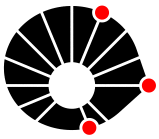

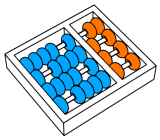

Plataforma de processamento de dados sísmicos como serviço em Nuvem

G. L. da Silva *E. Borin*

Relatório Técnico - IC-PFG-18-01
Projeto Final de Graduação
2018 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Plataforma de processamento de dados sísmicos como serviço em Nuvem

Guilherme Lucas da Silva Edson Borin

Resumo

Este trabalho é um relatório parcial de um projeto temático plurianual, visando o estudo comparativo de diversas espécies de cervejas nativas do sub-continente brasileiro. A realização deste trabalho contou com o suporte financeiro do CNPq e FAPESP, e foi imensamente facilitada pela infraestrutura de pesquisa cervisíaca instalada no Campus da UNICAMP.

Com base nessas pesquisas, determinamos que a altura da cerveja h e a altura da espuma e satifazem aproximadamente a inequação $(\sqrt{e^2 + h^2 + 2he})^3 \leq \exp(3 \log K_0^*)$, onde K_0^* é a altura do copo. Esta fórmula é válida, aparentemente, inclusive para espécies mais pigmentadas, como *Malzbier*. Em vista disso, e dos resultados análogos obtidos por A. B. Stémio em experiências com *Guaraná* e *x-Cola*, conjeturamos que a fórmula pode ser aplicada (com pequenas modificações) também a *Champagne* e outros líquidos de composição similar.

1 Indsdas

As propriedades lúdicas, mnemolíticas e catalógicas de soluções fraca- e medianamente concentradas de 1-metil-metanol em hidróxido de hidrônio aquoso ($H_3O^+HO^- \cdot nH_2O$) tem sido objeto de intensos estudos experimentais por cientistas no mundo todo [1, 2].

Durante os últimos 20 anos, os autores coordenaram uma equipe multidiplinar de pesquisadores, na UNICAMP e em outras instituições, cujo objetivo derradeiro é elucidar e[1] quantificar a relação entre...

2 Introducão

Computação em Nuvem é um conceito que está cada vez mais presente no cotidiano de todas as pessoas. Mesmo sem perceber, uma quantidade gigantesca de aplicações que usamos hoje em dia lançam mão desse conceito tão central para o desenvolvimento econômico e tecnológico de nossa sociedade contemporânea. Esse conceito se baseia na capacidade de usarmos o poder computacional de uma infinidade de serviços como Máquinas virtuais, Bancos de Dados e Redes Virutais sem necessariamente possuirmos máquinas físicas com tais serviços instalados e configurados, ainda sendo possível, de maneira simples, expor

[1] footnotes working fine

tais serviços e mecanismos para vários usuários ao redor do mundo. Entre as modalidades principais de serviços de computação em nuvem, temos três categorias mais relevantes:

- **IaaS(Infrastructure as a Service):** essa é a maneira que dá mais responsabilidade ao usuário. Essa modalidade te dá máquinas virtualizadas que você deve gerenciar instalação de programas, configurações e etc. Exemplo disso são máquinas virtuais que rodam sistemas operacionais GNU/Linux e são acessadas remotamente.

- **PaaS(Plataform as a Service):** Aqui, o usuário não precisa se preocupar com pacotes de sistema, softwares e configurações. Esse sabor de computação em nuvem, tenta facilitar para desenvolvedores publicares suas aplicações. Isso siginifca que se, por exemplo, um usuário quiser publicar uma aplicação, não vai precisar se preocupar em instalar compiladores nem pacotes necessários para rodar a aplicação naquela determinada linguagem.

- **SaaS(Software as a Service):** essa é a modalidade que traz menos autonomia para o desenvolvedor, sendo que, através dela, todo o rescurso é gerenciado pelo provedor de nuvem que é consumido. Alguns exemplos que podemos citar para ilustrar são bancos de dados como serviço, onde o usuário não se preocupa com nenhum tipo de infraestrutura, somente usa o endereço que o provedor cede e faz uso das possibildiades de armazenamento que a plataforma oferece.

Entre as vantagens de se adotar um modelo de computação em nuvem ao invés de investir em realmente adiquirir máquinas físicas, podemos citar:

- **Custo:** devido a possibilidade de pagar somente pelo o que está usando, ou seja, caso algum recurso está sendo pouco aproveitado, basta desligá-lo, usuários desse tipo de plataforma não gastarão para manter sistemas parados.

- **Escalabilidade:** A computação em nuvem permite que, caso a aplicação preciso de muito mais poder computacional do que possui no momento, seja extremamente simples aumentar o número de instâncias ou o tamanho das máquinas, para atender mais usuários, gerando assim, mais receita.

- **Foco na aplicação:** desenvolvedores podem focar totalmente em escrever suas aplicações, sem a preocupação de gerenciar infraestutura e plataforma.

Entre os principais players de nuvem pública atualmente temos gigantes como Microsoft Azure(www.azure.com), Amazon Web Services(aws.com.br) e Google Cloud Plataform(gcp.com.br), oferecendo opções extremamente diversas para computação em nuvem, desde máquinas virtuais até mesmo bancos de dados gerenciados, onde a preocupação de gerenciamente fica totalmente do lado do provedor do serviço.

Aplicações de HPC (High Processing Computing, computação de alto processamento, na tradução) são uma gama de programas que necessitam de muito poder computacional para completar as seus objetivos. Entre essas tarefas, estão o processamento de dados sísmicos, que são obtidas de maneira bruta e precisam de um trabalho muito pesado de tratamento

para que algumas conclusões possam ser feitas a partir dele. Ainda hoje, muitas dessas operações são feitas em servidores em institutos de pesquisas, por exemplo, podendo levar a um custo excessivo, trabalho excessivo para gerenciamento de infraestrutura além de baixíssima agilidade e flexibilidade quanto a infraestrura. Assim, tais aplicações podem tirar muito proveito dos provedores de nuvem citados acima.

3 Objetivos

Esse trabalho tem como objetivo desenvolver uma plataforma Open Source(CITACAO) que torne extremamente simples para usuários que não conhecem conceitos de Computação em Nuvem rodarem suas aplicações. A princípio, foram usadas aplicações para a prova de conceito. O desejado para o final da plataforma era que os usuários conseguissem processar os trabalhos sísmico sem nenhum conhecimento de Nuvem. Ao final, esperamos a criação de uma plataforma web, que tinha como principais funcionalidades:

- **Submissão e gerenciamente de dados:** Essa funcionalidade é a responsável pelo upload de dados que serão usados nos processamentos, além de consultar quais dados foram submetidos, baixá-los e também excluí-los, caso necessário.

- **Submissão e gerenciamento de binários:** esse componente do projeto é extremamente semelhante com a que detalhamos acima, porém, ao invés dos dados sísmicos, os artefatos que são gerenciados são os binários das aplicações que serão utilizados nos processos.

- **Definição de tarefas sísmicas:** Utilizando os dados e binários submetidos a partir das duas funcionalidades citadas anteriormente, o objetivo era conseguir lançar um processamento que combina os dados e binários, além dos argumentos necessários.

- **Obtenção dos resultados:** Ao final dos processos, é desejado que se consiga obter todos os resultados desse de maneira simples e rápida.

4 Relevância

Ao buscar por outras soluções que tem o mesmo intuito da plataforma que esse trabalho tem por objetivo desenvolver, notamos a sua extrema relevância, uma que vez que é extremamente difícil achar outros softwares open source que executam tal tarefa. Além da unicidade que o trabalho possui no cenários atual, ele se torna relevante já que ele pode abstrair os novos conceitos e a curva de aprendizado que vem junto com a computação em nuvem. Assim é possível tirar proveito de todos os benefícios que foram citados acima, levando a possibilidade de um uso muito mais inteligente dos recursos, baixando custos e aumentando a produtividade.

5 Procedimento

Para o sucesso final do projeto, ao começo dele, foi decidido que faríamos uma plataforma web. Essa fi a decisão devido a facilidade de desenvolver para esse tipo de plataforma, além do alcance gigantesco que plataformas web possuem, já que é praticamente obrigatório nos dispositivos que os usuários poderiam usar para acessar o sistema (smartphones e computadores) possuir um navegador de internet instalado. Além disso, vale ressaltar que desde o início do projeto todo o código estava totalmente aberto no Github, já que foi uma premissa desde o início do projeto que este seria Open Source.

Ao iniciar a análise do problema, foi possível notar que uma arquitetura de nuvem que se encaixa muito bem nesse problema é a de Work Queues(Filas de Trabalho), que segundo Brendan Burns em seu livro Designing Distributed Systems, "Em um sistema de filas de trabalho existe uma trabalho em batch para ser executado. Cada parte do trabalho é independente do outro e pode ser executado sem nenhuma interaçao". Isso é exatamente o que foi buscado, uma vez que os trabalhos eram intensivos, demoravam um tempo para serem reproduzidos e eram independentes um do outro. Assim, o que gostaríamos era que trabalhos fossem submetidos a uma fila e, alguma unidade de computação o usasse para executar uma definição de trabalho.

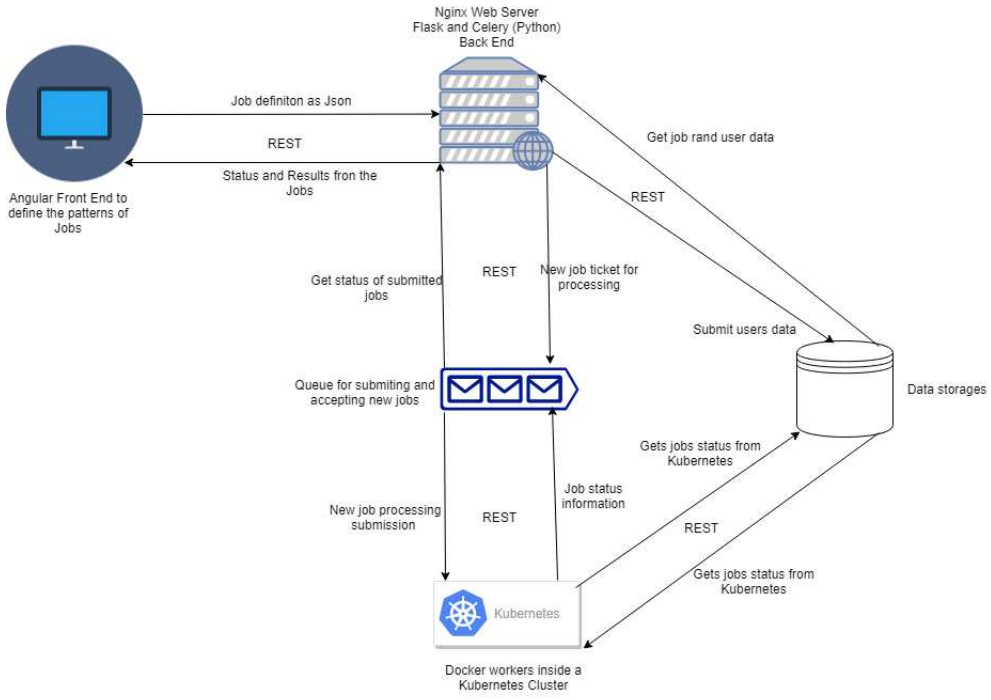

Figura 1: Arquitetura inicial do projeto e exemplo de filas da trabalho distribuídas

A imagem 1 ajuda a explicar como é o funcionamento dessa arquitetura e a arquitetura inicial do projeto.

Então, a plataforma foi separada em duas grandes partes:

- **Front-End:** A parte que o usuário realmente vê, o código que irá rodar no navegador do seu dispositivo. (Colocar alguma sitação)
- **Back-End:** Essa é a porção da plataforma responsável por características que são invisíveis ao usuário. Entre elas, podemos citar acesso ao banco de dados, autenticação, processamento de dados, submissão de dados para a nuvem, etc.

Existem alguns componentes que foram usados em comum entre essas duas partes:

- **Docker(Link):** uma maneira muito simples de empacotar as aplicações, isolando dependências, garantindo que o deploy sempre é feito da maneira correta. Assim, é possível entregar as aplicações completas, com todos os requisitos necessários instalados, o que da uma agilidade enorme durante o ciclo de desenvolvimento.
- **Nginx(Link):** Servidor web open source extremamente escalável e fácil de configurar. Foi usado tanto para servir o back end quando o front end. A outra opção para esse trabalho seria o Apache Web Server, porém o Nginx se mostrou mais simples e rápido de ter aplicações rodando.

5.1 Front End

Para o desenvolvimento do front end, foram definidas de antemão como seria a composição geral das telas e como seria o fluxo da aplicação. Assim, foi decidido que o layout geral das telas, seriam:

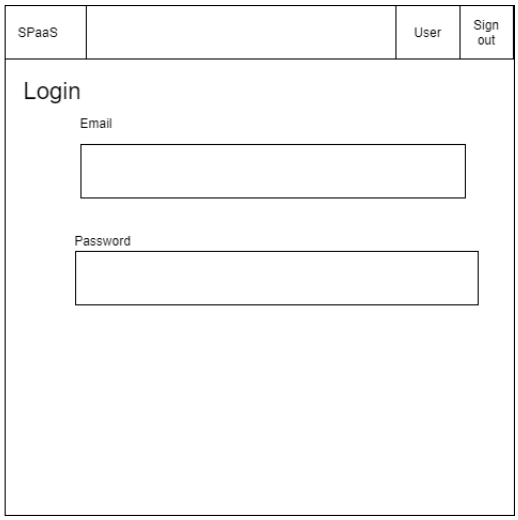

Figura 2: Página de Login

A figura 2 mostra como definimos quais informações deveriam estar na tela para que o usuário pudesse se conectar a plataforma. Como mostrado, as informações necessárias para logar no sistema eram email e senha que foram previamente cadastrado.

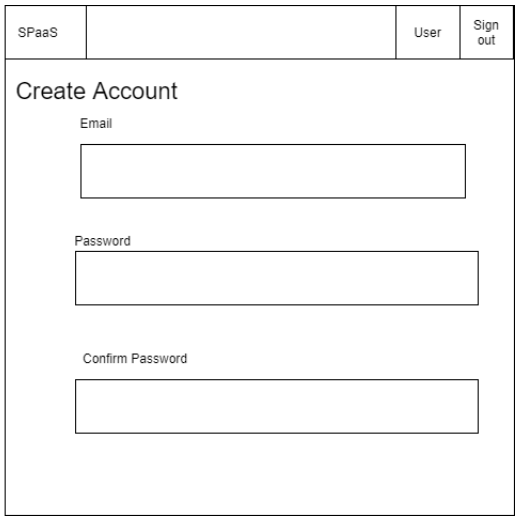

Figura 3: Página de criaçao de conta

Figura 4: Página de definição dos trabalhos

A figura 3 deixa claro quais foram as informações que eram requeridas para se criar uma conta no sistema.

A figura 4 exemplifica como imaginamos a tela de definicao de um trabalho para ser executado. Nesse primeiro protótipo, foi pensado que somente as informações mostradas na tela, eram o suficente.

O protótipo da tela de status dos trabalhos está mostrada na figura 5. Extremamente simples a princípio, uma tabela mostrando o status de cada trabalho.

A ultima figura **??** mostra como foi pensada a obtenção dos resultados.

Vale ressaltar que essa ideia inicial foi considerada com poucas das funcionalidades que realmente eram relevantes para a plataforma. Assim, o fluxo de todo o trabalho foi repensado para algo mais próximo do representado na figura 7. Nesse fluxo definido, os

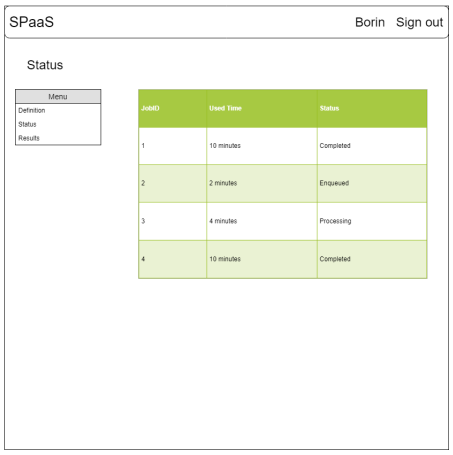

Figura 5: Página de observação do status

Figura 6: Página de obtenção dos resultados

quadrados representam as telas, com suas respectivas interações com as camadas de armazenamento que são necessárias para completar suas tarefas com sucesso. Assim, entrando no detalhe de cada componente da solução:

- **Data Management:** Componente responsável pelo gerencimento dos dados sísmicos da plataforma. A intenção nesse componente é ser capaz de adicionar, remover e movê-los de um tipo de armazenamento mais caro (porém mais rápido) para um mais barato, ligeiramente mais lento.
- **Flow Management:** Componente responsável por determinar uma sequencia de passos que devem ser feitas para completar uma tarefa. Diferente dos trabalhos (Tasks), aqui a definição é de um conjunto de tasks, encadeadas para obter um resultado maior.
- **Tools Management:** Essa funcionalidade procura realizar o gerenciamento dos

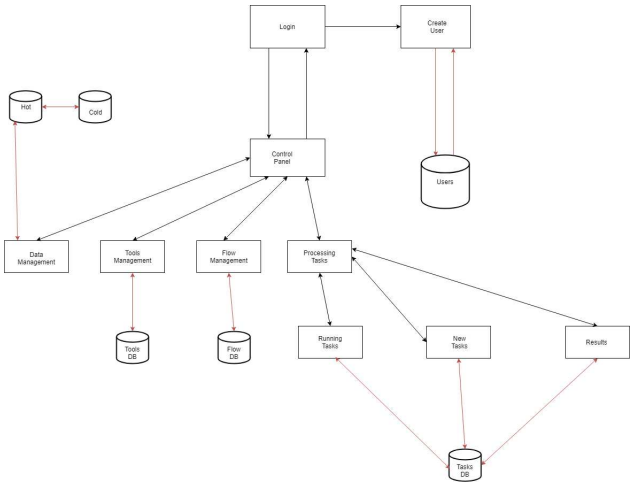

Figura 7: Fluxo entre os componentes da plataforma

binários que serão executados. Assim como o componente respoonsável pelo gerenciamento dos dados, esse componente busca dar a possibilidade de adicionar, mover e excluir binários da plataforma.

- **Running Tasks:** Nessa tela, é desejado que o usuário consiga ver qual é o status dos processamentos que iniciou. Status como Pendente e Pronto são alguns dos possíveis.

- **New Tasks:** Responsável por definir novas tarefas a serem rodadas. É desejado que as características necessárias para essa definição sejam binário, dado sísmico e argumentos para a execução acontecer com sucesso.

- **Results:** Os resultados dos processamentos executados estarão aqui, organizados pela execução que o gerou.

Como tecnologias para o desenvolvimento da nossa plataforma, especificamente para o front end, foram escolhidas as seguintes tecnologias:

- **Angular(LINK):** Framework(link abaixo) para desenvolvimento de aplicações web Open Source, desenvolvido inicialmente pelo Google e agora também mantido pela comunidade. Usa como linguagem de desenvolvimento o Typescript(LINK). A escolha desse framework aconteceu uma vez que ela é extremamente produtiva e por ser um framework completo, já encapsulando todo o conceito de serviços, estilo de página e linguagem de marcação, roteamento. Assim, com esse framework ainda é necessário conhecer linguagens de marcação como HTML (link) e CSS (link), mas a junção de todos esses conceitos fica muito mais simples.

- **Bootstrap(LINK):** Frameworks que possui o estilo de vários componentes web prontos, como tabelas, menu laterais, barras superiores, etc. É o que muitos sites usam

hoje em dia, como é o caso do github(LINK) Estilizar páginas web pode ser um trabalho extremamente complicado e demorado, por isso optamos por usar um framework. Existem outros frameworks CSS interessantes, porém poucos tem a maturidade do bootstrap.

Foi garantido que o deploy fosse feito da maneira mais simples e flexível para o cliente. Então, foi usado o servidor web nginx(site) para isso. Aém disso, também foi disponibilizado um Dockerfile, que possibilita o deploy da solução de maneira extremamente simples e direta sobre Containers Docker.

5.2 Back End

Ao início do desenvolvimento do back end, buscamos definir quais seriam as melhores tecnologias para esse componente, além de definir como seriam as rotas e contratos que a API (link) do back end iria expor para que o front end pudesse consumir de forma eficaz e direta, garantindo assim, um total desacoplamento entre os dois componentes. Assim, os componentes que decidimos para o back end foram os seguintes:

- **Python(link):** Linguagem criada por Guido Van Rossum, extremamente relevante nos dias de hoje devido a sua enorme gama de aplicações, que vão desde embarcados até sistemas de análise de dados de alto desempenho. A escolha dela foi devido principalmente a bibliotecas muito maduras para desenvolvimento de API's back end, filas de trabalho distribuídas e conectores com bancos de dados.

- **Flask(link):** Framework para desenvolvimento de APIS para Python. Foi escolhido devido a extrema agilidade para colocar uma API rodando. Além disso, é muito simples entender como a aplicação está funcionando, fator muito crítico para decidirmos na escolha desta, uma vez que o projeto tem como premissa ser Open Source e facilitar o entendimento de todos que o usam. Ao invés de Python/Flask, foram cogitadas também ASP .NET com C# e NodeJS. A primeira foi descartada devido a complexidade para simplesmente criarmos uma API. NodeJS não foi escolhido devido a dificuldade que novos programadores podem enfrentar para compreender as fucnionalidades em um primeiro momento, quando fossem aumentar o projeto.

- **Celery(link):** Uma vez escolhido Python e Flask como ferramentas para o desenvolvimento da solução, foi muito natural a escolha do Celery como ferramenta para trabalhar com filas de trabalho. Existem muitos casos de sucesso espalhados pela internet de engenheiros e desenvolvedores que usaram essa combinação com total eficácia. O Celery é um componente do Python que é o responsável por gerenciar, submeter e obter resultados de uma fila de trabalho, de uma maneira muito simples. Além disso, dá suporte a vários componentes de mensageria, como por exemplo, RabbitMQ(link), Redis(link), Amazon SQS(link).

- **Redis(link):** Após a escolha de qual biblioteca iríamos usar para facilitar a tarefa de filas de trabalho, que foi o Celery, foi preciso escolher algum componente para a mensageria do projeto, ou seja, o componente que realmente é responsável por armazenar

e distribuir as mensagens relacionadas aos trabalhos submetidos na solução. Entre as opções estavam RabbitMQ, Redis e Amazon SQS. Essas as que melhor se integravam com o celery, sendo possível obter todas as suas funcionalidades com qualquer um desses três. Porém, Amazon SQS é uma opção presa a um provedor de nuvem, o que complica o andar do projeto, caso seja preciso usar outra cloud diferente da AWS. RabbitMQ e Redis ofereciam funcionalidades muito parecidas, porém o Redis foi escolhido, devido a infinidade de materiais sobre o uso dele e suas outras funcionalidades que podem ser tiradas proveito também, como, por exemplo, cache e armazenamento de dados em memória.

- **MongoDB(link):** Para o armazenamento dos dados de cadastro de usuários, informações sobre argumentos de cada tipo de binário e etc, a escolha foi um banco de dados NoSQL(referencia), o MongoDB. A escolha deste foi muito direta, uma vez que os dados que estavam trafegando entre um serviço e outro tinham uma estrutura muito parecida com os documentos que são guardados nesse tipo de banco, não sendo necessário de preocupar em transformar os dados para os esquemas presentes em tabelas SQL(referencia), o que agiliza muito o trabalho.

- **Blob Storage(link):** o Blob Storage do Microsoft Azure foi a opção aqui pois era necessário uma espécie de armazenamento de propósito geral. Aqui ele está sendo usado para armazenamento dos dados sísmicos e binários. Foi escolhido devido a excelente integração que possui com Python. Outras opções existentes eram, por exemplo, Amazon S3(link), que oferece serviços muito parecidos. Outra opção era deixar esses dados em no disco de máquinas virtuais e serví-los por meio de servidores, porém, isso aumentaria muito a complexidade e dificultaria a tarefa de gerenciar esse tipo de infraestrutura.

5.3 Ferramentas para Deploy

Como dito anteriormente, ao iniciar o projeto, não era desejado que estivéssemos presos a um modelo específico de deployment. Assim, para facilitar o deploy da aplicação independente do modo que foi escolhido para o deploy, foi criado uma série de scripts em ferramentas específicas para ajudar. Existem dois modelos que estão sendo facilitados:

- **Kubernetes(link):** Orquestrador de Containers extremamente famoso nos dias de hoje, que tem como principal funcionalidade tornar mais simples, escalável e performática o deploy de aplicações baseadas em Containers. Nascida dentro do Google, a partir do Borg (paper) atualmente está sob a jurisdição da Cloud Native Foundation(link) e é Open Source. Foi uma das plataformas que foi visada por ter apresentado extremamente relevancia nos ultimos tempos, com todos os grandes provedores de Nuvem oferecendo opções gerenciadas dessa ferramenta.

- **Ansible(link):** um modelo muito comum de deploy de aplicações é baseada em máquinas virtuais. Nesse caso, é bom garantir que todas as dependências necessárias para que a aplicação rode com sucesso estejam instaladas. Nesse aspecto, o Ansible se

mostra extremamente eficaz. Essa ferramenta surgiu da Red Hat e também é mantida Open Souce, sendo um ótimo gerenciador de configurações, que torna possível escrever configurações como código, o que garante que a aplicação sempre terá sucesso para ser executada.

5.4 Resultados

Após seguir todo o experimento citado acima, foi alcançado um resultado muito próximo do esperado. A arquitetura final do projeto, está mostrada na figura (linkar a figura).

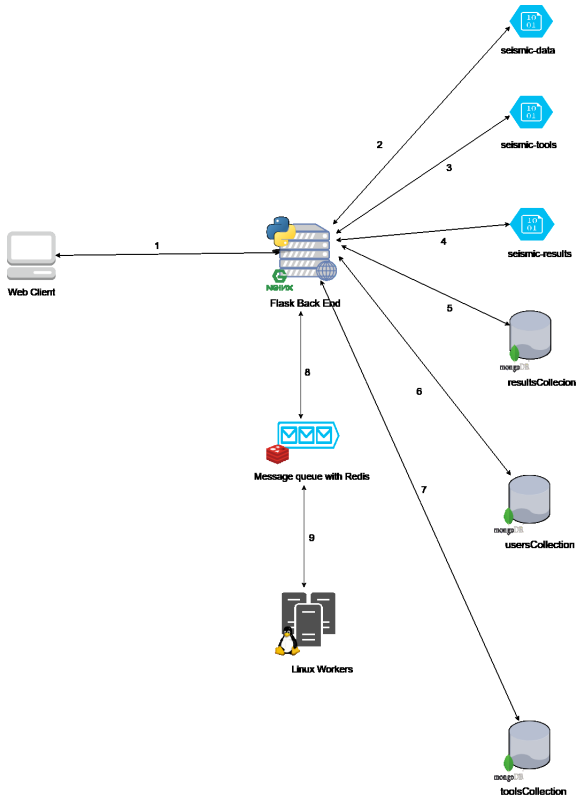

Figura 8: Arquitetura final do projeto

Dada a arquitetura final do projeto mostra na figura 8 os componentes mostrados tem, cada um, a seguinte função:

- **Web Client:** componete que roda no lado do cliente. Seguimos com o que foi proposto no início do projeto e explicado durante o procedimento, ou seja, todo o código do lado do cliente foi escrito em Typescript, com o auxílio do framework Angular, a estilização das páginas foi feita com Bootstrap e servido via Nginx.
- **Flask Back End:** responsável por receber todas as requisições que saem do front end. Funciona como um gateway(talvez uma citação). Também foi seguido o plano

inicial de escrever todo o back end com Python, Flask e Celery. Esse é o componente que possui mais responsabilidades: autenticação, comunicação com a fila com o auxílio do Celery, acesso aos Blobs e acesso ao banco.

- **Linux Workers:** Componentes que são responsável pela execução propriamente dita dos pedidos. Com o auxílio do Celery, esperam por pedidos entrarem na fila, e assim que isso acontece, executam. É possível ter um cluster de máquinas com esse papel, aumentando o paralelismo das execuções.

- **Message queue with Redis:** também parte do plano inicial, a fila de mensagens com o Redis foi a maneira mais eficaz e resiliente que encontramaos de desacoplar o back end dos Linux workers. Assim O back end somente precisava submeter uma mensagem com as definições para a execução de um trabalho na fila e algum worker pegaria essa requisição e faria o processamento necessário. Aqui vale um observação importante. A escolha do Celery e do modelo de filas de trabalhos distribuídas, que já foi explicada anteriormente, torna possível, além de desacoplar o back end dos workers, ainda torna possível a execução de processamentos em paralelo e distribuídos, não criando um gargalo na solução, possivelmente ocasionado pelo grande número de pedidos na fila esperando por um único worker.

- **toolsCollection:** coleção dentro do banco que guarda informações sobre os binários que estão armazenados. Novos dados são inseridos nela quando submetidos um novo binário é armazenado, quando um novo pedido de processamento será incluído, já que é necessário consultar quais os parâmetros para esse e também quando um binário é excluido, apagando o registro. O modelo que está sendo usado para guardar essas informações é o que esta na 9

- **resultsColleciton:** coleção de documentos no banco para guardar os dados do resultado, armazena o id do processamento, dado e binário usado, além dos argumentos. Essa coleção é acessada após o término de cada processamento, para salvar as informações do resultado e também quando é desejado que todos os resultados sejam mostrados na tela de gerenciamento de resultados. O modelo segue o padrão da 10.

- **usersCollection:** usada para armazenamento de dados dos usuários, como senha e email para autenticação. Um novo registro é criado nessa coleção quando um usuário cria uma nova conta. A coleção também é acessada para autenticação de um novo usuário. Para que todos esses processos sejam executados com sucesso, o modelo usado é o da 11.

- **seismic-tools:** Blob usado para armazenar os binários submetidos pelos usuário. Acessado pelo componemte de gerenciamento de ferramentas quando uma nova ferramente é inserida na plataforma e quando o processamento vai acontecer, pois nesse momento é necessário baixar esses dados para execução em um dos workers.

- **seismic-data:** Blob que armazena os dados sísmicos submetidos. Também são acessados pelo componente de gerenciamento de dados quando um novo dado é incluido, excluído e na execução dos trabalhos, já que é necessário baixar esses dados.

```
{
    "_id" : ObjectId("5be74b71c0f26007790bc37d"),
    "args" : [
        {
            "description" : "Azimute (em graus) do dado sol",
            "name" : "1"
        },
        {
            "description" : "Velocidade inicial",
            "name" : "2"
        },
        {
            "description" : "Velocidade final",
            "name" : "3"
        },
        {
            "description" : "Step da velocidade",
            "name" : "4"
        },
        {
            "description" : "Abertura limite em meio afastamento (Semblance)",
            "name" : "5"
        },
        {
            "description" : "Abertura limite em meio afastamente (Empilhamento)",
            "name" : "6"
        },
        {
            "description" : "Dado prestack",
            "name" : "7"
        },
        {
            "description" : "Pasta de armazenamento dos dados postack",
            "name" : "8"
        }
    ],
    "name" : "cmp"
}
```


Figura 9: Modelo presente na coleção de ferramentas binárias

- **seismic-results:** Outro blob. Esse, responsável por armazenar os resultados dos processamentos. Todos os resultados são empacotados em um arquivo tar.gz(link) e disponibilizados para download, no componente responsável por gerenciar os resultados.

Figura 10: Modelo presente na coleção de resultados

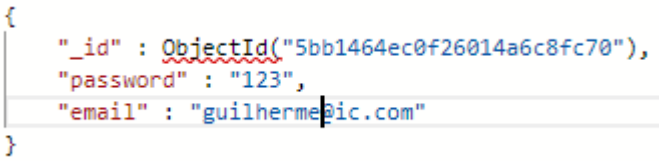

Figura 11: Modelo presente da coleçao de usuários

Referências

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley (1901).

[2] D. E. Knuth and L. Lamport, *A structural analysis of the role of gnus and gnats in the post-modernistic, crypto-existential Weltanschauung of neo-liberal Tibeto-Vietnamese leaf blower operators as manifest in the sexual symbology of the Los Angeles Phone Directory.* Journal of Gnu Technology, **23** (6), 12–87 (March 1996).