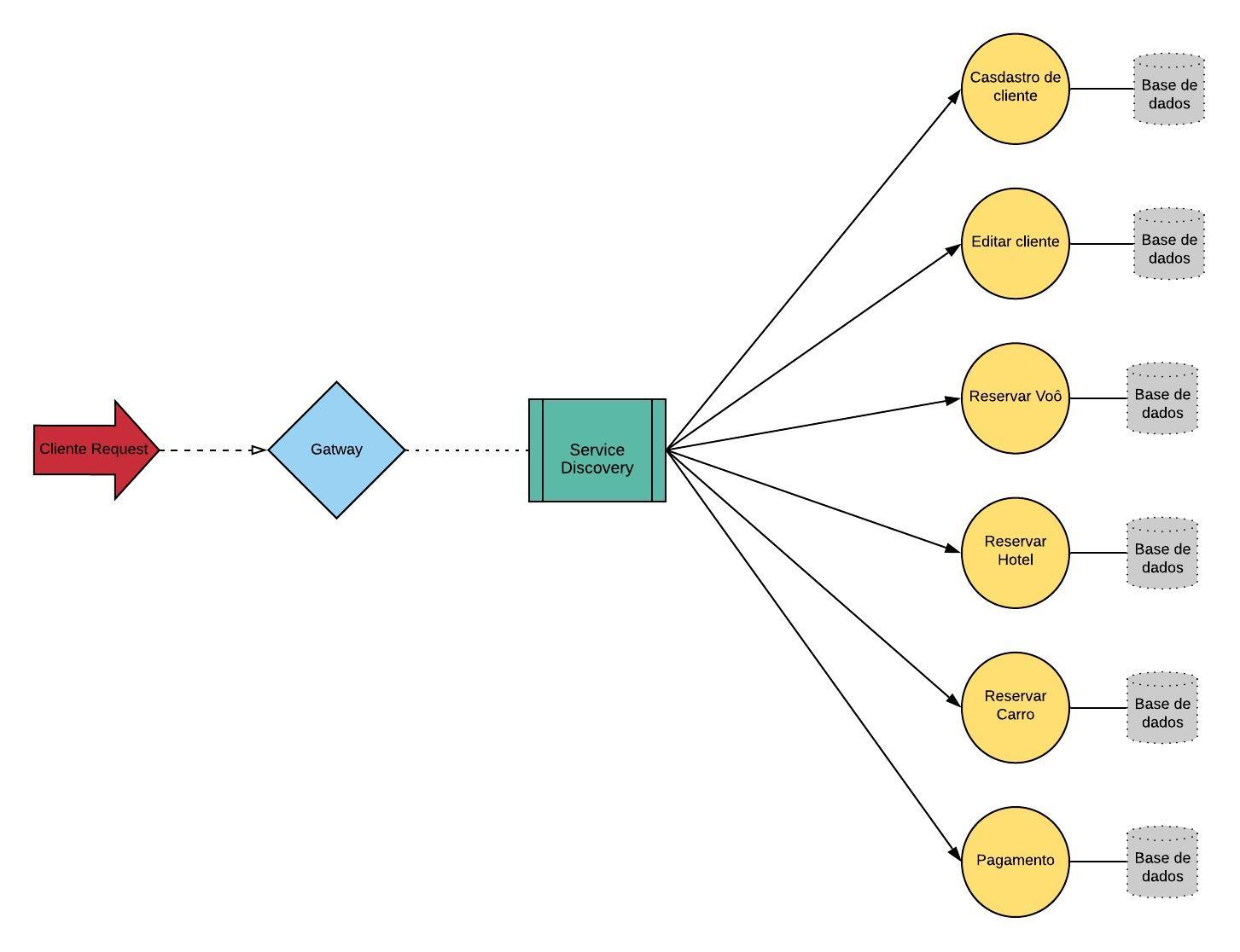
Aplicação baseada em Microserviços

Micro serviços é uma maneira particular de desenvolver aplicações de maneira que cada módulo do software é um serviço standalone cujo deploy e escala acontecem de maneira independentes da “aplicação principal” ([não confundir com SOA](https://www.luiztools.com.br/post/microservices-vs-soa-entenda-as-diferencas)). Enquanto na arquitetura tradicional de software, chamada monolítica, quebramos uma grande aplicação em bibliotecas, cujos objetos são utilizados in-process, em uma aplicação modular como proposta na arquitetura de microservices cada módulo recebe requisições

Diagrama da Solução



Padrões arquiteturais e de Projetos (design patterns) utilizados

Microserviços de forma geral, esta arquitetura compartilha de alguns benefícios:

* Os componentes têm baixo acoplamento. Podem ser desenvolvidos, testados, “deployados” e escalados de forma independente
* Os componentes podem ser desenvolvidos cada um com um stack de tecnologias própria
* Eles geralmente implementam funcionalidades avançadas e padrões como o service discovery, circuit breaking, load balancing etc.
* São mais leves e com funcionalidades especificas. Um serviço de autenticação, por exemplo, só sabe fazer autenticação.
* Geralmente tem um longo e extenso setup de monitoramento e troubleshooting

As **vantagens** deste tipo de arquitetura podem ser listadas como:

* Baixo acoplamento e melhor isolamento, mais fácil de testar e mais rápido para inicializar a aplicação
* Interações de desenvolvimento mais rápidas. Novos recursos podem ser desenvolvidos mais rapidamente e os existentes facilmente refatorados
* Problemas como memory leak em um dos serviços estão isolados e não vão derrubar a aplicação toda
* A adoção de novas tecnologias é mais simples, os componentes podem ser atualizados de forma independente e de forma incremental, possibilitando um stack com diferentes tecnologias e linguagens
* Os serviços serão inicializados de forma mais rápida e possivelmente de forma paralela
* Os times serão independentes. Encaixa direitinho em squads e times ágeis

Certo, nem tudo são flores! Vamos as **desvantagens** deste tipo de arquitetura:

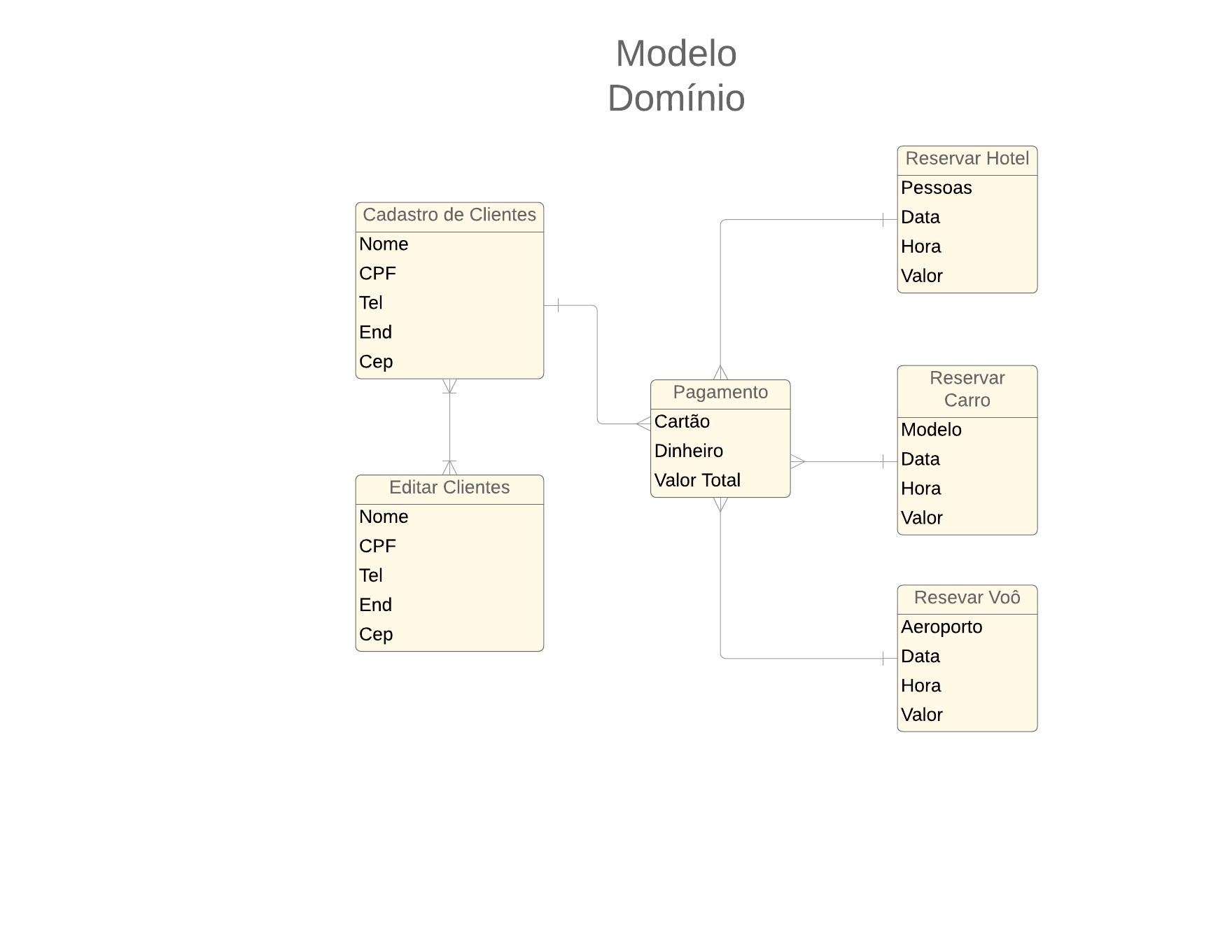
* Em termos gerais a aplicação é mais complexa
* Os testes de end-to-end e integração são complexos e custosos, pois existem mais partes móveis dentro da aplicação
* A aplicação toda é mais complexa de efetuar o “deploy” pois geralmente envolvem containers e virtualização
* Escalar é mais fácil, porém definir as regras de upscaling são mais complexas e necessitam de funcionalidades avançadas como o Service Discovery, DNS Routing e por ai vai…
* Geralmente precisa de um time maior para manter o software
* O skill set da equipe se torna mais variado, dificultando o compartilhamento de conhecimento e deixando reposição de recursos mais complicada
* O stack de tecnologias é mais complexo e é mais difícil de aprender
* O desenvolvimento inicial é mais lento, atrasando o time to market
* Precisa de uma infraestrutura complexa. Docker, multiplas JVMs e app containers

Características da solução tais como: Acoplamento, , Comunicação

entre os serviços, Monitoramento

Baixo Acoplamento, o acoplamento se refere a uma medida de dependência.

Um baixo acoplamento significa que implementações específicas de um Serviço podem ser substituídas, modificadas e evoluídas sem que os consumidores deste Serviço sintam qualquer descontinuidade. Fica claro que Serviços não devem expressar a lógica (regras) de negócio



Fluxos de comunicação simples usando dois protocolos o HTTP usando APIs e um sistema de mensagens.

Plataformas e frameworks utilizados

Foram criados 6 Microservises usando Framework Spring boot

O Spring Boot é uma ferramenta que visa facilitar o processo de configuração e publicação de aplicações que utilizem o ecossistema Spring.

O Spring Boot fornece a maioria dos componentes baseados no Spring necessários em aplicações em geral de maneira pré-configurada, tornando possível termos uma aplicação rodando em produção rapidamente com o esforço mínimo de configuração e implantação.

Posteriormente foram registrados os serviços no Eureka Discovery

Em uma aplicação moderna que utiliza a arquitetura de microserviços, geralmente a comunicação entre os serviços é feita através de APIs REST.  Neste modelo, também é comum a aplicação ser disponibilizada em nuvem, o que permite a criação de instâncias dinamicamente levando em consideração novos *deploys*, escalabilidade e possíveis falhas.

Nesse cenário, o conceito de Service Discovery consiste em obter de maneira dinâmica os endereços de IP dos serviços que iremos consumir.

Para que isso ocorra haverá comunicação entre três componentes:

* **Service Provider –** Fornecerá o serviço e está sujeito a mudança de endereço.
* **Service Consumer –** Consumirá o serviço e precisa conhecer o endereço que será chamado.
* **Service Registry –**Mantém o endereço atualizado de todas as instâncias dos serviços.

Para testar os serviços foi utilizado o Swagger

**Swagger**é um framework para descrição, consumo e visualização de serviços RESTful. E seu grande objetivo é permitir que a documentação possa evoluir no mesmo ritmo da implementação, já que pode ser gerada automaticamente com base em anotações do código.

O Banco utilizado foi o H2 que sob a memória quando a aplicação sobe

H2 é um mecanismo de banco de dados SQL gravado no Java™ que implementa a API do JDBC. Um aplicativo do console baseado no navegador está incluído.