Aula teórica - ciclo /backend
Projeto de software

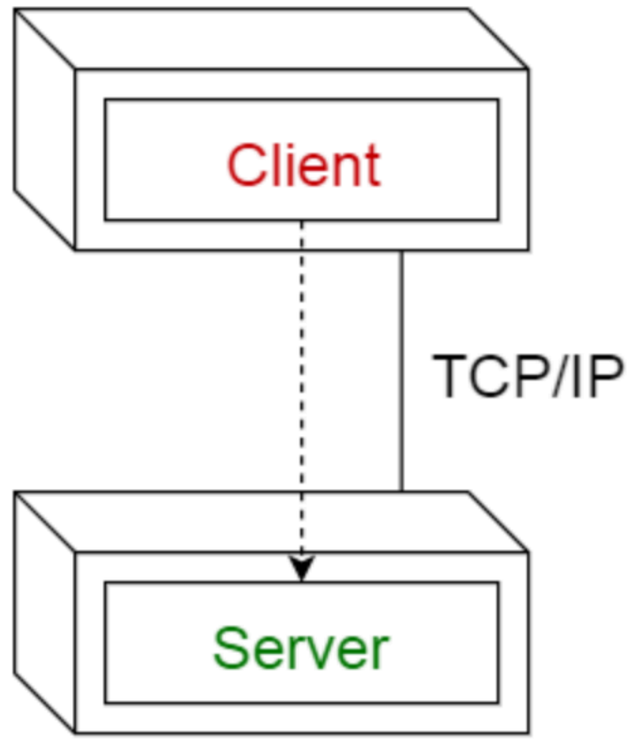
Raquel Lopes

Agenda

- Arquiteturas tradicionais de aplicações Web (cliente/servidor x P2P, monolíticas x microsserviços)
- API (definição, vantagens, frontend/backend/API)
- Framework, dependências
- MVC
- API REST - exemplo com spring boot

Cliente/servidor

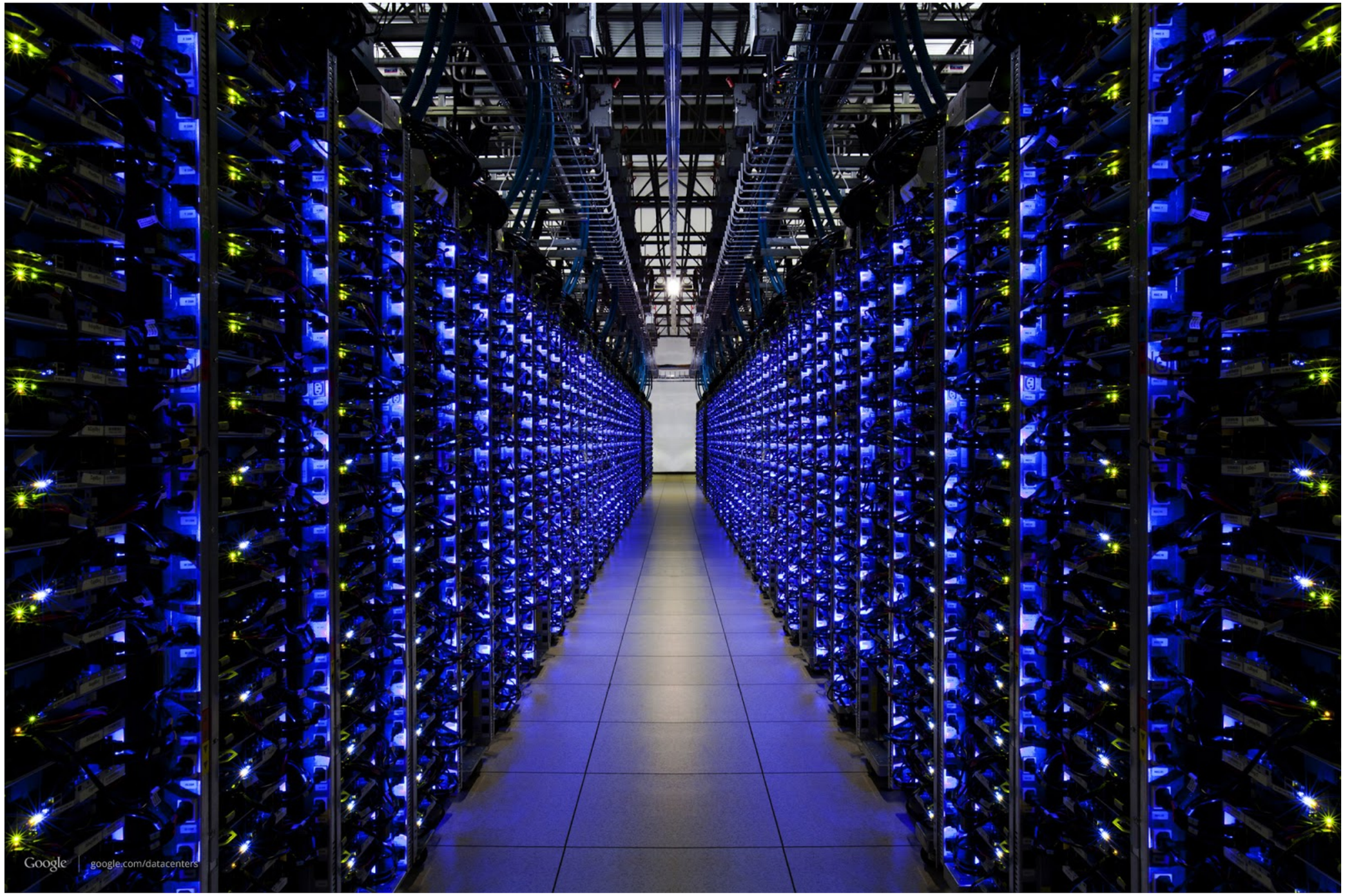
- O servidor oferece serviços para múltiplos clientes
  - Cliente universal: browser
- No servidor temos um serviço
  - Ouve uma porta e atende os pedidos
    - Web
  - Multithread, multiprocessadores, slaves, etc.



Client-server pattern

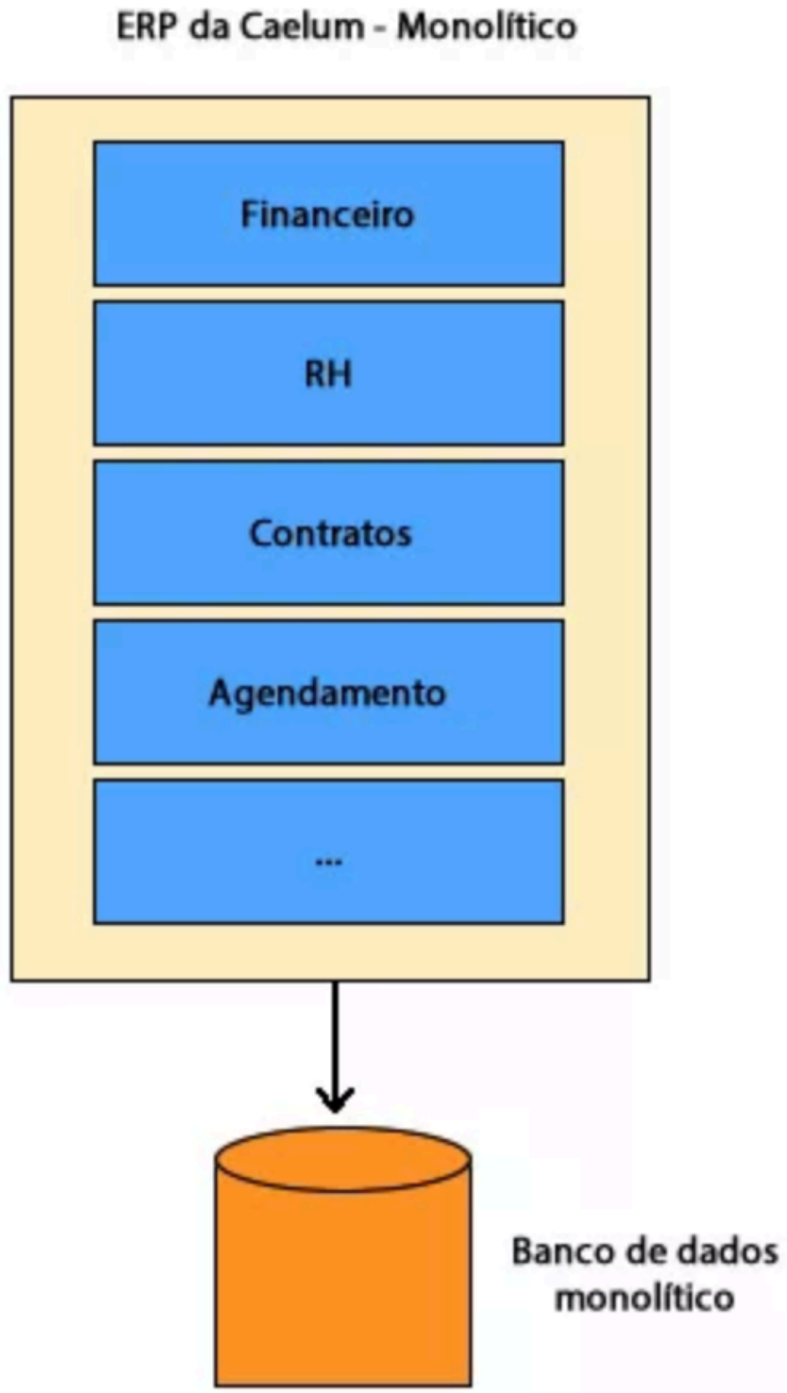
Arquitetura P2P - peer to peer

- Não há separação de cliente e servidor
- Cada nó da rede funciona tanto como cliente quanto como servidor
- Permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central



Onde ficam os servidores?

Nuvem?
Cluster privado?
Datacenter da própria empresa?



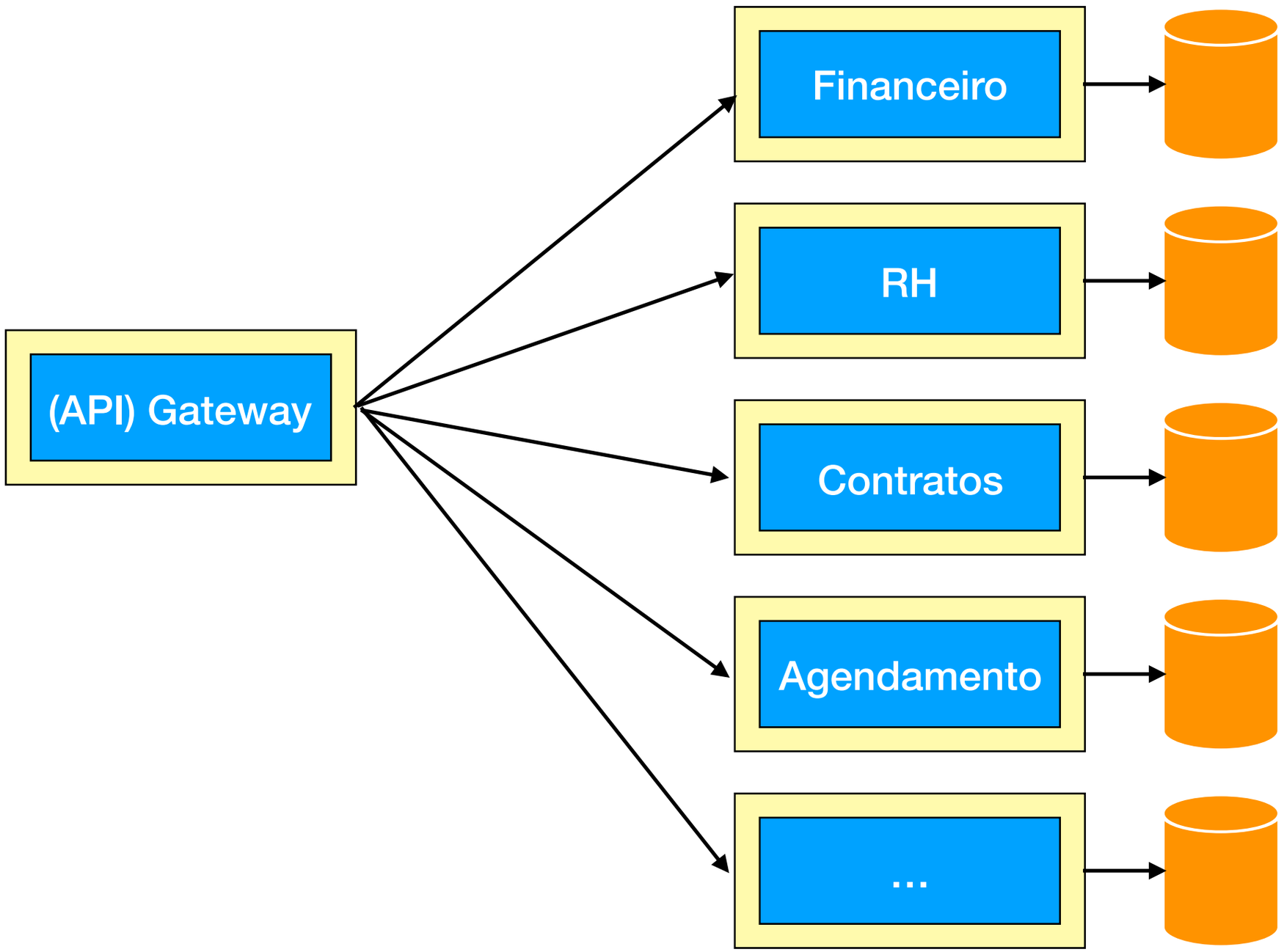
Um único sistema, com todos os módulos dentro dele

Arquitetura monolítica

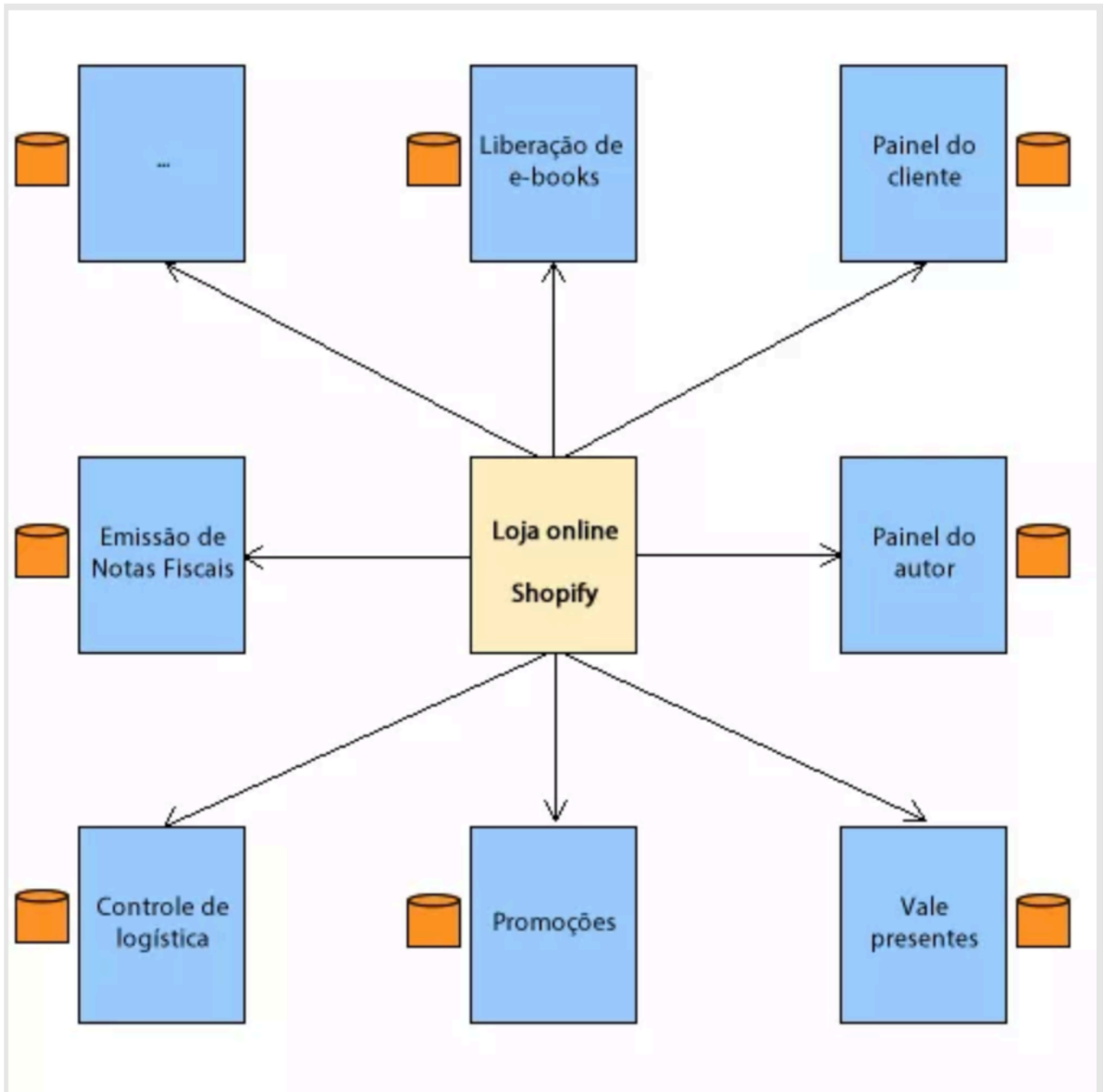
Na prática: vc tem um único jar
Tudo roda em um servidor

https://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica/

Arquitetura microserviços



Arquitetura de microserviços - Editora casa do código



Vários sistemas que são notificados por outro via HTTP quando um determinado evento ocorre. Cada sistema decide o que fazer com o JSON que é enviado para ele

Distribuição

Monolítico vs Microserviços

Vantagens e desvantagens!

- Ponto único de falha

- Base de código extensa/ complexa

- Escolher uma tecnologia

+ Mais simples de implantar

+ Não há duplicidade de código

+ Falha de serviços independentes

+ Sistemas com código base menor e mais simples

+ Permite trabalhar com diferentes tecnologias

- implantação distribuída

- Repetição de código e de dados

Como decidir?

- Experiência do arquiteto de software
- Pesar vantagens e desvantagens com características da aplicação
- Dica: começar com monolítico e migrar microserviços que façam sentindo depois
    - Pode até depender do início de outro projeto (semelhante)
- Não existe solução perfeita

Vamos focar no mundo monolítico

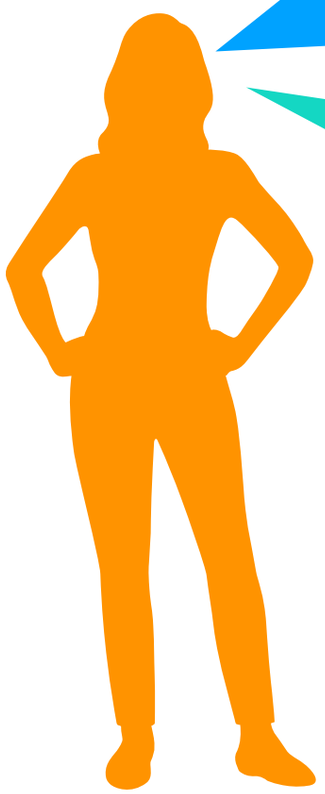
Primeiro contato, muito comum, base para micro-serviços

Agenda

- Arquiteturas tradicionais de aplicações Web (cliente/servidor x P2P, monolíticas x microsserviços)
- API (definição, vantagens, frontend/backend/API)
- Framework, dependências
- Padrão MVC
- API REST - exemplo com spring boot

API

uma API é uma coleção de métodos claramente definidos para comunicação entre diferentes componentes de um software



uma API Web é a interface criada pelo back-end: a coleção de endpoints e recursos que esses endpoints expoem

API e backend

- Application Programming Interface - Interface de Programação de uma aplicação
    - O cliente de é sempre outro programa
- O backend de uma aplicação é “publicado" na forma de uma ou mais APIs
- APIs são invisíveis
- APIs podem ser clientes de outras APIs
- De forma prática, o que seria uma API?

Backend

- É a parte da aplicação que não conseguimos ver
- É o cérebro da aplicação (código que processa os dados) e sua memória (os dados, em geral em um banco de dados)
- Responsável por conhecer as regras de negócio
- Pode expor APIs (serviços)
- Persistência dos dados, segurança da informação, performance, etc.

Backend (Cont.)

- Muito mais variação do que o frontend (linguagens, tecnologias)
  - Ex. Python (Django), Ruby (Rails), JavaScript (Node.js), Java (Spring)
- Pilha: servidor/OS, framework, aplicação, banco de dados
- **APIs conectam software, aplicações, bancos de dados, e serviços juntos de forma transparente**
  - **Conceito chave importante!**

APIs em nossa vida

- Quando usamos conta google ou facebook para nos logar em outro serviço
- Compra online (operadoras de crédito)
- Conseguir dados de redes sociais através de uma aplicação. É possível?
- Reserva de hotel facilitada

Backend ≋ API

- API oferece um serviço (processo ouve uma porta)
- O que acontece quando chega uma requisição?
- Leitura/processamento da requisição e dados de entrada
- Usar dados recebidos pra **identificar e realizar a ação solicitada**
    - Autenticação, autorização, acesso a bancos de dados, logging/exceções, (acesso a outras APIs), etc.
- Compor resposta, enviar resposta

Agenda

- Arquiteturas tradicionais de aplicações Web (cliente/servidor x P2P, monolíticas x microsserviços)
- API (definição, vantagens, frontend/backend/API)
- Framework, dependências
- Padrão MVC
- API REST - exemplo com spring boot

Framework de desenvolvimento

- Oferece serviços para **aumentar a produtividade**
- O programador perde o controle sobre o que realmente está acontecendo
- Melhora com o uso do framework (tentar entender os serviços aos poucos - não fazer por fazer, na tentativa e erro)
- Tem que **aprender a usar o framework** e o que ele oferece
- Vamos usar maven para controle de dependências

Framework de desenvolvimento

- Camada funcional de sua aplicação
- Basicamente gerencia qualquer comunicação necessária
  - Entre frontend/backend, dentro do backend (acesso aos dados)
- Controla o fluxo da aplicação para você
- Oferece muitos serviços (autenticação/autorização, segurança, facilita acesso aos dados, manipulação de erros)
- Permite que os desenvolvedores se concentrem apenas nas regras de negócio

Hollywood principle

- Inversão de Controle é uma parte fundamental do que torna um framework diferente de uma biblioteca
- Biblioteca: conjunto de funções que você pode chamar, geralmente organizadas em classes
- Framework: incorpora algum design abstrato, com comportamento embutido. Para usá-lo, você precisa inserir seu comportamento em vários lugares no framework. O código da estrutura chama seu código nesses pontos
  - Template method, **plugin**/separated interface

Dependências

- Você é estilista de roupas e calçados
- Se você quer vender suas criações (mas você mesmo só sabe desenhar - que é a parte criativa, talvez a mais importante)
- O que você faz?
- E se você tiver pressa de entrar no mercado?

- Software ad apache
- Gerencia dependências
    - Faz download de bibliotecas e configura o class path pra gente
- pom.xml

Agenda

- Arquiteturas tradicionais de aplicações Web (cliente/servidor x P2P, monolíticas x microsserviços)
- API (definição, vantagens, frontend/backend/API)
- Framework, dependências
- Padrão MVC
- API REST - exemplo com spring boot

Padrões arquiteturais

- Semelhante a padrões de projeto, mas para arquiteturas
- Solução geral aplicável a um contexto específico
- Alguns padrões muito comuns
    - Isso é ortogonal à aplicação ser monolítica ou de microserviços
- Alguns padrões relacionados à comunicação entre os componentes da aplicação serão vistos depois

Agenda

- Arquiteturas tradicionais de aplicações Web (cliente/servidor x P2P, monolíticas x microsserviços)
- API (definição, vantagens, frontend/backend/API)
- Framework, dependências
- Padrão MVC
- API REST - exemplo com spring boot

Padrão MVC

- **M**odel-**V**iew-**C**ontroller

- O que é o modelo? É quem **sabe fazer o que precisa ser feito** (backend - regras de negócio e dados)

- O que é a visão? **Apresenta informações** para o usuário (frontend) - muitas views são possíveis

- O que é o controlador? É quem **roteia os pedidos dos usuários** para quem sabe fazer

- Adotado pela maioria das aplicações Web

Representational State Transfer (REST)

- Conjunto de princípios que beneficiam a aplicação com a arquitetura e **padrões da própria Web**
- Cliente/servidor
    - **HTTP** para comunicação
- API baseada em **URIs** (Uniform Resource Identifier)

A URI une o Protocolo (https://) a localização do recurso (exemplo.com.br:8080) e o nome do recurso (/parmegianas/) para que você acesse a API

- As respostas vem em geral em um formato de dados específico chamado **JSON** (ou html ou xml)

REST - Cont.

- Interação cliente/servidor é **stateless** (sem estado)
  - Cada requisição traz a informação necessária para entendê-la
  - O estado das sessões fica sendo mantido apenas nos clientes
  - **Bom** para escalabilidade do servidor
  - **Ruim** porque usa mais rede
  - Pode usar cache no cliente -> requisições "cacheáveis" e não "cacheáveis"

Recursos REST

- Um **recurso** é uma abstração sobre um determinado tipo de informação que a API gerencia
    - e-commerce: produtos, clientes, vendas…
- Em REST todo recurso deve possuir uma identificação única
    - Para diferenciar qual dos recursos deve ser manipulado em uma determinada solicitação

Design da API

- O design consiste em pensar nos recursos que serão expostos, seus parâmetros e suas respostas
  - Restaurante italiano de entrega parmegiana (pode ser de filé, frango ou peixe), tamanhos variados
  - /parmegiana-file-grande, /parmegiana-file-media, … / parmegiana-peixe-pequena OU
  - Um recurso /parmegiana e informações no corpo da mensagem?

REST - boas práticas

- Use URIs legíveis, de fácil dedução (por humanos). Por que?
- Utilize o mesmo padrão de URI na identificação dos recursos. Por que?
- **Evite** URI's que contenham a operação a ser realizada em um recurso (/produto/excluir, /produto/adicionar)
- **Evite** adicionar na URI o formato desejado da representação do recurso. Por que?
- Evite alterações nas URI's. Por que?

Uso dos métodos HTTP

Exemplo de boa prática para manipular o recurso /clientes

Método	URI	Utilização
GET	/clientes	Recuperar os dados de todos os clientes.
GET	/clientes/id	Recuperar os dados de um determinado cliente.
POST	/clientes	Criar um novo cliente.
PUT	/clientes/id	Atualizar os dados de um determinado cliente.
DELETE	/clientes/id	Excluir um determinado cliente.

https://blog.caelum.com.br/rest-principios-e-boas-praticas/

Resposta para cliente

- Também faz parte do design

Família	Descrição
2xx	Em geral indica sucesso (código 200 - OK, 201 - Created)
4xx	Indica que ocorreu um erro em geral que a requisição que chegou não está bem formada. É um erro de quem gerou a requisição.
5xx	Indica que ocorreu um erro interno no servidor ao processar a requisição, é um erro no backend.

- Linguagem de programação Java

- (micro)framework para backend java - springboot (com maven)

- REST API

Tarefa de casa

- https://raquelvl.github.io/projsw.github.io/material/back_springMVC.html

- Em casa siga esse material para criar uma API REST bem simples