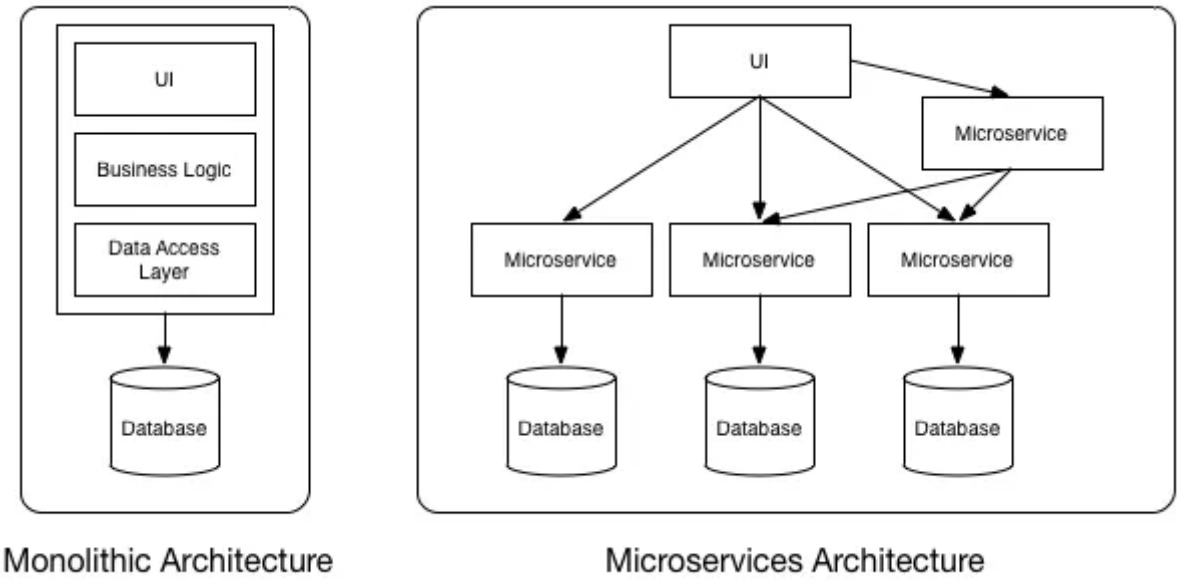
Published in codigorefinado

Clayton K. N. Passos Follow
Dec 29, 2019 · 5 min read

Save

Padrões de Microserviços

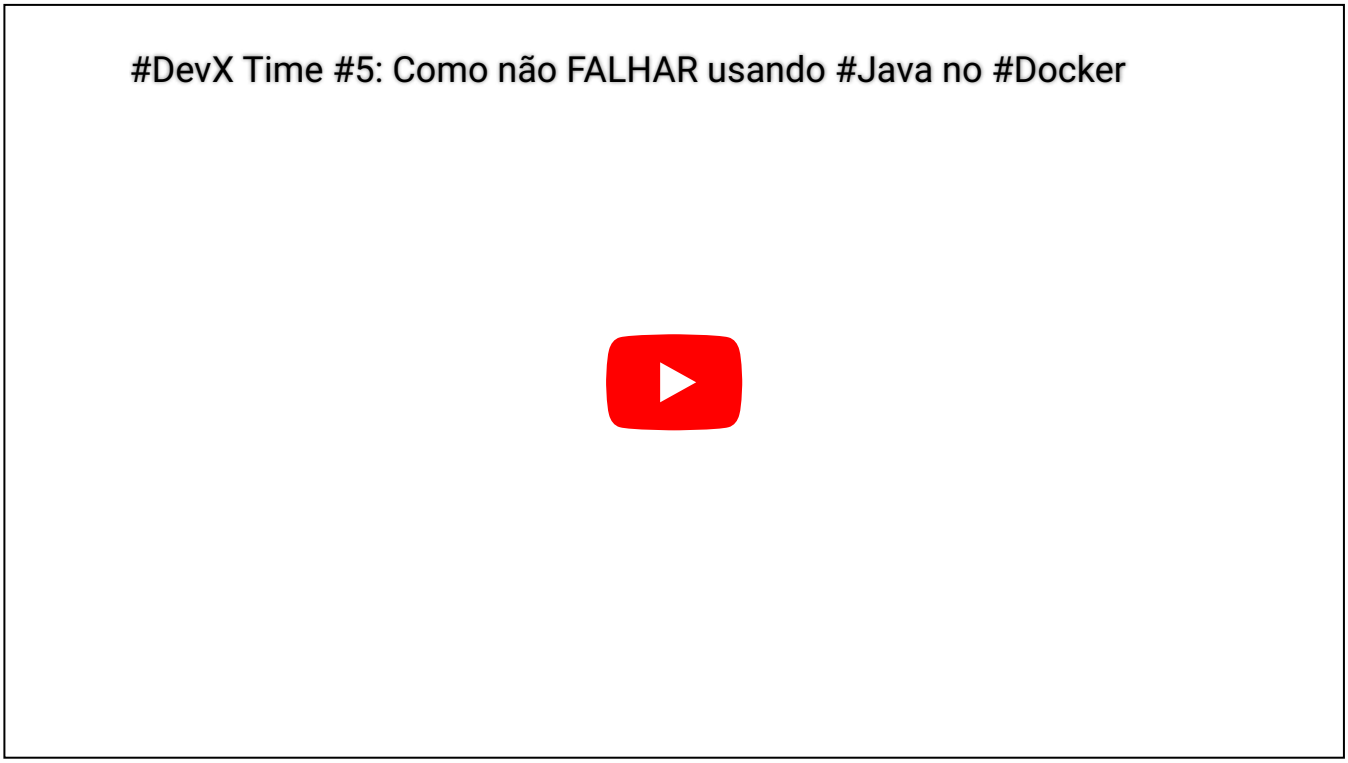


Monolithic Architecture — Microservices Architecture

Bem, acredito que você já sabe oque é, quais vantagens e desvantagens das duas arquiteturas, moni litica e de micro serviços. Se não sabe, fica de tarefa de casa para pesquisar, OK?

Java com Docker

Talvez você queira desenvolver seus micro serviços em Java, é provável que você também queira executar suas aplicações em um conteiner Docker, oque talvez você não saiba é que **apenas apartir da OpenJDK 12 que o Java esta preparado para ter aplicações executando dentro do Docker efetivamente**. Quer saber mais? Da uma olhada no vídeo abaixo:



Resumindo, caso você queira utilizar OpenJDK 8 você deve usar a imagem docker da fabric8, ou passar parâmetros para controlar o consumo de memória.

- -XX:+UnlockExperimentalVMOptions
- -XX:+UseCGroupMemoryLimitForHeap

Se você quer criar micro-serviços utilizando Java você também deve conhecer:

- Graal que precisa ser habilitada na OpenJDK e JVM,
- GraalVM
- Quarkus

Rest/json o padrão de fato

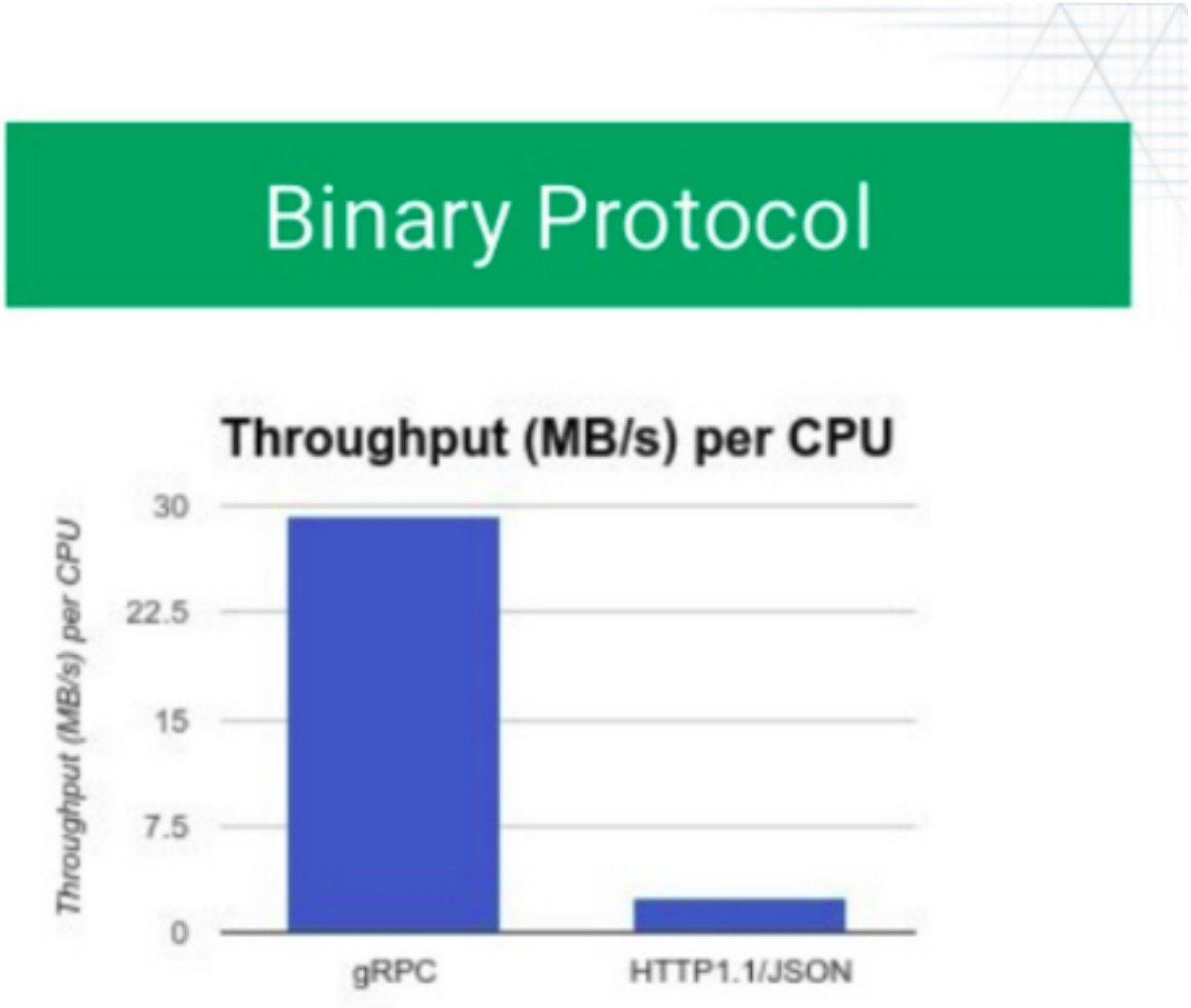
Provavelmente você deve conhecer Rest, como forma de trafegar informações entre micro serviços, ele é um padrão de fato, muitos consideram json leve, isto se deve a comparação com SOA (xml). Mas e quando json se torna pesado, qual é a opção? Temos algumas opções:

- gRPC — Algumas empresas estão utilizando para substituir o Restfull/Json, ele vai te lembrar a transferência de objetos Java via rede (RMI) que utilizávamos com EJB, desserializando e serializando objetos, mas de forma a ser independente de linguagem.

- MQTT — É um protocolo de comunicação mais utilizado em IoT (internet das coisas)

gRPC

gRPC é construido sobre o HTTP2, que é mais leve :D, veja uma demonstração no site http://www.http2demo.io/.



Por se tratar de um dado binário sendo trafegado na rede, até o momento gRPC deve ser evitado quando:

- Precisamos ter as informações sendo lidas por humanos
- Os dados devem ser consumidos pelo navegador
- O servidor for escrito em JavaScript

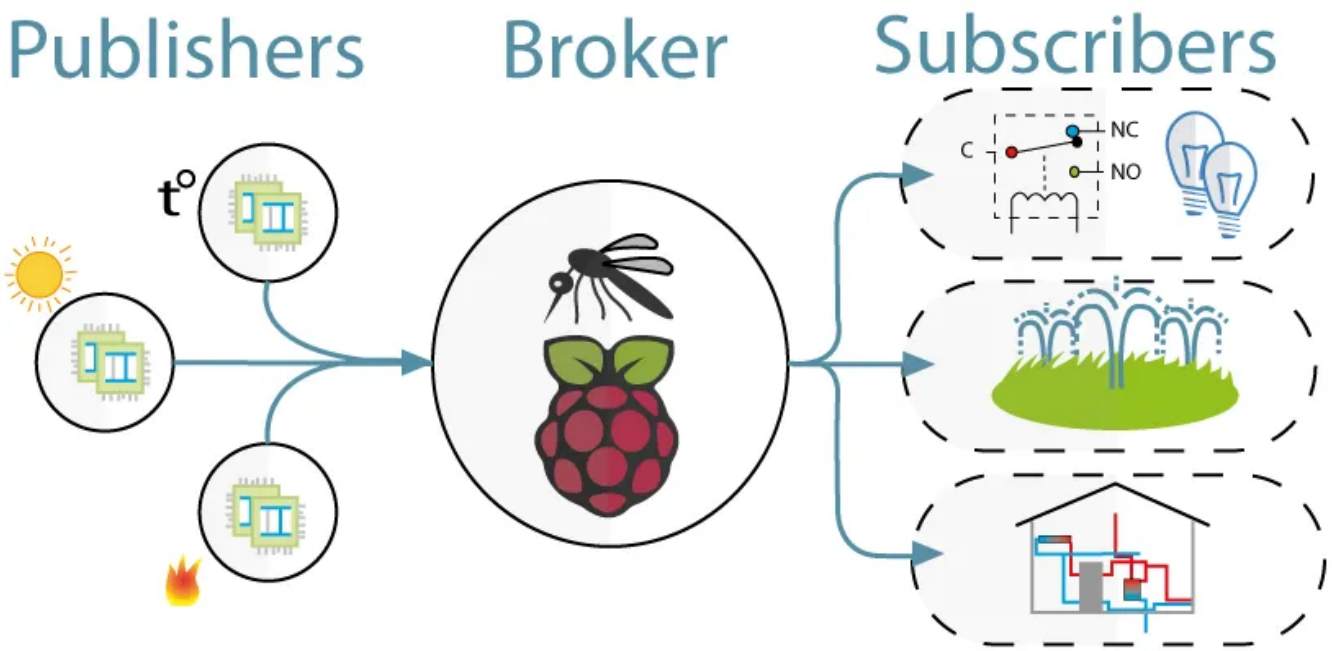
Apesar que estas indicações, são porque eles adicionariam algum overhead para conseguir realizar a operação, que seria desnecessário se fosse utilizado Restfull/json.

MQTT

O protocolo **MQTT** é a sigla para Message Queuing Telemetry Transport, ou seja, é o protocolo de mensagens entre máquinas. Desenvolvido pela IBM no final dos anos 90, ele tinha como objetivo principal ajudar na comunicação entre sensores e satélites. MQTT tem sido a melhor alternativa para a internet das coisas, já que pode suportar comunicações assíncronas, ou seja, pode suportar dados transmitidos em fluxos estáveis.

Debaixo do projeto eclipse, há uma série de projetos relacionados a IoT que você pode explorar, de uma olhada em: https://iot.eclipse.org/projects/

O principal caso de uso que tenho visto com este procolo é oque hoje conhecemos como PUB/SUB.



Para mim é relativamente comum encontrar arquiteturas distribuídas que lidam com diversos protocolos diferentes entre aplicações, e para você?

Não se engane

Não pense que apenas estes protocolos estão sendo utilizados, e nem mesmo que o uso se restringe aos casos de uso mais comuns, apesar de Rest/JSON ser o padrão de

fato, há várias empresas que demonstram que só o utilizam para se comunicar com o navegador (chrome, firefox, edge), que em sua rede interna utilizam gRPC, e o motivo é simples, JSON tem se mostrado pesado, sim há mais bytes sendo trafegados que que eles gostariam, e estão "enxugando bits" com gRPC para ter maior performance e menos banda sendo gasta na rede interna.

Ao adotar gRPC potencialmente você vai perder a comodidade do JSON, e ganhar complexidade de desenvolvimento e complexidade ao procurar bugs (troubleshooting).

Conheça Os doze fatores, antes de começar

Quando você for construir seus micro serviços, conheça os doze fatores, se começar sem este conhecimento, provavelmente você tomará decisões piores.
Provavelmente no monólito você também precisa lidar com estes doze itens, mas no micro serviço por estarmos em um ambiente de complexidade distribuída vale repensar nestes pontos:

I. Base de Código — Uma base de código com rastreamento utilizando controle de revisão, muitos deploys

II. Dependências — Declare e isole as dependências

III. Configurações — Armazene as configurações no ambiente

IV. Serviços de Apoio — Trate os serviços de apoio, como recursos ligados

V. Build, release, run — Separe estritamente os builds e execute em estágios

VI. Processos — Execute a aplicação como um ou mais processos que não armazenam estado

VII. Vínculo de porta — Exporte serviços por ligação de porta

VIII. Concorrência — Dimensione por um modelo de processo

IX. Descartabilidade — Maximizar a robustez com inicialização e desligamento rápido

X. Dev/prod semelhantes- Mantenha o desenvolvimento, teste, produção o mais semelhante possível

XI. Logs — Trate logs como fluxo de eventos

XII. Processos de Admin — Executar tarefas de administração/gerenciamento como processos pontuais

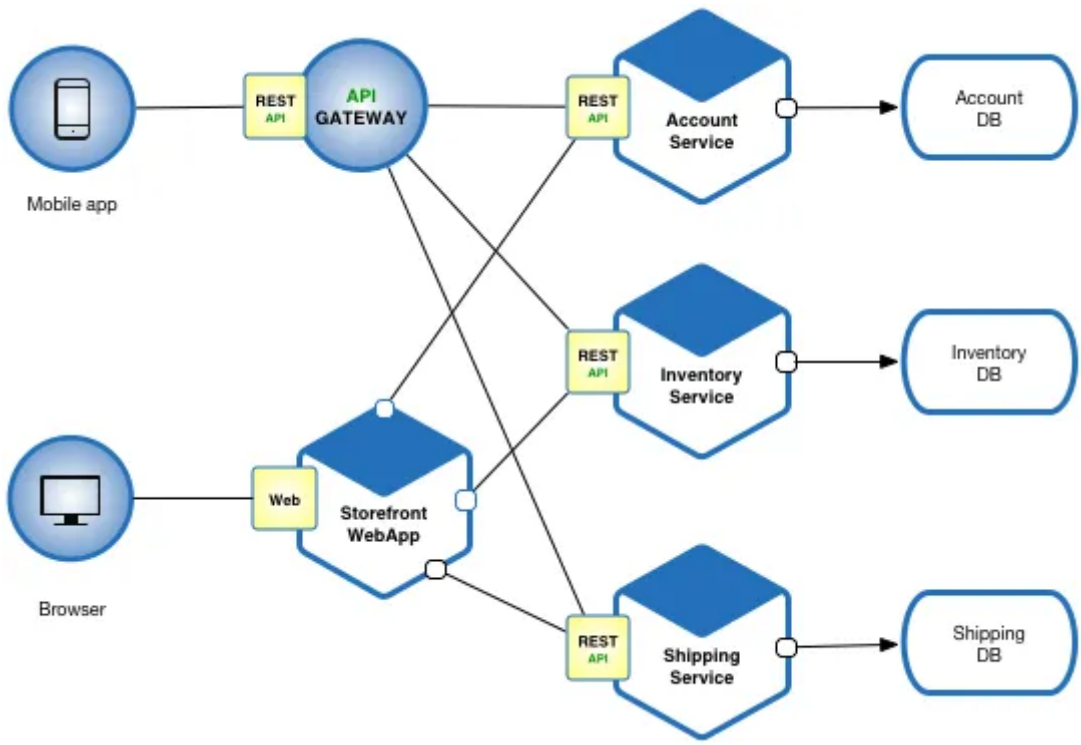
Conheça o teorema CAP, antes de começar

Em resumo, o teorema CAP diz que há três requisitos que você vai desejar, mas apenas dois será possível conseguir, e você terá de abrir mão de um deles, para conseguir os outros dois. Consistência, Disponibilidade e tolerância a falhas são estes três requisitos.

Em um sistema distribuído, acabamos assumindo que a consistência pode ser eventual, sendo assim, queremos alta disponibilidade e alta tolerância a falhas mesmo que eventualmente a informação não esteja consistente.

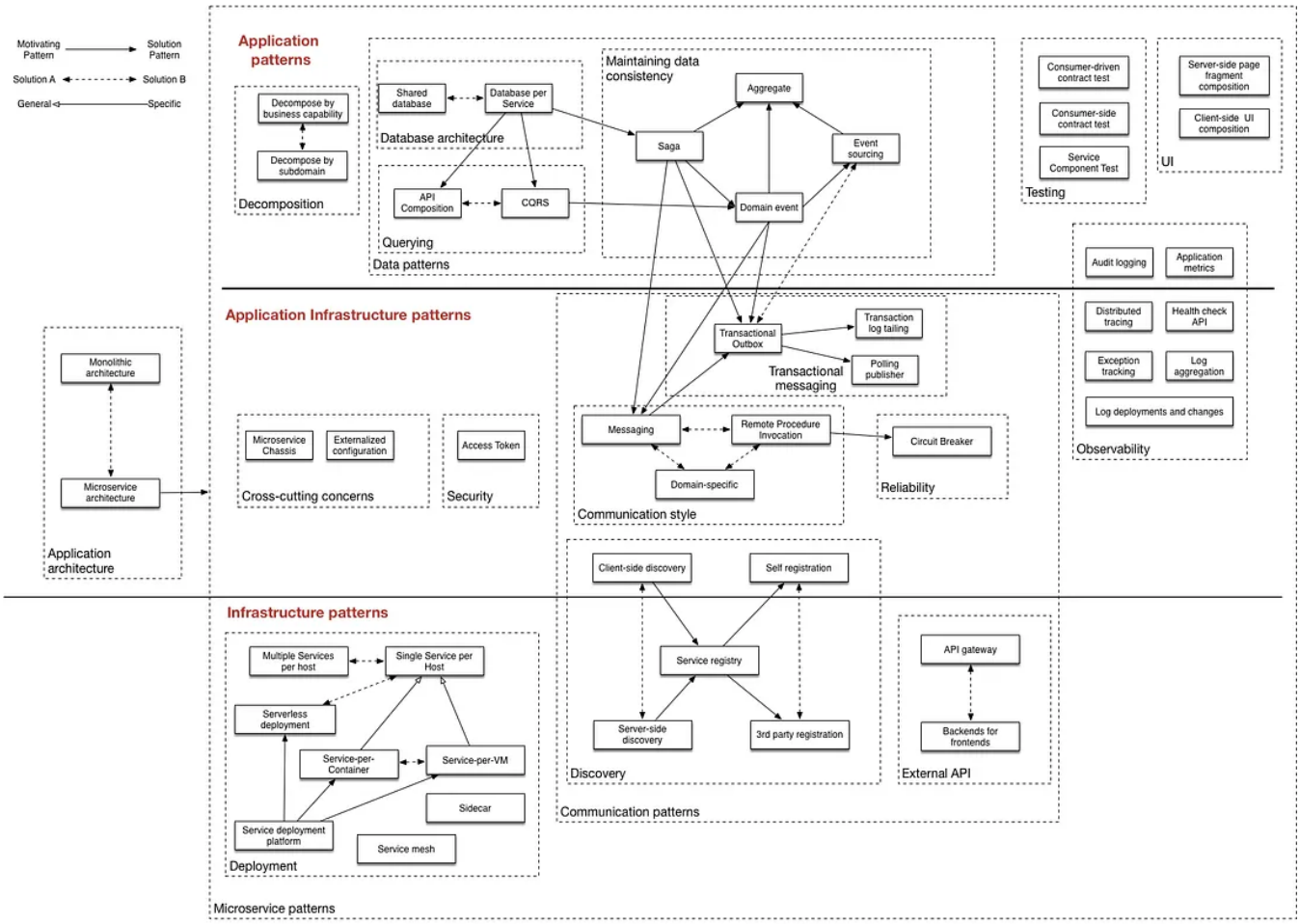
Este teorema, afeta diretamente a forma como você vai armazenar seus dados, e como cada serviço irá se comunicar entre eles. Conheça-o antes de começar sua jornada em direção a uma arquitetura de sistemas que utiliza micro serviços.

Abaixo vou deixar algumas imagens, que ilustram o tamanho do desafio que é trabalhar com micro-serviços, já entenda que micro-serviços é um jogo que se joga no nível “asiático” :D, tudo é muito mais difícil de se fazer, manter, depurar.



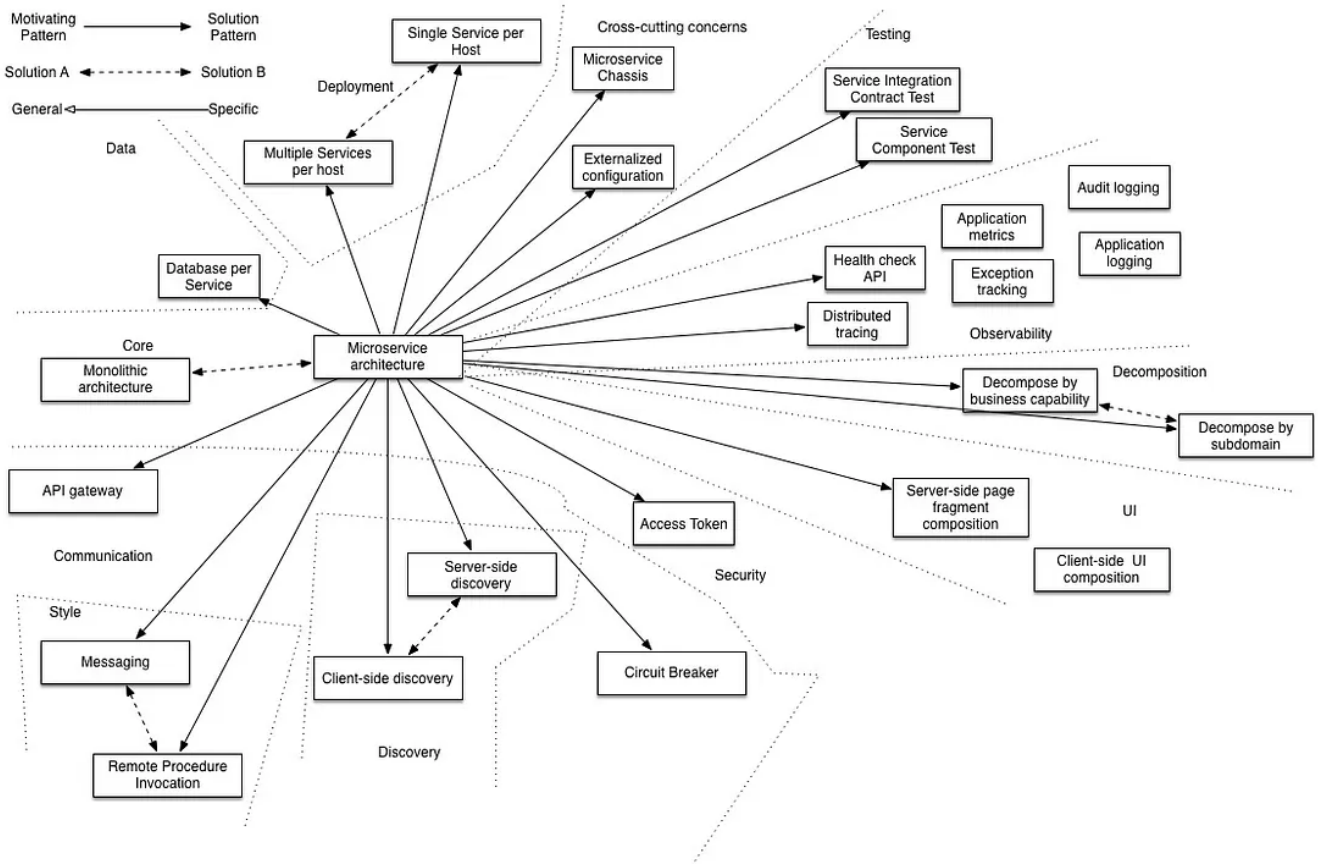
https://microservices.io/patterns/microservices.html

Nos próximos textos, vamos continuar falando sobre padrões em arquitetura de micro serviços, não necessariamente padrão arquitetural ou de código, mas padrões e modelos no geral, as imagens abaixo ilustram um pouco do que poderíamos abordar.



Copyright © 2019. Chris Richardson Consulting, Inc. All rights reserved. http://microservices.io/

https://microservices.io/patterns/index.html



https://microservices.io/patterns/microservices.html

462

13

Microservices Software Architecture Software Development Docker

Sign up for Codigo refinado

By codigorefinado

Receba atualizações do blog Take a look.

Your email

We couldn't process your request. Try again, or contact our support team.