1. Overview

A arquitetura da solução será baseada em Containers (Docker). Para garantir o provisionamento e escalonamento dos Dockers de forma eficiente será necessário a utilização de um orchestration (existem vários disponíveis no mercado). O orchestration utilizado será o Rancher — uma plataforma opensource que trabalhar em conjunto com o Kubernetes — que torna mais intuitivo a gerência dos usuários.

Como teremos 3 bases de dados com características diferentes, será adotado uma diferente tecnologia para cada tipo.

A Base de dados **A** requer segurança, e não se preocupa com o desempenho. Por isso, não faremos uso de cache para o banco (como Redis por exemplo). Em relação a tecnologia, será usado o PostgreSQL. Outra opção seria o OracleDB, porém mais uma vez foi optado pelo opensource (licença BSB).

A Base de dados **B,** apesar de também possuir dados críticos, precisa ser mais performática que a Base **A.** Para aumentar a performance desse DB, será utilizado o banco de dados MongoDB. Além de ser não relacional (NoSQL), ele também pode trabalhar de forma clusterizada. Outra opção para o aumento do desempenho seria o uso do sistema de cache Redis, que é inserido na frente do banco de dados criando um cache na memória, diminuindo a latência e aumentando o throughput das requisições. Porém é bom realizar um *Trade-off*, porque ao mesmo tempo que melhora o desempenho o Redis também deixa mais vulnerabilidades que podem ser exploradas por pessoas mal-intencionadas.

Por último, a base de dados **C** tem o requisito de ser extremamente rápida. O ideal seria utilizar um banco não relacional com cache e clusterização, porém essa base de dados envolve muitas transações. Essas transações, dependendo da complexidade, pode gerar inconsistências nos dados. Tendo isso em mente, o ideal para esse tipo de banco de dados será o uso do PostgreSQL (clusterizado) somado ao uso do Redis.

2. Integração

Os microservices que acessarão o banco **A** não tem desempenho como prioridade, então podemos usar um framework leve e que fornece o que é necessário para que o sigilo dos dados sejam mantidos. O flask em conjunto com *OAuth 2.0* (apesar desse tipo de autenticação causasr certo overhead) formam um conjunto bom para desenvolvimento de microservices que prezem pela segurança, e que não tenha desempenho como uma das prioridades. Um dado a mais que vejo necessário do banco **A** seria o/a conjugue. Essa informação seria útil para fazer cruzamento e análise de dados mais complexos. Nesse caso (e nos outros dois) é interessante que seja colocado um gateway onde o banco esteja alocado, para ele que gerenciar as conexões externas

Já o Banco **B** requer desempenho e segurança. Manter o Oauth 2.0 pode ser necessário, porém a utilização de um framework com melhor desempenho seria mais adequado. O tornado é um

framework python que trabalha de forma assíncrona, isso garante que ele funcione sem que haja bloqueio por operações de I/O de rede, tornando uma ferramenta altamente escalável se necessário.

Finalizando temos a Base **C**, que deve ter como características o alto desempenho, e não se importar tanto com segurança, já que não tem nenhum tipo de dados críticos. Uma autenticação do tipo Basic Authentication. O tornando aqui também seria uma boa escolha, devido ao desempenho requerido pela aplicação.

Os dados de todas as bases ficarão disponível através de API's Rest. Cada uma delas com suas peculiaridades e cuidados necessários para seu correto funcionamento e segurança, como já bem citado mais acima.

3. Considerações Finais

Como forma de garantir ainda mais a segurança da aplicação, seria interessante a utilização do CentOS como sistema operacional. Além de ter um longo LTS, o SO também uma ferramenta muito interessante quando se trata de segurança, o SELinux. Como essa ferramenta podemos criar políticas de acesso que restringem até o usuário root no que ele pode ou não alterar.