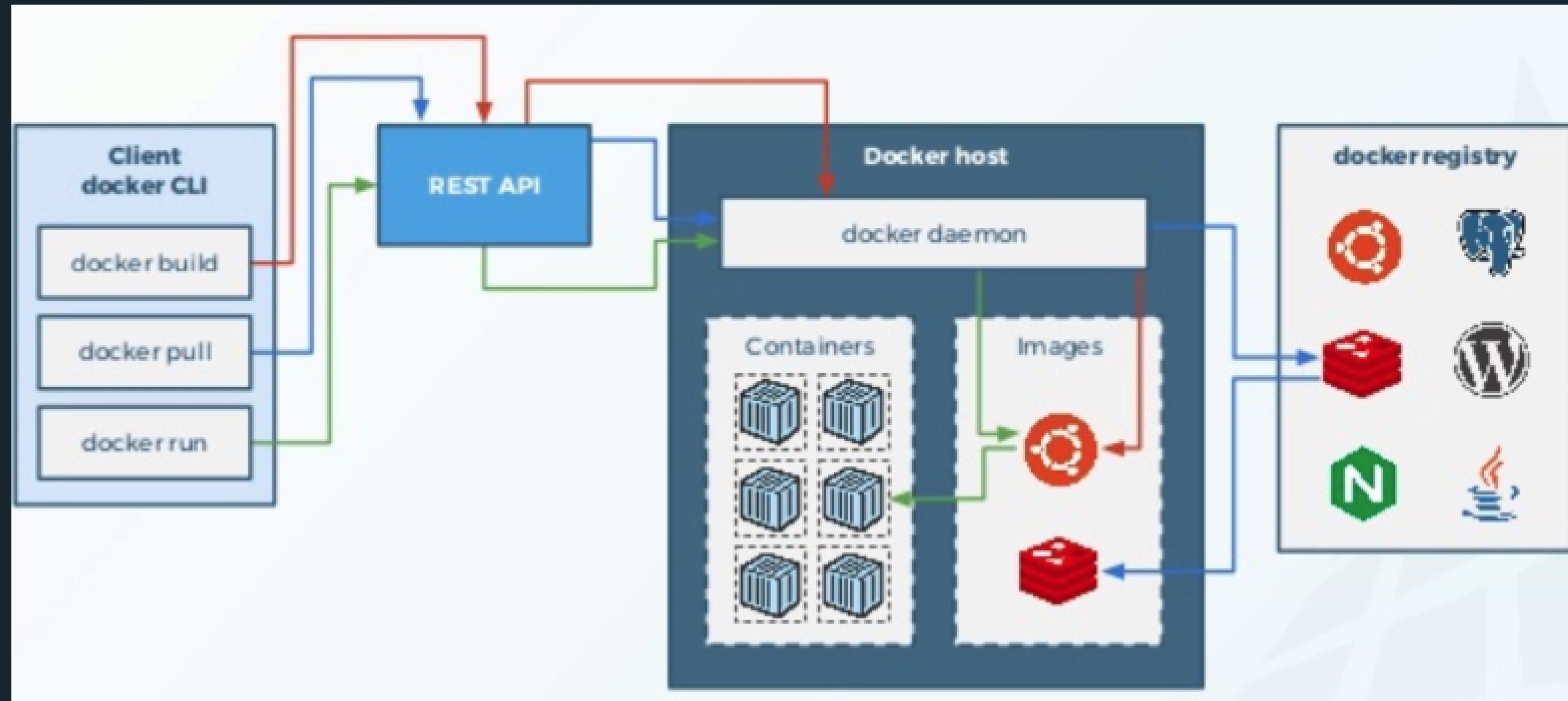
São scripts com uma série de comandos para criação de uma imagem, nesses scripts podemos fazer uma séries de coisas como executar comandos sh, criar variáveis de ambiente, copiar arquivos e pastas do host para dentro da imagem.

```
FROM java:8-jdk-alpine
COPY target/spring-boot-docker-demo-0.0.1-SNAPSHOT.jar /usr/app/
WORKDIR /usr/app
RUN sh -c 'touch spring-boot-docker-demo-0.0.1-SNAPSHOT.jar'
ENTRYPOINT ["java","-jar","spring-boot-docker-demo-0.0.1-SNAPSHOT.jar"]
```

Docker Registry

É como um repositório GIT, onde as imagens podem ser versionadas, comitadas, “puxadas” etc, quando recuperamos uma imagem, usando o comando docker pull por exemplo, estamos normalmente baixando a imagem de um registro Docker, o repositório oficial do Docker é o Docker HUB, onde é possível hospedar e versionar imagens públicas e privadas.

Arquitetura Básica



PUCRS online  uol edtech