

Arquitetura Funcional em Microservices

4 anos depois

Lucas Cavalcanti
@lucascs



nu



nu

4 anos atrás: Era tudo mato

Contexto inicial

- 5 engenheiros
- Greenfield: Datomic, Clojure, Kafka
- Ninguém sabia programar em Clojure antes de entrar
- Nenhum guideline consolidado de arquitetura em Clojure ou Funcional

Escolhas

- Ir o mais rápido possível, colocar no ar, testar, e se der certo refatorar pra ficar direito, pivotar se não
- OU
- Investir na infraestrutura de entrega contínua e já começar com tudo automatizado, usando as melhores práticas

Monolito ou Microservices?

Porquê?

- auth, accounts, customers, acquisition, geo
- requisitos de dados/acesso muito diferentes
- Negócio complexo, poder alterar pequenas partes
- grande carga de automação, grande carga de código de infraestrutura

Continuous Delivery

Pré requisitos

- Automação de build até prod
- Automação de infraestrutura AWS
- Bateria de Testes (unit, integration, e2e)
- CI Server (TW Go)
- Zero downtime deploy (Blue-Green deployment)
- Deploy pra prod automático/um clique

- >1000 pessoas!
- >140 na engenharia
- >150 serviços Clojure
- > 30 times
- > 4 anos de idade
- > 3 Mi clientes
- > 15 Mi pedidos



É mais fácil adicionar código hoje do
que era há 3 anos!



SÃO PAULO, BRASIL

Agenda

Imutabilidade

Funções Puras

Schemas

Componentes

Ports and Adapters - Arquitetura hexagonal

Microserviços

Imutabilidade

SOUTHEAST BRAZIL REGION FROM SPACE

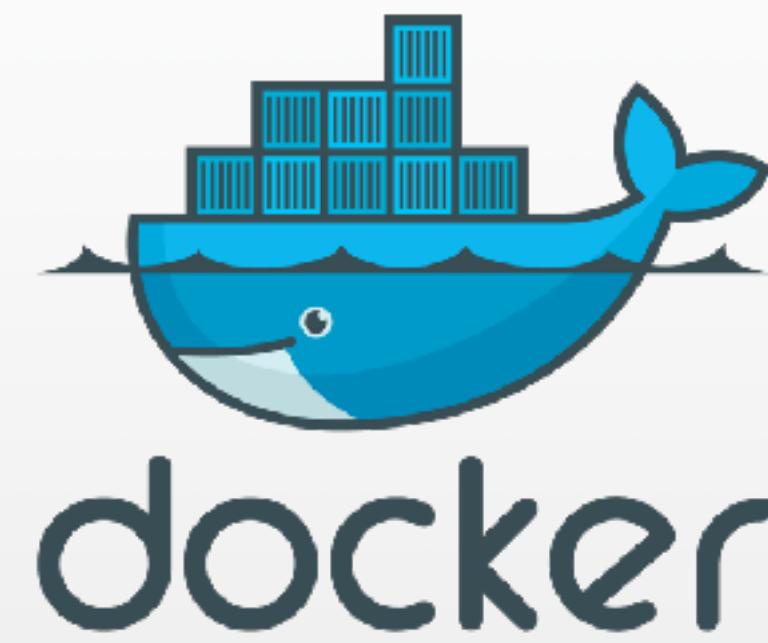
Imutabilidade

Definição

“Se eu recebo um valor, é garantido que ele nunca vai mudar”

Tecnologias escolhidas

Imutabilidade





Todas as estruturas de dados padrão são imutáveis:

- Maps, Lists, Sets
- Records

Mutabilidade é explícita:

- atoms/refs: @
- dynamic vars: * ... *
- chamadas java: .nomeDoMetodo
- funções com side effect: funcao!



Datomic guarda as mudanças/transações, não só os dados

- parecido com o Git
- append only
- db como valor
- tudo é salvo no banco como dados (transações, schemas, entidades)

Kafka

Immutabilidade



Filas e tópicos persistentes

- cada consumer tem um offset da sua última mensagem
- possibilidade de fazer replay de mensagens

AWS + Docker

Immutabilidade



- Cada build gera uma imagem Docker
- Cada deploy sobe uma nova máquina com a nova versão.
- Assim que a nova máquina está saudável, a máquina antiga é morta. (blue-green deployment)
- Mudanças de arquitetura geral são feitas criando uma nova stack de todos os serviços, e depois fazendo a troca do DNS

Funções Puras

SOUTHEAST BRAZIL REGION FROM SPACE

Funções puras

Definição

“Dadas as mesmas entradas, sempre será produzida a mesma saída”

Simplicidade

Funções Puras

- **mais fácil de entender** o que acontece, só é necessário olhar pras entradas
- **mais fácil de testar**, não precisa de mocks/preparação de contexto.
- **parallelizável por padrão**, sem necessidade de locks, sem preocupação de ter race conditions

Testes generativos

Funções Puras

- para **lógicas mais complexas** é difícil cobrir todos os casos com testes unitários
- gera **entradas aleatórias**, de acordo com os seus schemas
- verifica uma **propriedade da saída**

Funções impuras

Funções puras

- as funções que produzem **efeitos colaterais devem ser marcadas como tal**. Nós usamos `!` no final do nome das funções.
- separamos o código** que trata e transforma os dados do código que trata dos efeitos colaterais
- devem ser **movidas para as bordas do fluxo**, sempre que possível

Componentes

SOUTHEAST BRAZIL REGION FROM SPACE

Bibliotecas compartilhadas

Componentes

- em geral a **comunicação** do serviço **com o mundo exterior**, via http, kafka, banco de dados, arquivos, etc
- protocolos/interfaces** que podem ter **múltiplas implementações**, por exemplo mocks para testes

Padronização

Componentes

- mais fácil controlar a **comunicação** entre os serviços
- monitoramento e logging padronizados
- caixa de ferramentas** que facilitam o desenvolvimento

Dependências

Componentes

- sabem responder a mudanças de ambientes
- no **startup** do serviço eles são construídos e configurados, de acordo com suas dependências
- estão **disponíveis nas pontas** do sistema, e são passados pras funções como argumentos
- As **dependências** dos fluxos ficam **explícitas**

Schemas

SOUTHEAST BRAZIL REGION FROM SPACE

Tolerant Readers

Schemas

"Be conservative in what you do, be liberal in what you accept from others" -- Postel's Law

https://en.wikipedia.org/wiki/Jon_Postel#Postel.27s_Law

Tolerant Readers

Schemas

- **produtores** de um schema declaram e validam o **schema estrito** da mensagem produzida.
- **consumidores** de um schema declaram e validam **só os campos** do schema **relevantes** para o seu uso, ignorando os outros campos.

Formatos de comunicação: wire schemas

Schemas

- Wire schemas são **como você expõe os dados** para outros serviços/clientes
- esses schemas podem ter campos **diferentes dos dados das entidades**, assim conseguimos evoluir as entidades sem quebrar os clientes
- necessitam de uma **camada de adaptação/conversão**
- wire schemas são **sempre declarados e validados nas entradas/saídas** do serviço

Testes de comunicação

Schemas

- como os schemas estão declarados nas pontas, temos uma bateria de testes que **gera dados aleatórios** de acordo com o schema de saída e **valida contra o schema** de entrada
- assim a gente garante que a **comunicação entre dois serviços** é compatível
- pega bugs de **quebras de compatibilidade** antes de ir pra produção

Ports and Adapters (a.k.a Arquitetura Hexagonal)

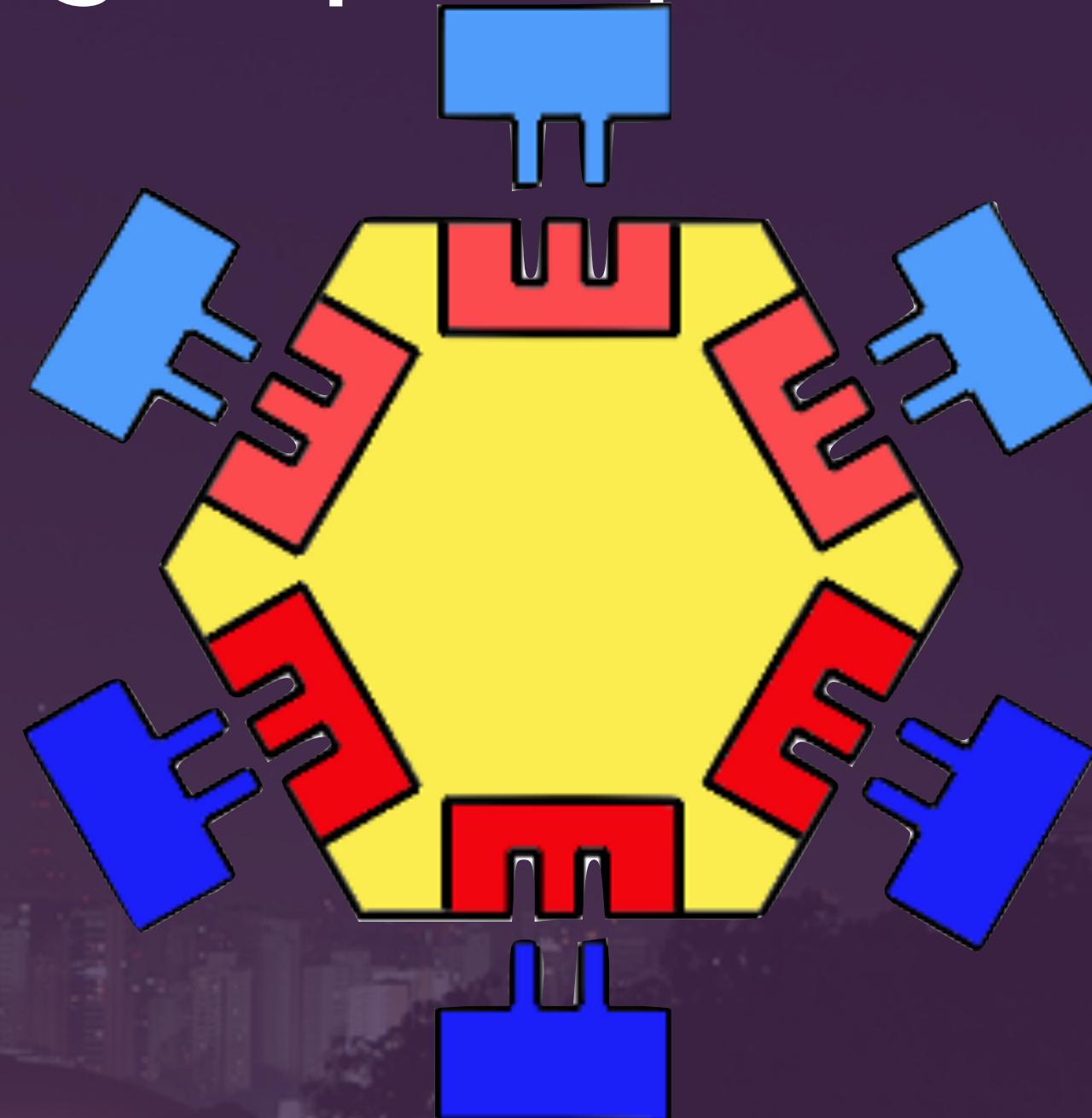
Ports and Adapters

Definição

A lógica principal é independente de onde ela foi executada (amarelo)

Uma porta é uma das entradas/saídas da aplicação (azul)

Um adaptador é a ponte entre a porta e a lógica principal (vermelho)



Ports and Adapters (versão Nubank)

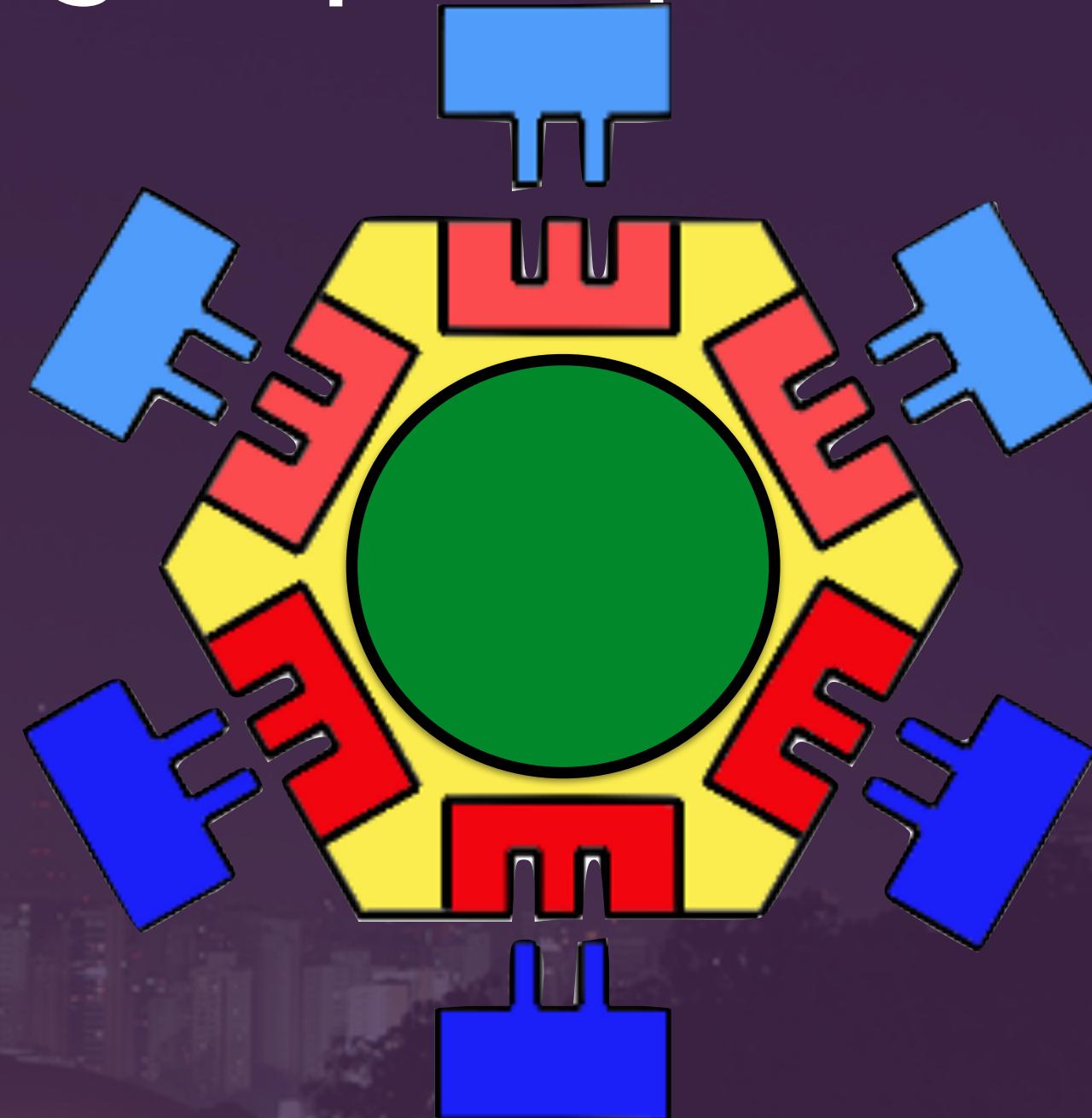
Definição estendida

Lógica de negócios pura (verde)

Controller, que liga os fluxos entre as portas (amarelo)

Uma porta é uma das entradas/saídas da aplicação (azul)

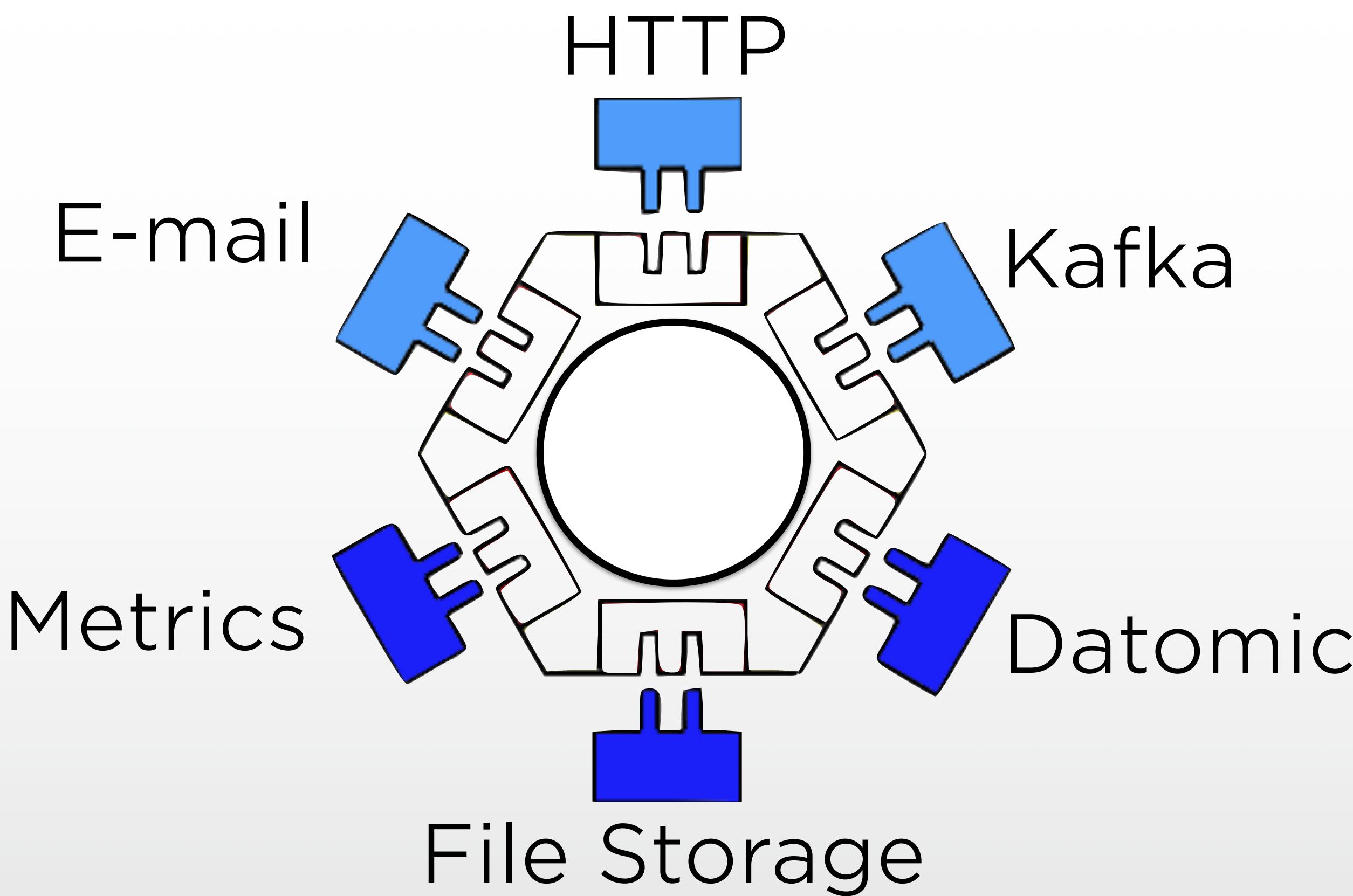
Um adaptador é a ponte entre a porta e a lógica principal
(vermelho)



Portas (Componentes)

Ports and Adapters

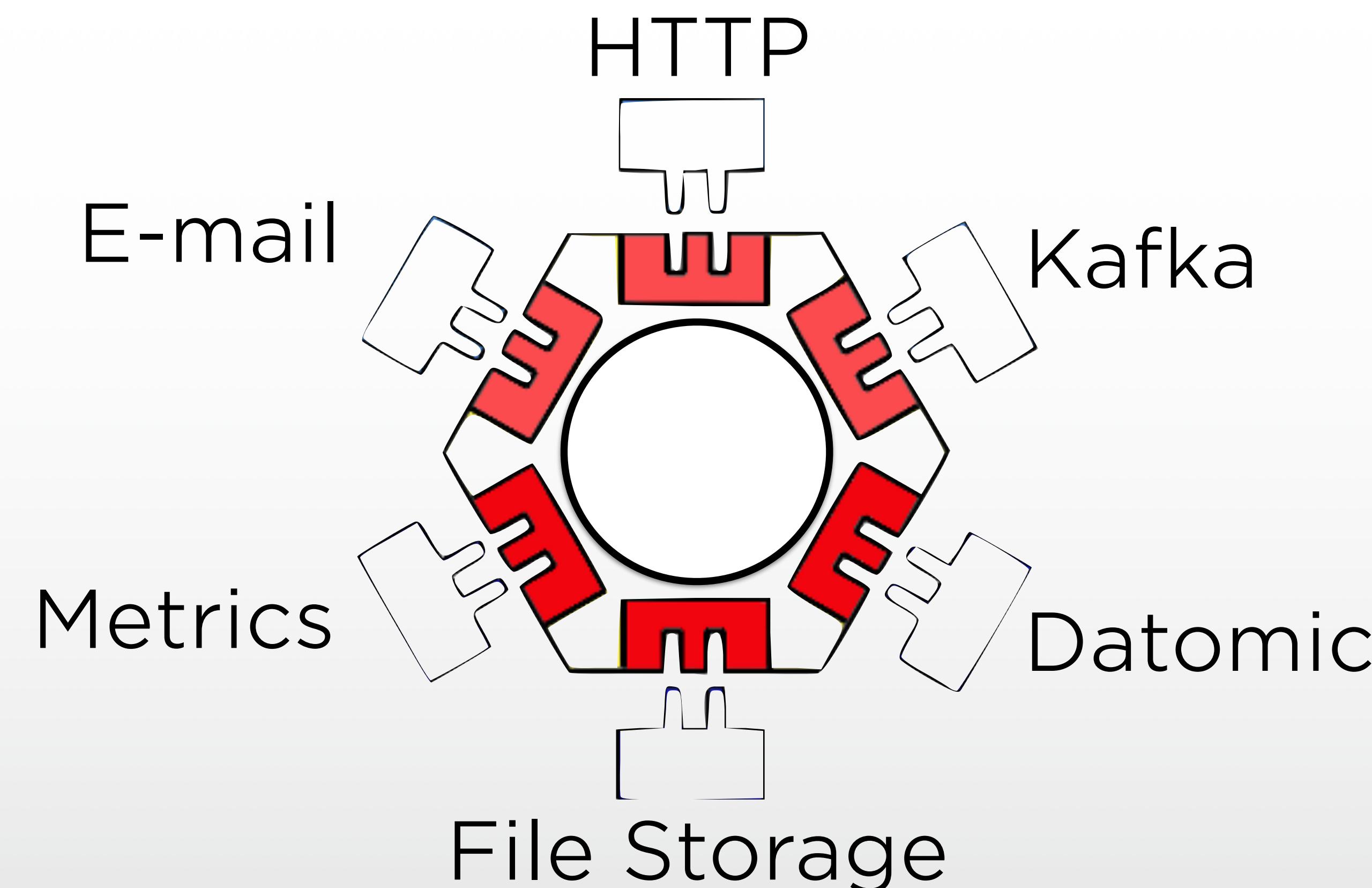
- Portas são inicializadas junto com o serviço
- Cada porta tem seu respectivo **componente**
- Serializa os dados para algum formato de transporte (ex. JSON, Transit, XML)
- Em geral vem de uma biblioteca compartilhada entre todos os serviços
- Testada via **testes** de integração, **ponta a ponta**



Adaptadores (Diplomat)

Ports and Adapters

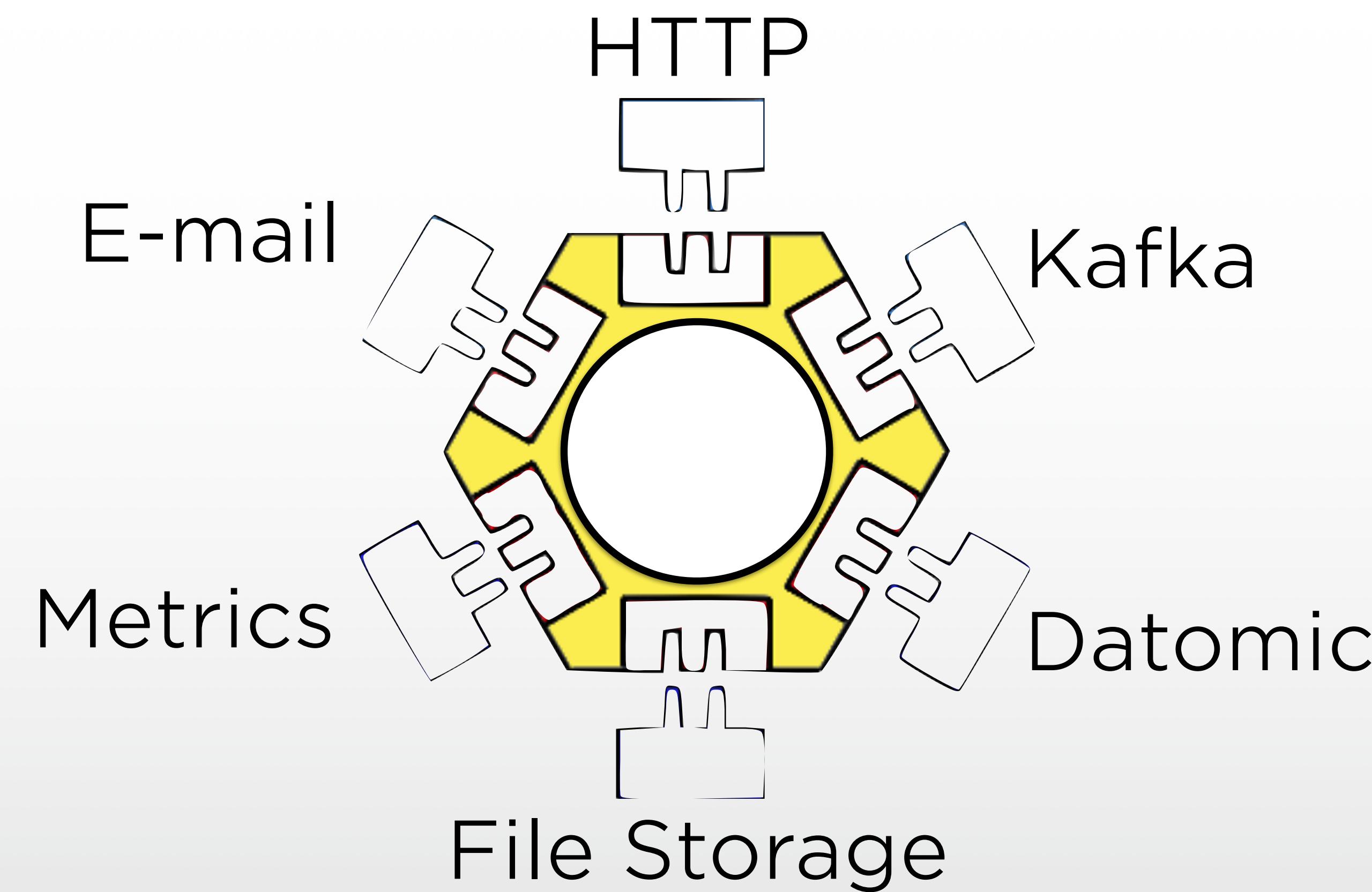
- Adaptadores são os conectores das portas
- Contém as funções que tratam as chamadas HTTP, mensagens do Kafka, etc
- Adaptam o wire schema para o schema interno**
- Chamam e são chamados pelas funções controllers
- Testada com** versões falsas dos componentes, ou **mocks**



Controllers

Ports and Adapters

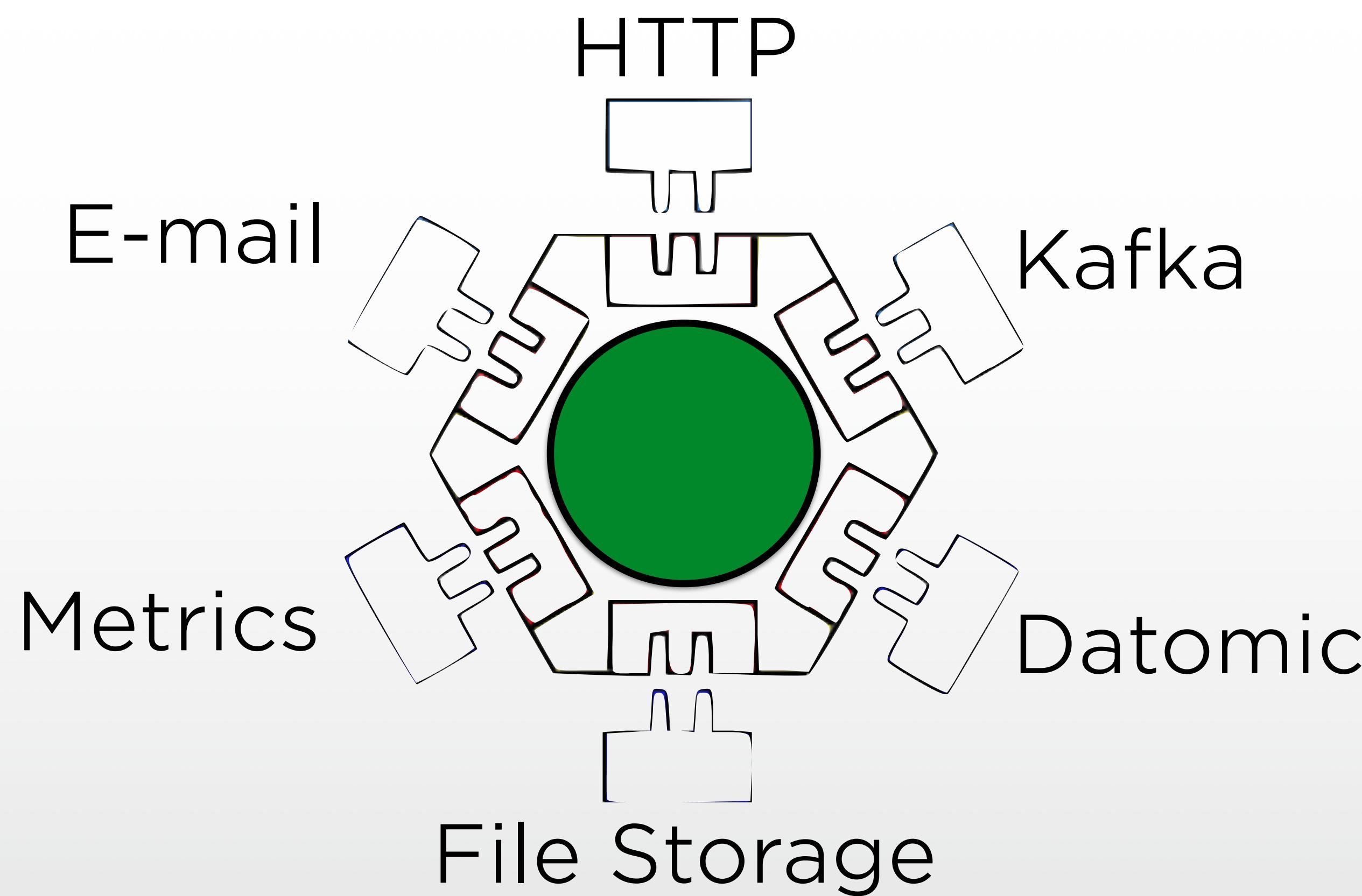
- Controllers ligam os fluxos entre as entradas e os seus side-effects
- Só lidam com schemas internos
- Delegam a lógica de negócio para funções puras
- Compõem os resultados dos efeitos colaterais**
- Geralmente testado com mocks ou via **testes de integração**



Lógica de Negócio

Ports and Adapters

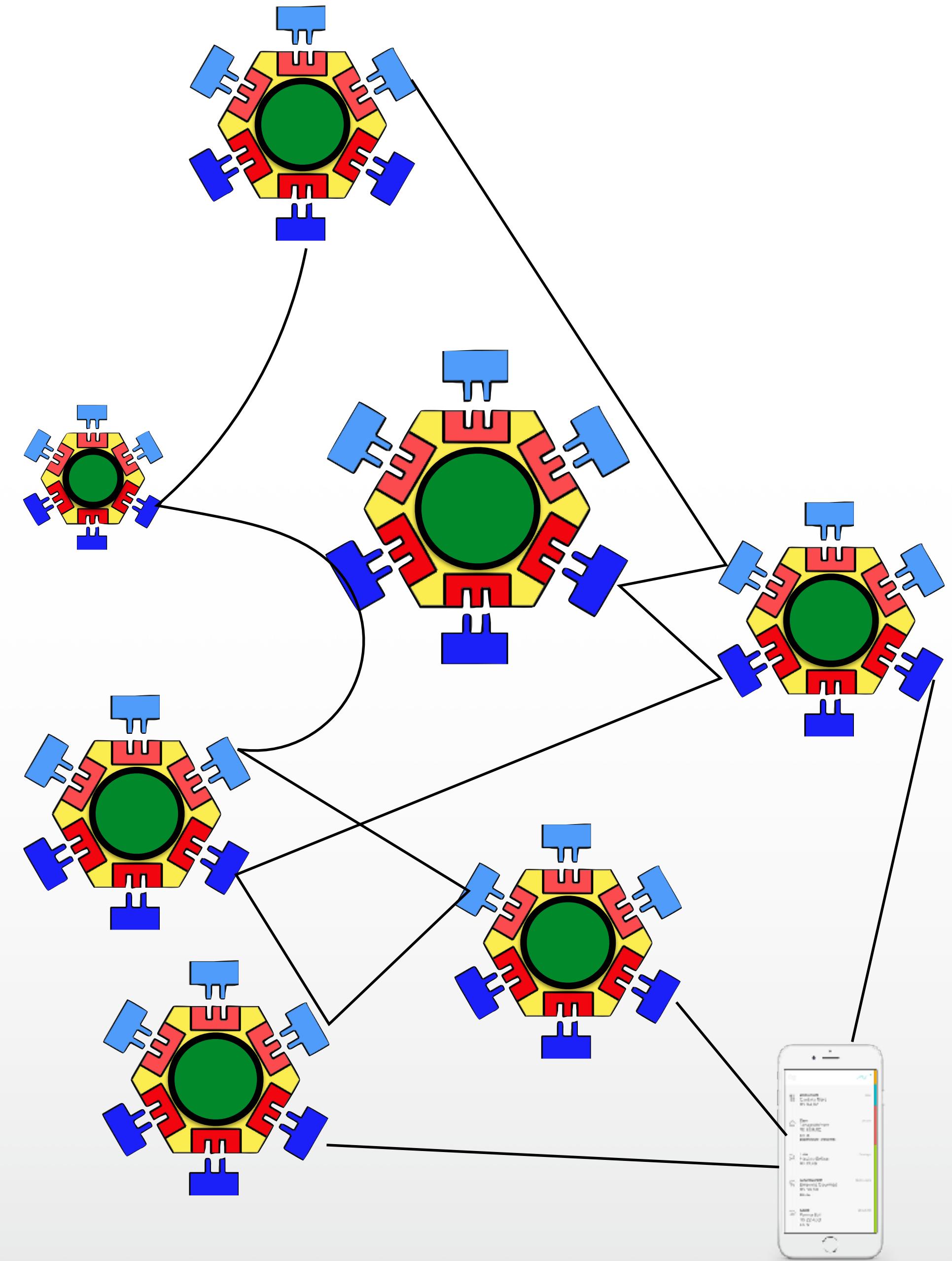
- Trata e transforma dados **imutáveis**
- **Funções puras**
- Melhor lugar para checar **invariante s e regras de negócio**
- Pode ser testado com testes generativos/propriedade
- Deveria ser a maior parte do código da aplicação
- Testada com **testes de unidade ou generativos**



Microserviços

Ports and Adapters

- Cada serviço segue mais ou menos o mesmo design
- Os serviços se comunicam via uma das portas (ex HTTP ou Kafka)
- Serviços **NÃO** compartilham bancos de dados
- HTTP responses contém hipermédia, então podemos substituir um serviço sem ter que mudar os clientes mobile
- Testados com **testes ponta a ponta**, com **todos os serviços**



Lições aprendidas

Microserviços

- homogeneidade
- scalability units / sharding
- anti corruption layer para lidar com terceiros
- property based testing pra lógicas complexas

Lições aprendidas

Microserviços

- tolerant readers vs central schemas
- e2e vs checar contratos
- na falta de documentação, testes de integração ajudam

Lições aprendidas

Microserviços

- squads cuidando de grupos de serviços
- documentação e onboarding cada vez mais importantes
- quebrar um serviço que ficou grande é bastante traumático, mas necessário
- domínio mais complicado, serviços menores
- carga de automatização foi bem maior do que o esperado
- monitoração e tolerância a falhas

Resumo

Imutabilidade em todos os níveis
Lógicas de negócio em **Funções Puras**
Schemas validados na comunicação
Componentes compartilhados
Arquitetura hexagonal padronizada
Microserviços controlando o escopo

sou.nu/vagasnu



Lucas Cavalcanti
@lucascs

Obrigado!

nú