UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Hiago Porto Dutra

Implantação de Sistemas de Visualização Científica sob demanda utilizando a plataforma Amazon Web Services

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Hiago Porto Dutra

Implantação de Sistemas de Visualização Científica sob demanda utilizando a plataforma Amazon Web Services

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Bruno Augusto Nassif Travençolo

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2018

Hiago Porto Dutra

Implantação de Sistemas de Visualização Científica sob demanda utilizando a plataforma Amazon Web Services

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 13 de dezembro de 2018:

Prof. Dr. Bruno Augusto Nassif Travençolo
Orientador

Mestre Eduardo Henrique Silva

Mestre Jean Roberto Ponciano

Uberlândia, Brasil
2018

Agradecimentos

Agradeço a Deus por me dar força de seguir em todos os desafios propostos ao decorrer da vida. Agradeço a minha família e namorada por me incentivar e apoiar na chegada do fim deste ciclo. Agradeço a meu orientador por me orientar e ter paciência pela vida conturbada que levo devido ao trabalho exercido durante o desenvolvimento desta Monografia. Também sou grato a todos os professores que me repassaram conhecimento de alta qualidade durante os anos universitários.

Resumo

A computação em nuvem vem evoluindo constantemente com o passar dos anos, a virtualização de ambientes computacionais permite a implantação de sistemas de diversas finalidades com baixo custo. A visualização científica também evoluí bastante com auxílio das tecnologias associadas a GPU. Este trabalho teve por objetivo desenvolver uma aplicação que possibilita implantar diversos sistemas de visualização científica na nuvem, com pagamento sob demanda com recursos computacionais de ponta e alta qualidade. São exploradas tecnologia e métodos como arquitetura sem servidor, Infraestrutura como Código e Aplicações de Página Única para desenvolvimento e implantação do sistema proposto na nuvem da Amazon Web Service.

Palavras-chave: AWS, Computação em nuvem, Visualização científica, Paraview, Infraestrutura como código .

Lista de ilustrações

Lista de abreviaturas e siglas

AMI	*Amazon Machine Image*
AWS	*Amazon Web Services*
CPU	*Central Process Unit*
EC2	*Amazon Elastic Compute Cloud*
FaaS	*Function as a Service*
GPU	*Graphics Processing Unit*
IaaS	*Infrastructure as a Service*
IaC	*Infrastructure as code*
IP	*Internet Protocol*
JS	*JavaScript*
PaaS	*Plataform as a Service*
REST	*Representational State Transfer*
S3	*Amazon Simple Storage Service*
SaaS	*Software as a Service*
SDK	*Software Development Kit*
SLA	*Service Level Agreement*
SO	Sistema Operacional
SPA	*Single Page Application*
SSH	*Secure Shell*
VM	*Virtual Machine*
VTK	*Visualization Toolkit*

Sumário

1 Introdução

As novas plataformas de sistemas distribuídos impulsionaram a computação em nuvem. Grandes empresas como Amazon, Google e Microsoft são provedores desse novo conceito, que ganha importância e evolui rapidamente. Os arquitetos de software e pesquisadores consideram a computação em nuvem atrativa por eliminar a necessidade de investimento prévio com recursos físicos e tornar possível uma infraestrutura escalável. Esta se caracteriza por uma arquitetura capaz de alterar de modo dinâmico conforme a demanda (ZHANG; CHENG; BOUTABA, 2010).

Com o grande crescimento da computação em nuvem surgiram termos e métodos que auxiliam o desenvolvedor de software a implementar e gerenciar aplicações de dimensões e complexidades distintas. Conceitos como computação sem servidor e Infraestrutura como Código ajudam no desempenho da codificação da aplicação e reduz a complexidade da implantação e gerenciamento se comparados aos cenários fora da nuvem (GIENOW, 2018).

Além de todas as vantagens que a nuvem traz em comparação aos ambientes fora dela, o custo para manter a aplicação cai quando esses novos conceitos são utilizados. É possível manter toda aplicação em serviços sem servidor, ou seja, a cobrança é feita apenas pelo o que é usado, eliminando a necessidade de ter um ambiente ativo a todo tempo.

A visualização científica é a representação visual de estudos científicos por computação gráfica. Os estudos científicos em sua maior parte geram dados de grande extensão, que exigem hardware de alto valor financeiro, como placas de vídeo, para o processamento e visualização em tempo satisfatório. Muitos projetos não têm este potencial financeiro para o investimento necessário em recursos computacionais, dificultando o processamento dos dados e consequentemente causam atrasos na obtenção de resultados.

A computação em nuvem pode oferecer processamento com GPU sob demanda, facilitando a utilização de recursos de alto desempenho sem a necessidade de um investimento inicial de hardware. O processamento em GPU ainda não oferece serviços sem servidor, a necessidade de manter um ambiente virtualizado em execução é necessária.

Várias tecnologias desenvolvidas para utilizar Infraestrutura como Código já possuem integração com os provedores de nuvem, o que possibilita montar um ambiente completo de forma automática em minutos, e após a utilização excluí-lo sem nenhuma dificuldade.

Visto a evolução das tecnologias da computação em nuvem e a necessidade de

possibilitar a utilização de recursos de alto desempenho com baixo custo para pesquisas, este trabalho propôs uma aplicação web com arquitetura sem servidor, tendo o papel de controlar o ciclo de vida dos ambientes providos de GPU. A aplicação é capaz de iniciar um ambiente com GPU rodando o Kitware Visualizer, um sistema de visualização utilizado na análise de amostras científicas. O aplicativo web também possibilita o usuário excluir todo o ambiente montado, reduzindo o custo operacional do serviço.

1.1 Objetivo

Os objetivos desta monografia foram divididos em geral e específicos.

1.1.1 Geral

O objetivo deste trabalho é de desenvolver um aplicativo capaz de gerenciar ambientes sob demanda na nuvem, com capacidade de processamento em GPU utilizando sistemas virtualizados em contêineres Docker.

1.1.2 Específicos

- Desenvolver um ambiente capaz de fornecer aplicações que demandem alto poder de processamento gráfico.
- Validar o ambiente criado com a implantação de uma aplicação de visualização científica.
- Criar um modelo de Infraestrutura como Código para automatizar a implantação e remoção do ambiente virtualizado na nuvem.
- Desenvolver uma aplicação web para gerenciar o ciclo de vida dos ambientes na nuvem.

2 Computação em Nuvem

A Computação em Nuvem, em inglês *Cloud Computing*, é um paradigma de alocação e entrega de serviço pela Internet. Ela é muito atrativa pois inibe a necessidade de investimento em hardware, e abre a possibilidade de iniciar um pequeno projeto e aumentar de acordo com a demanda.

Nos ambientes tradicionais de computação em nuvem a provisão de serviços é dividida em dois tópicos: provisão de infraestrutura e provisão de serviços. Grandes empresas como Google, Amazon e Microsoft disponibilizam serviços em nuvem, confiáveis e de alto custo benefício, elas adaptam e criam ferramentas para aproveitar os diferenciais desse novo paradigma de computação distribuída. Tais benefícios podem ser observados pelo fato de não ser necessário o investimento em uma estrutura física para iniciar um projeto, as operações para expansão e redução de recursos são de baixo custo e fácil gerenciamento. É um ambiente de alto potencial escalável, acessado de qualquer dispositivo que tenha acesso a Internet, trazendo uma redução drástica de complexidade e inibindo a manutenção de hardware (ZHANG; CHENG; BOUTABA, 2010).

Logo, a computação em nuvem pode ser definida em um modelo de acesso fácil, sob demanda, em tempo real. Não obstante, conveniente para um ambiente compartilhado de recursos de computação configuráveis, por exemplo: redes, servidores, armazenamento, aplicações e serviços. Estes que podem ser rapidamente fornecidos e liberados com esforço de gerenciamento mínimo, com interação ao serviços do provedor. Tal modelo de nuvem é composto por cinco características essenciais, três modelos de serviço, e quatro modelos de implementação que serão descritos abaixo (MELL; GRANCE, 2011).

2.1 Arquitetura

A arquitetura do ambiente de computação em nuvem pode ser divido em 4 camadas: física, de infraestrutura, de plataforma e de aplicação.

- A camada física corresponde a implementação do Data Center que hospeda o serviço da nuvem. Ela contém centenas de servidores organizados em armários e interligados por switches, roteadores e outras interfaces.
- A camada de infraestrutura é responsável por criar o conjunto de recursos computacionais usando tecnologias de virtualização. A alocação dinâmica de recursos só é possível devido aos métodos de virtualização.

- A camada de plataforma é montada no topo da camada de infraestrutura. Ela consiste em sistemas operacionais e aplicações em plataformas disponibilizadas pelo provedor da nuvem. Tais plataformas geralmente dispensam a necessidade de lidar com um servidor, entregando a funcionalidade especifica do serviço reduzindo trabalho operacional e monitoramento.

- A camada de aplicação é o topo da arquitetura da nuvem, nela são executados os sistemas utilizados pelo usuário final. Aplicações disponibilizadas na nuvem integradas com as camadas inferiores, geralmente entregam serviços com uma boa experiência para o usuário final.

A Figura 1 demonstra como funciona a arquitetura de um ambiente em computação em nuvem.

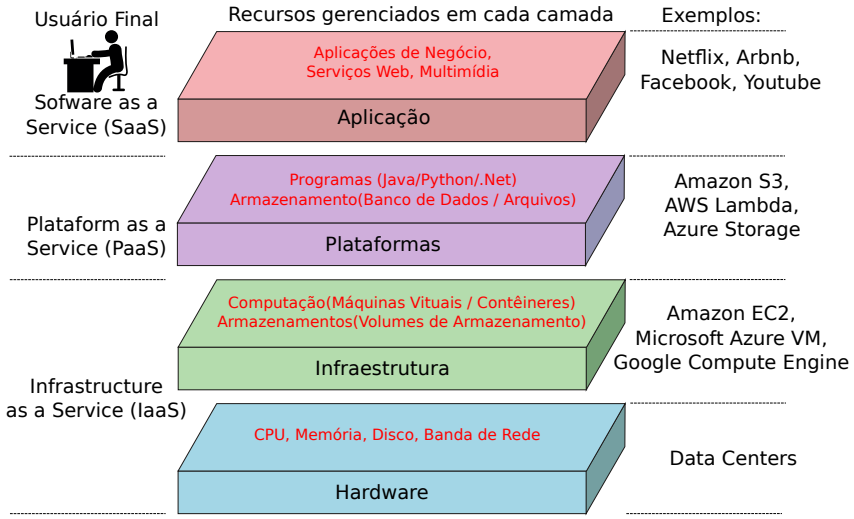

Figura 1 – Arquitetura de um ambiente em computação em nuvem. Figura adaptada de (ZHANG; CHENG; BOUTABA, 2010).

A arquitetura modularizada permite que a computação em nuvem suporte uma grande gama de requerimentos para aplicações, enquanto reduz o gerenciamento e manutenção de todo o sistema (ZHANG; CHENG; BOUTABA, 2010).

2.2 Modelos de negócio

A computação em nuvem emprega um modelo de negócio dirigido a serviço. Na prática são oferecidos serviços em cima das camadas da nuvem, estes modelos podem ser agrupados em três categorias principais.

- IaaS, iniciais de *Infrastructure as a Service* em inglês, no qual refere-se a provisão de recursos de infraestrutura, em termos é a virtualização de ambientes. Alguns serviços de IaaS são Amazon EC2, Azure Virtual Machines e Google Compute Engine.

- PaaS, iniciais de *Plataform as a Service* em inglês, refere a camada de plataformas, incluindo suporte para serviços de computação de várias finalidades para desenvolvimento de software. Exemplos de serviços PaaS são Google App Engine, Microsoft Windows Azure, AWS Lambda, Cloudformation, entre outros.
- SaaS, iniciais de *Software as a Service* em inglês, referente a provisionamento de aplicativos pela Internet hospedados na nuvem. Alguns exemplos são Netflix, Airbnb e aplicação desenvolvida neste trabalho.

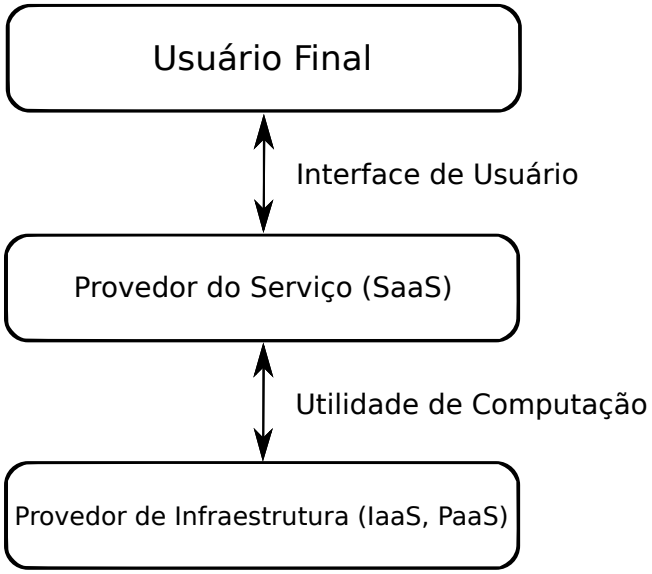

Figura 2 – Representação das camadas do modelo de negócio na computação em nuvem. Figura adaptada de (ZHANG; CHENG; BOUTABA, 2010).

Observando a arquitetura da computação em nuvem é visível que PaaS sempre irá executar em cima de um IaaS, por isso no contexto atual os provedores dos serviços são da mesma organização que desenvolvem as plataformas (ZHANG; CHENG; BOUTABA, 2010).

2.3 Características

A computação em nuvem possui características, que se diferenciam dos sistemas tradicionais. A primeira é que vários clientes ficam alocados em um mesmo *Data Center*, logo todos dividem a mesma infraestrutura, fato que leva à separação natural das responsabilidades de cada camada na nuvem. Essa divisão faz com que o responsável por cada camada foque apenas nela sem se preocupar com as demais, além dos provedores da nuvem adotarem modelo de operação orientada a serviço, utilizando SLA, iniciais de *Service Level Agreement* em inglês, de atendimento. Caso haja algum tipo de falha na infraestrutura, existe um prazo que é negociado com os clientes de acordo com arquitetura de cada sistema.

Os recursos computacionais compartilhados fazem com que cada usuário crie seus próprios ambientes virtualizados, os gerencie e opere seu custo. O usuário pode alocar e desalocar recursos de forma dinâmica graças a virtualização. Além disso, a nuvem possui um sistema de auto gerenciamento de recursos computacionais para auxiliar o controle dos ambientes hospedados nela.

Outro fato importante é que existem vários *Data Centers* dos grandes provedores de nuvem em todo o Planeta. Tal variedade geográfica faz com que a conexão com os serviços da nuvem tenham desempenho otimizado pela proximidade entre o usuário e o servidor de acesso, isto acontece porque a conexão é feita pela Internet, tornando a proximidade é um fator relevante (ZHANG; CHENG; BOUTABA, 2010).

3 Tecnologias e métodos

Este capítulo apresenta as tecnologias utilizadas no desenvolvimento deste projeto, bem como os métodos para implantação do mesmo na nuvem.

3.1 ParaView

O Paraview é uma ferramenta para visualizar e analisar conjuntos de dados, além de fornecer a criação e execução dinâmica de tarefas de visualização (JOURDAIN; AYACHIT; GEVECI, 2010).

O ParaView é um projeto de código aberto que utiliza o Visualization Toolkit, conhecido como VTK (SCHROEDER; LORENSEN; MARTIN, 2004), para processamento de dados e sistema de renderização. Ele possui funcionalidades como processamento batch, interação 3D, criação de visualizações para analisar dados usando técnicas qualitativas e quantitativas e pode processar uma grande quantidade de dados utilizando sistema de memória distribuído em clusters.

3.1.1 Visualizer

O Visualizer é um aplicativo web que aproveita os recursos ParaView como servidor para produzir visualizações interativas pelo navegador. Ele foi desenvolvido pela Kitware, para exemplificar os recursos da biblioteca ParaviewWeb escrita em Javascript (KITWARE, 2018a).

3.2 Unidade de processamento gráfica

A Unidade de Processamento Gráfica, ou GPU do inglês *Graphical processing unit*, é um tipo de processador encontrados nas placas de vídeo. Bem diferentes das Unidades de Processamento Central, ou CPU, que têm função de controlar todo ambiente computacional, a GPU tem função de fazer cálculos mais específicos e em paralelo. Este tipo de tarefa comumente é utilizada com finalidades gráficas dentro de um sistema operacional.

Enquanto as CPU mais atuais possuem no máximo 28 núcleos (LEATHER, 2018), as GPU mais poderosas podem chegar a 5120 (NVIDIA, 2018a). A grande diferença dos núcleos das duas categorias é a complexidade do que é feito neles: na CPU são feitos cálculos para coordenar todo ambiente computacional, enquanto os feitos na GPU tem finalidade mais específica. Os cálculos feitos na GPU são paralelizados, o que torna eles

mais simples para cada núcleo, agilizando o processamento das tarefas atribuídas a eles em relação às de CPU.

A visualização científica não sai muito desta regra: a utilização de GPU para processamento do Paraview traz grandes vantagens, como o ganho de agilidade na entrega de resultados virtualizados a medida que o servidor está equipado com o recurso.

3.2.1 OpenGL

OpenGL é uma biblioteca escrita em C e C++ destinada ao desenvolvimento de aplicações com conteúdo gráficos, ambientes 3D, jogos, entre outros (GROUP, 2018). O Paraview utiliza a biblioteca OpenGL para o processamento de seus ciclos de virtualização na GPU, dispensando tempo significativo de processamento.

3.3 Docker

Um contêiner é uma unidade padrão de software que empacota o código e todas as suas dependências para que o aplicativo seja executado de maneira rápida e confiável de um ambiente de computação para outro (INC., 2018b). O Docker é uma tecnologia que permite a criação e o uso de contêineres Linux. O sistema foi criado pela Docker Inc., porém o projeto se tornou código aberto e possui uma comunidade de desenvolvedores apoiando-o com melhorias e correções ágeis.

Uma imagem de contêiner do Docker é um pacote de software leve, autônomo e executável que inclui tudo o que é necessário para executar um aplicativo, como código, tempo de execução, ferramentas do sistema, bibliotecas do sistema e configurações. (INC., 2018b)

Quando são utilizadas máquinas virtuais, em cada uma delas é executado um sistema operacional exclusivo. Cada virtualização pode possuir um sistema operacional diferente, mesmo executando todas em um único servidor físico. Cada virtualização tem seus próprios binários, bibliotecas e aplicativos que a atende, gerando muita alocação de recurso computacional.

A tecnologia Docker usa recursos do *kernel* do Linux em que está instalando para segregar processos de modo que eles possam ser executados de maneira independente. O objetivo dos contêineres é criar essa independência, criando a possibilidade de executar diversos processos e aplicativos separadamente e utilizar melhor a infraestrutura, além de manter a segurança necessária em sistemas separados (HAT, 2018).

As ferramentas de contêiner, incluindo o Docker, fornecem um modelo de implantação com base em imagem. Isso facilita o compartilhamento de um aplicativo ou conjunto de serviços, incluindo todas as suas dependências, em vários ambientes.

A modularidade do Docker permite serviços com funções distintas conseguirem executar separados, mesmo fazendo parte do mesmo aplicativo.

Cada arquivo de imagem Docker é composto por uma série de camadas. Essas camadas são combinadas em uma única imagem. Uma nova camada é criada quando há alteração na imagem. Toda vez que um usuário especifica um comando, como executar programas ou copiar arquivos, uma nova camada é criada (HAT, 2018).

O interessante de utilizar camadas é a possibilidade de montar o ambiente para execução da sua aplicação uma única vez, depois disso ele poderá executar em qualquer recurso computacional que aceite Docker sem risco de erros.

O Docker reutiliza essas camadas para a construção de novos contêineres, o que torna o processo de criação muito mais rápido. As alterações intermediárias são compartilhadas entre imagens, o que melhora ainda mais a velocidade, o tamanho e a eficiência. O controle de versões é inerente ao uso de camadas. Sempre que é realizada uma nova alteração, gera-se um histórico de mudanças integrado, o qual fornece controle total sobre as imagens do contêiner (HAT, 2018).

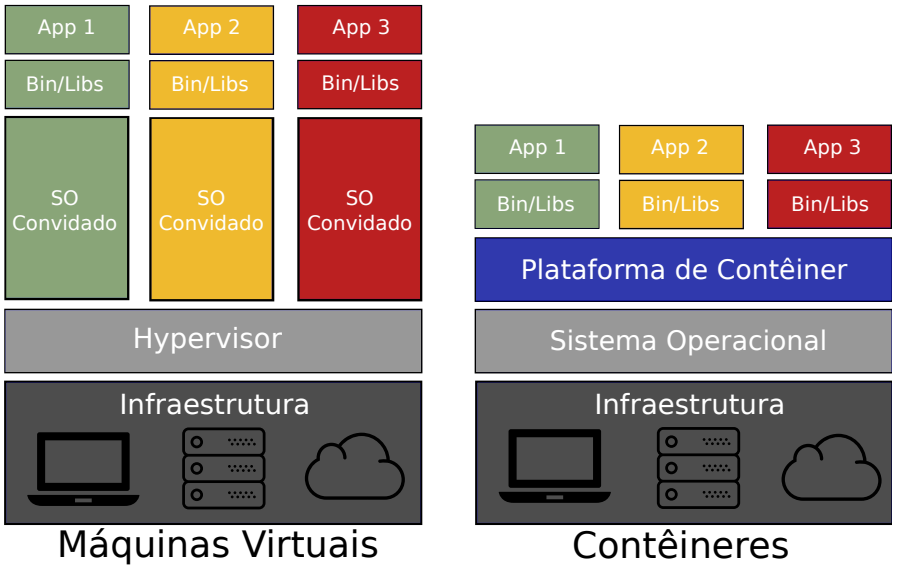

Figura 3 – As máquinas virtuais são gerenciadas por um *hipervisor* e utilizam o hardware da VM, enquanto os sistemas de contêiner fornecem serviços do sistema operacional a partir do *host* subjacente e isolam os aplicativos usando o hardware em ambiente virtualizado. Figura adaptada de (BREY, 2018).

3.3.1 Dockerfile

O Docker pode criar imagens automaticamente lendo instruções em um Dockerfile. O Dockerfile é um documento de texto que contem instruções Linux via linha de comando nos quais podem ser incorporados em uma imagem Docker (INC., 2018a).

No inicio do Dockerfile, com a palavra reservada "FROM", deve apresentar uma imagem base. Existem vários tipos de imagens base disponibilizados pela comunidade de desenvolvedores, elas podem ser encontradas no Docker Hub. O Docker Hub é um repositório que contem milhares de imagens disponibilizadas por empresas e desenvolvedores, sem custo algum.

Após selecionada uma imagem base, a cada linha é adicionado um comando para o sistema operacional da base selecionada. Após a execução de todos os comandos, uma imagem será montada e poderá ser executada em qualquer servidor que contenha o Docker instalado.

3.3.2 NVIDIA Container Runtime for Docker

Os contêineres Docker encapsulam as dependências dos aplicativos para fornecer uma execução confiável e reproduzível. O ambiente de execução NVIDIA-Docker permite a implementação de aplicativos acelerados por GPU em qualquer servidor Linux que possua o recurso e tenha suporte a ele (NVIDIA, 2018c).

Com o NVIDIA-Docker é possível utilizar contêineres acelerados por GPU via NVIDIA GPU Cloud, infraestrutura de nuvem pública e até mesmo estações de trabalho locais com recurso gráfico da empresa. Ele oferece uma estrutura que habilita o Docker a utilizar o recursos de GPU NVIDIA dos servidores em tempo de execução, mantendo o encapsulamento de cada contêiner conforme é representado na Figura 4.

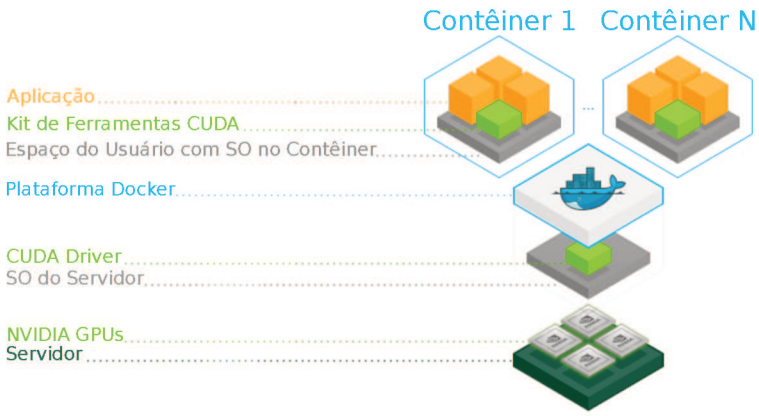

Figura 4 – Arquitetura NVIDIA Docker no servidor com recurso de GPU. Figura adaptada de (NVIDIA, 2018b).

3.4 Arquitetura sem servidor

As arquiteturas sem servidor, em inglês *Serverless*, são projetos de aplicativos que incorporam serviços FaaS, as iniciais de *Function as a Service* em inglês, de computação em nuvem que incluem execução de código personalizado em contêineres efêmeros, ou seja que tem curto tempo de execução, gerenciados em um serviço no modelo PaaS.

Ao usar a arquitetura sem servidor em conjunto de novos conceitos, como aplicativos de página única (MIKOWSKI; POWELL, 2013), removem parte da necessidade de um componente, como, por exemplo, um um servidor sempre ativo para manter o sistema utilizável. A arquitetura sem servidor pode se beneficiar pela redução significativa do custo operacional, complexidade e gerenciamento de engenharia, porém demanda maior dependência de fornecedores e serviços da nuvem (ROBERTS, 2016).

3.5 Infraestrutura como código

Infraestrutura como código, ou em inglês IaC (iniciais de *Infrastructure as Code*), é a prática de gerenciar toda a infraestrutura de uma aplicação via código, além de ter o papel de controlar todo o ciclo de configuração e implantação. Este código pode ser feito usando uma linguagem de alto nível ou qualquer linguagem descritiva, a fim de gerenciar configurações e automatizar o provisionamento de infraestrutura em ambientes virtualizados (SITAKANGE, 2016).

3.6 AWS

A AWS, iniciais de *Amazon Web Services*, é um conjunto de serviços de computação em nuvem vendido pela Amazon.com. Atualmente é a plataforma de nuvem mais usada em toda a rede mundial de computadores, oferece dezenas de serviços com finalidades diversas. Alguns exemplos incluem: ambientes computacionais virtualizados, armazenamento de dados, sistemas de bancos de dados relacionais e não relacionais, inteligência artificial, computação sem servidor, dentre outros. Serão detalhados nas próximas seções os serviços utilizados neste trabalho para chegar ao resultado final esperado.

3.6.1 Elastic Compute Cloud

O *Elastic Compute Cloud*, ou EC2, pode ser considerado o serviço mais importante de toda AWS. Ele fornece ambientes virtualizados de acordo com a necessidade do projeto, podendo variar de servidores com capacidade mínima utilizados em desenvolvimento e testes, até maquinas com alto poder computacional para suportar aplicações em produção. Instância é o nome dado para cada ambiente virtualizado no EC2 e cada tipo

varia de acordo com a necessidade de cada aplicação. As instâncias com GPU oferecem recursos gráficos em 3D de alta performance para aplicativos que utilizam OpenGL e DirectX (AWS, 2018d), como o Paraview.

Para inicializar uma Instância é necessário selecionar um tipo para a mesma, levando em conta a necessidade computacional do que será executado. Após ter sido escolhido o tipo da Instância, é selecionada uma imagem para virtualização do ambiente, denominadas AMI (iniciais de *Amazon Machine Image*) dentro do EC2. Estas AMI podem ser um sistema operacional Linux ou Windows, porém o desenvolvedor pode personalizá-la e criar uma imagem própria para seu ambiente.

3.6.2 CloudFormation

O CloudFormation é um serviço que ajuda a modelar e configurar recursos da AWS com intuito de reduzir tempo gerenciando de recursos e privilegiar o negócio das aplicações executadas na nuvem. É criado um modelo que descreve todos os recursos da AWS desejados, como instâncias do EC2, e o CloudFormation cuida do provisionamento e configuração destes recursos. Não é necessário criar e configurar individualmente os recursos da AWS e descobrir o que depende do que, o CloudFormation centraliza tudo via código (AWS, 2018b).

3.6.2.1 Pilha

Uma Pilha é um conjunto de recursos da AWS que pode ser gerenciado como uma unidade, ou seja, é possível gerenciar o ciclo de vida de um conjunto de recursos criando, atualizando ou excluindo Pilhas do CloudFormation. Todos os recursos em uma Pilha são definidos por Modelos do CloudFormation. Uma Pilha, por exemplo, pode incluir todos os recursos necessários para executar um aplicativo web, como um servidor, um banco de dados e as regras de rede para comunicação entre eles, e caso a utilização deste ambiente não seja mais necessária estes recursos podem ser excluídos com um simples comando no CloudFormation referenciando esta Pilha (AWS, 2018a).

3.6.2.2 Modelos

Para provisionar e configurar recursos de uma Pilha, é necessário compreender os Modelos do CloudFormation. Estes que são arquivos de texto formatados em JSON ou YAML. Esses arquivos descrevem os recursos e métodos necessários para inicializar, atualizar e remover uma Pilha. É possível usar o CloudFormation Designer ou qualquer editor de texto para criar e salvar os arquivos referentes aos Modelos. (AWS, 2018e)

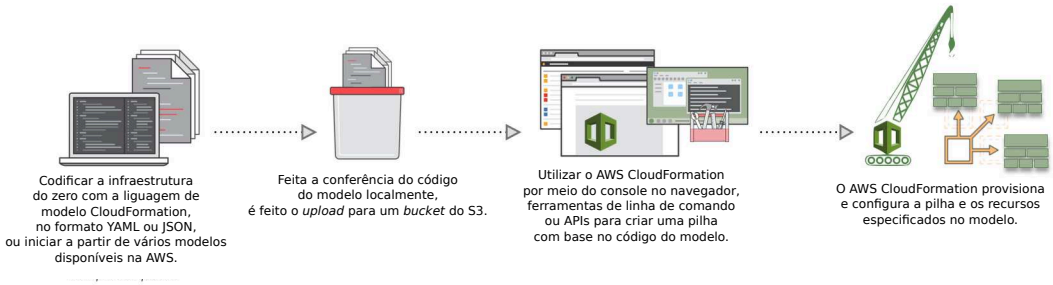

Figura 5 – Representação de processo de implantação do AWS Cloudformation. Figura adaptada de (AWS, 2018b).

3.6.3 Lambda

O Lambda é um serviço FaaS que permite a execução de códigos sem provisionar ou gerenciar servidores. O Lambda possibilita a execução de código em Node.js, Java, C#, Go e Python para praticamente qualquer tipo de aplicativo ou serviço de *Back-end* descartando a necessidade de administração de servidores. Para a execução de uma função basta carregá-la para a AWS e dimensionar o código com alta disponibilidade (AMAZON, 2015b).

Uma das maiores vantagem do Lambda é o custo, são gratuitas as primeiras um milhão de solicitações divididas em 400 mil gigabytes por segundo de tempo de computação por mês. Ou seja, o tamanho da memória que é escolhida para funções do Lambda determina por quanto tempo elas podem ser executadas no nível gratuito. Após ultrapassar este limite, as próximas um milhão de solicitações com os mesmos parâmetros são cobradas apenas vinte centavos de dólar (US$ 0,20) (AMAZON, 2015a).

Cada execução tem mínimo de alocação de memória de 128 megabytes e máximo de 3008 megabytes, caso esse limite seja ultrapassado a função é encerrada. O disco efêmero possui 512 megabytes e cada função deve executar em no máximo em 300 segundos (AMAZON, 2015a), que são limites bem confortáveis para um serviços web.

3.6.3.1 Serverless Framework

Dividir uma aplicação em várias funções resulta em várias implantações separadas, o que pode aumentar drasticamente a complexidade da implantação do sistema. O Serverless Framework é um projeto de código aberto para desenvolver e implantar aplicativos em vários provedores de nuvem com uma experiência consistente utilizando arquitetura sem servidor (SERVERLESS, 2018).

A utilização do Serverless Framework traz vários benefícios como aumentar a velocidade do desenvolvimento, pois possibilita que a codificação, os testes e a implantação aconteçam em um único ambiente de trabalho.

O primeiro passo para desenvolver um serviço sem servidor é montar um arquivo YAML referente a estrutura que deve ser construída para a aplicação. Com um único comando esse serviço pode ser implantado em um provedor de nuvem. Além de facilitar a implantação, o Serverless Framework possibilita ter controle de versão. Caso haja algum erro na implantação, é simples o processo de reversão do serviço. Outro fato importante é que, com a montagem do arquivo YAML a infraestrutura do projeto fica toda em código, facilitando o trabalho operacional (SERVERLESS, 2018).

3.6.4 AWS SDK

O termo SDK, iniciais de *Software Development Kit* em inglês, é dado para um conjunto de ferramentas, bibliotecas, documentações, amostras de código, processos e guias que permitem aos desenvolvedores criarem aplicativos em uma plataforma específica. O AWS SDK tem a finalidade de auxiliar o desenvolvedor a criar softwares para serem implantados na nuvem da Amazon, tendo várias funções para integrar o aplicativo escrito com os serviços fornecidos pela empresa.

O AWS SDK possui versões para Node.js, Java, .NET, PHP, Python, Ruby, Go e C++. Devido a grande variedade de plataformas aceitas pelo AWS SDK, quase todo tipo de software pode ser implantado na nuvem da Amazon com uma complexidade reduzida.

3.7 React

React é uma biblioteca JavaScript desenvolvida para a criação de interfaces com o usuário. O React foi criado por Jordan Walke, engenheiro de software do Facebook, influenciado pelo XHP, um sistema de componentes baseado em PHP, mas também por ideias de programação funcional. Pete Hunt, também engenheiro do Facebook, queria usar o React no Instagram, então ele ajudou a extrair o React do código específico do Facebook. Isso preparou o React para ser de código aberto (FISHER, 2015).

São criadas visualizações simples para cada estado da aplicação e o React atualiza e renderiza os componentes de acordo com a alteração dos dados relacionados a eles. As exibições declarativas tornam o código mais previsível, simples de entender e fácil de depurar (FACEBOOK, 2018).

4 Desenvolvimento

4.1 Fases do projeto

Nesta seção será apresentada a fase inicial do projeto e elaboração da arquitetura que foi proposta para implantação da aplicação na nuvem.

4.1.1 Fase Inicial

Na fase inicial do projeto foram feitos levantamentos com os objetivos de facilitar e agilizar a implantação de um sistema de análise de dados e visualizações com fins científicos utilizando Paraview. Foi vista a necessidade de utilizar recursos computacionais de alto processamento visando baixo custo. Por necessitar de respostas rápidas durante as visualizações científicas e interações com usuário, a utilização de GPU foi imprescindível.

O processamento em GPU traz um custo mais elevado ao ambiente, pois os preços cobrados nas instâncias advindas do recurso são mais altos. Visto esse impasse foi levantada a necessidade de todo os sistema ser pago por utilização, ou seja, sob demanda.

4.1.2 Arquitetura

A arquitetura proposta neste trabalho deve proporcionar um ambiente com poder computacional forte e baixo custo para os usuários finais. Além disso, deve deixar a possibilidade de expansão caso a arquitetura venha necessitar de um suporte maior de usuários ou de trabalhos com maior peso computacional.

A ideia foi de desenvolver uma estrutura em que o usuário possa ativar o ambiente com GPU quando for utilizar e desativá-lo quando o trabalho terminar. Assim, a redução de custo do ambiente na nuvem cai significativamente pois não haverá servidores ligados o dia todo sem usuários utilizando. Além disso, a interface em que o usuário vai habilitar o ambiente tem a arquitetura sem servidor, que da mesma forma é paga apenas pelo que é utilizado, evitando gastos desnecessários.

A Figura 6 demonstra como ficou a arquitetura do sistema proposto neste trabalho. O usuário acessa pelo navegador ou aplicativo móvel uma interface escrita em Javascript, que possibilita o gerenciamento do ciclo de vida de sistemas de visualização científica. A interface se comunica com serviços feitos com a arquitetura sem servidor no AWS Lambda, tendo como finalidade iniciar, colher dados e desativar aplicações cadastradas no sistema. Os serviços utilizam ferramentas que enviam comandos para o CloudFormation, no qual tem o papel de gerenciar o ciclo de vida das aplicações cadastradas dentro da nuvem da

AWS, ou seja, iniciando ambientes virtualizados e disponibilizando dados para sistema sem servidor, que devolve os dados para o usuário via interface. Após o usuário possuir os dados dos ambientes virtualizado ele pode acessa-lo, executar seus trabalhos e por fim desativa-lo pela interface e sua estrutura.

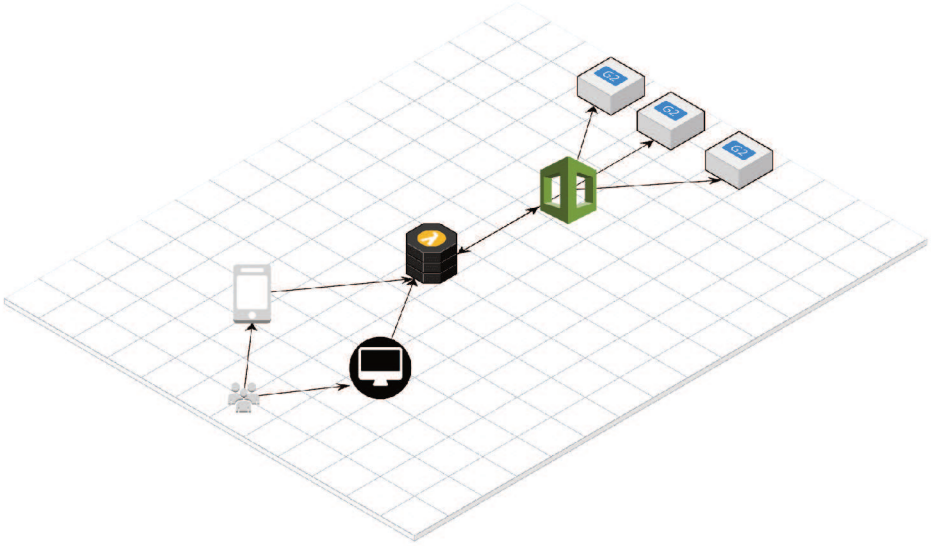

Figura 6 – A figura representa o usuário final interagindo com as funções sem servidor via uma interface JavaScript. As funções sem servidor utilizam o AWS SDK para o envio de comandos para o CloudFormation, que tem papel de executar tarefas e obtenção de dados dos ambientes disponíveis na nuvem.

5 Execução

Logo após definidas as tecnologias e métodos a serem utilizados, iniciou-se o processo de execução da implementação do sistema na nuvem. A seguir são listadas todas as fases da implementação do sistema.

5.1 Imagem Docker

Foi necessária inicialmente a implementação de um Dockerfile para executar o Kitware Visualizer em contêineres Docker. Foram desenvolvidos dois tipos de Dockerfile para implementação, com GPU e sem o recurso, assim possibilitando a execução do contêiner em todos os ambientes, até mesmo na estação de trabalho de um desenvolvedor que deseje alterar algo na estrutura proposta. Durante a implementação da imagem com GPU houve vários problemas devido a grande necessidade de dependências externas na compilação do Paraview. Para facilitar o andamento do trabalho foi utilizado um modelo proposto pela Kitware para a execução do Visualizer em Docker. Este Dockerfile é disponibilizado pela empresa via Github (KITWARE, 2018c).

5.2 Instância EC2

A experiência na qual os usuários finais aprovam, necessita de uma navegação com desempenho satisfatório. Para que a implementação do sistema de visualização científica na nuvem dê uma experiência aceitável ao usuário final, será necessário a implementação de instâncias EC2 com disponibilidade de processamento em GPU.

Foram levantados vários tipos de instâncias para execução do sistema, porém a escolhida foi a *g2.2xlarge*. Ela provém de 8 núcleos de CPU virtuais, 15 gigabytes de memória RAM e 60 gigabytes de armazenamento. O motivo da escolha desse tipo de instância foi a comparação sobre o preço cobrado durante a execução sob demanda. Enquanto o tipo de instância escolhido no trabalho custa US$ 0,65 por hora, o primeiro acima deste com disponibilidade de processamento com GPU tem o valor de US$ 1,14 por hora, cerca de 57% a mais em cobranças.

Devido a flexibilidade que a arquitetura em nuvem proporciona, a possibilidade da alteração do tipo de instância estará bem simplificada ao fim do processo de implementação. O trabalho tende sempre desacoplar ao máximo os componentes de cada camada do sistema, possibilitando futuras alterações com facilidade.

5.3 AMI

Na montagem da AMI foi utilizada a imagem base do Ubuntu Server 18.04, e foram instalados os pacotes de *drivers* NVIDIA para utilização do recurso de GPU.

O segundo passo foi a instalação do Docker para execução dos contêineres. Após a instalação dos *drivers* e do Docker foi necessário fazer com que os contêineres utilizassem o recurso de GPU dos ambientes virtualizados. Para possibilitar essa comunicação, foi feita a instalação do NVIDIA Docker.

Após todos esses procedimentos serem feitos, o console da AWS foi acessado via o serviço EC2, e foi feita a solicitação da imagem a partir da instância criada e alterada.

5.4 Modelo CloudFormation

Para montagem do ambiente na AWS foi criado um Modelo Cloudformation, descrevendo todas as necessidades e configurações obrigatórias para inicializar o Visualizer com processamento em GPU na nuvem.

No modelo é descrito o tipo de instância, a imagem a ser virtualizada e as configurações iniciais para execução do aplicativo. Durante a inicialização do ambiente foram programados comandos para baixar na rede algumas amostras de dados e colocá-las dentro da aplicação. Após tudo ser provisionado o Cloudformation reporta qual é o endereço para o acesso via navegador e SSH.

5.5 Lambda

Apesar da computação em nuvem possibilitar baixos custos, não há necessidade de um ambiente virtual ficar disponibilizado 24 horas por dia se não for utilizado grande parte do tempo. Com intuíto de diminuir custos, foram desenvolvidos serviços REST na tecnologia Node.js, com finalidade de recepcionar solicitações de clientes e refleti-las no Modelo Cloudformation.

Foram desenvolvidas quatro serviços com funcionalidade de iniciar um ambiente, buscar dados de acesso do mesmo, desativá-lo e listar Pilhas Cloudformation disponíveis por meio de requisições via protocolo REST. As funções recebem parâmetros referentes as Pilhas do Cloudformantion, e utilizando o AWS SDK executa comandos para colher informações ou executar mudanças nos recursos relacionados.

O Serverless Framework foi utilizado para facilitar o desenvolvimento e implantação das funções na nuvem, mesmo que a AWS ofereça ferramentas similares para o mesmo fim. A escolha ocorreu pela facilidade de implantar e modificar as Pilhas gerenci-

adas pelo Serverless, além de existirem vários *plugins* para o *framework* que facilitam o desenvolvimento e teste das funções em um ambiente local.

5.6 Interface de usuário

Para finalizar o desenvolvimento do projeto foi criada uma interface para interação com usuário utilizando o ReacJS. Foram desenvolvidas duas telas essenciais para utilização do usuário, a primeira delas é a de Painéis, onde o usuário consegue visualizar o estado dos ambientes disponíveis. A segunda tela é composta por três estados, os quais correspondem às funções de criação, listagem e obtenção de informações de acesso ou deleção do ambiente.

A possibilidade de usar uma única tela para gerenciar todo o ciclo de vida dos ambientes é possível graças ao conceito de criação de componentes do ReactJS. A criação de componentes reusáveis para todo o aplicativo facilita a implementação do conceito de Aplicação de Página Única, no inglês SPA ou *Single Page Application*, reduzindo o impacto durante uma expansão com novas telas ou estados para futuras funções.

6 Resultados Obtidos

Neste capítulo será demonstrado o aplicativo criado para possibilitar usuários comuns utilizarem ambientes computacionais virtualizados com GPU na nuvem. Serão também demonstradas as interações entre a interface do usuário e o ambiente sem servidor.

6.1 Painéis

A primeira tela é bem simples, ela possui apenas três painéis nos quais informam ao usuário quantos ambientes estão em execução, em alteração ou em deleção conforme apresentado na Figura 7.

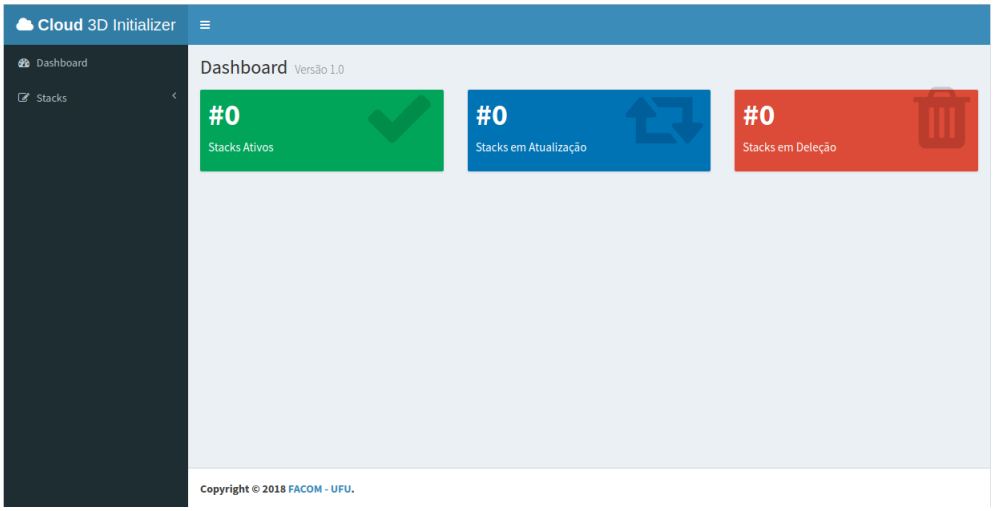

Figura 7 – Interface com usuário apresentado os Painéis.

Quando a interface do usuário faz a requisição para o serviço disponibilizado para os Painéis, é utilizado AWS SDK para resgatar as informações necessárias e enviá-las para a interface do cliente conforme apresentado na Figura 8.

```
Serverless: GET /cf/summary (λ: getSummary)
Serverless: The first request might take a few extra seconds
Serverless: [200] {"statusCode":200,"body":"{\"ativos\":0,\"emDelecao\":0,\"emAtualizacao\":0}"}
```

Figura 8 – Log referente a requisição feito a o serviço referente aos Painéis.

6.2 Ciclo de Vida dos Ambientes

Para controlar todo ciclo de vida dos ambientes foi criada apenas uma interface em forma de abas, que vão se alterando de acordo com as interações do usuário e mudanças

nas Pilhas Cloudformation.

6.2.1 Criação

A primeira aba do ciclo de vida dos ambientes é a de criação. Esta interface possui apenas um campo, no qual é informado o nome deste novo ambiente conforme apresentado na Figura 9.

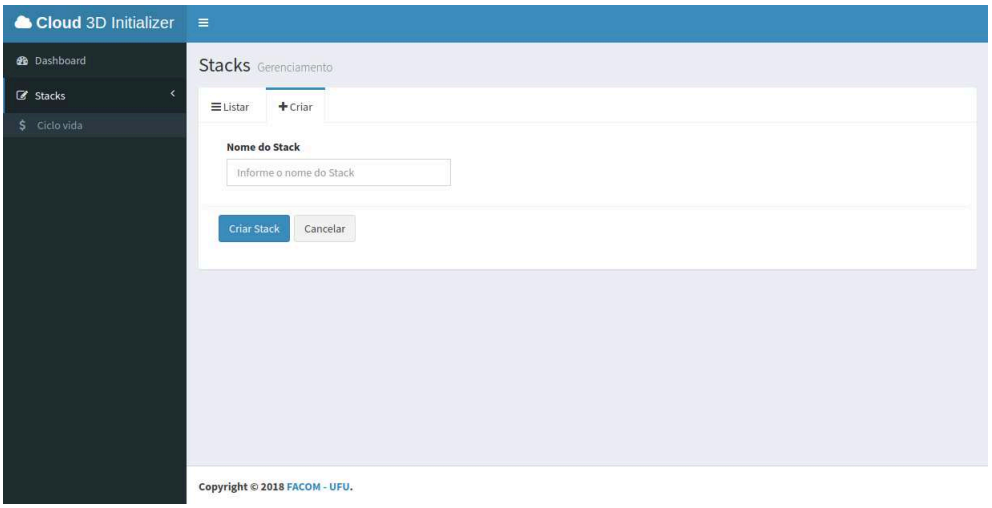

Figura 9 – Interface com usuário para criação de novos ambientes.

Após a solicitação da criação de um novo ambiente, a interface faz uma requisição para o serviço disponibilizado e apresenta uma mensagem para o usuário aguardar o processo terminar como apresentado na Figura 10.

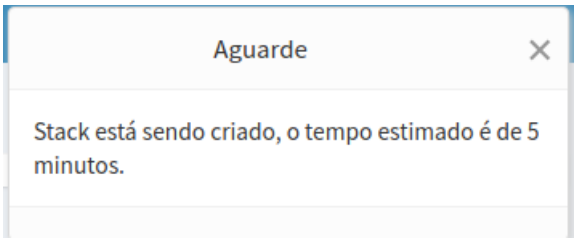

Figura 10 – Mensagem apresentada na interface do usuário devido ao processo de inicialização do ambiente.

O serviço de criação novamente utiliza o AWS SDK para repassar o Modelo Cloudformation produzido e inicializar o Visualizer na AWS, conforme apresentado na Figura 11.

```
Serverless: POST /cf (λ: createStack)
[21:01:00] Creating GPU-AmbienteTeste: AWS::CloudFormation::Stack - GPU-AmbienteTeste CREATE_IN_PROGRESS User Initiated
[21:01:05] Creating GPU-AmbienteTeste: AWS::EC2::SecurityGroup - WebServerSecurityGroup CREATE_IN_PROGRESS
[21:01:09] Creating GPU-AmbienteTeste: AWS::EC2::SecurityGroup - WebServerSecurityGroup CREATE_IN_PROGRESS Resource creation Initiated
[21:01:11] Creating GPU-AmbienteTeste: AWS::EC2::SecurityGroup - WebServerSecurityGroup CREATE_COMPLETE
[21:01:14] Creating GPU-AmbienteTeste: AWS::EC2::Instance - WebServer CREATE_IN_PROGRESS
[21:01:15] Creating GPU-AmbienteTeste: AWS::EC2::Instance - WebServer CREATE_IN_PROGRESS Resource creation Initiated
[21:01:34] Creating GPU-AmbienteTeste: AWS::EC2::Instance - WebServer CREATE_COMPLETE
[21:01:34] Creating GPU-AmbienteTeste: AWS::CloudFormation::Stack - GPU-AmbienteTeste CREATE_COMPLETE
Serverless: [200] {"statusCode":200,"body":"{\"InstanceId\":\"i-00757fb2d9865cf84\",\"PublicIP\":\"54.196.241.138\",
\"WebsiteURL\":\"http://ec2-54-196-241-138.compute-1.amazonws.com\"}}"}
```

Figura 11 – Log do serviço de criação após feita solicitação de inicialização de um novo ambiente.

Após o termino da criação do ambiente uma mensagem, semelhante a da Figura 12, é apresentada ao usuário

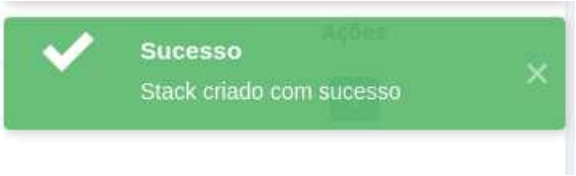

Figura 12 – Mensagem apresentada na interface do usuário após inicialização do ambiente.

6.2.2 Listagem

Quando um serviço está totalmente inicializado, a interface de listagem apresenta o ambiente criado e um botão para redirecionar para suas informações de acesso, conforme apresentado na Figura 13.

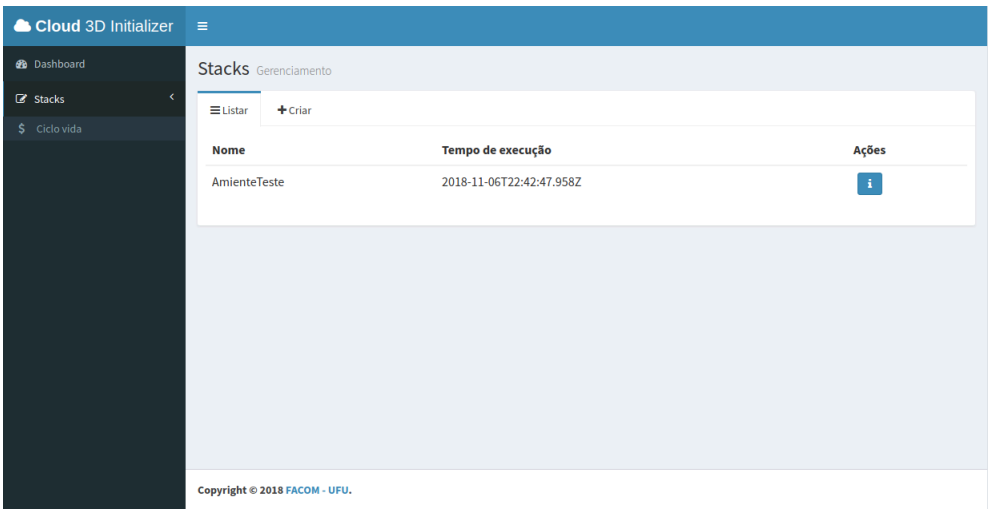

Figura 13 – Interface que lista ambientes disponíveis para utilização.

A Figura 14 apresenta o serviço de listagem respondendo a requisição REST feita para apresentar os ambientes disponíveis para utilização.

```
Serverless: GET /cf/list (λ: listStacks)
Serverless: [200] {"statusCode":200,"body":"[{\"StackId\":\"arn:aws:cloudformation:us-east-1:AWS_ACCOUNT_ID:stack/GPU-AmbienteTeste/d42bb700
-e217-11e8-a82d-50faeaee4499\",\"StackName\":\"AmbienteTeste\",\"CreationTime\":\"2018-11-06T23:01:00.671Z\"},
\"StackStatus\":\"CREATE_COMPLETE\"}]"}
```

Figura 14 – Serviço de listagem de ambiente respondendo a requisição da interface do usuário.

6.2.3 Obter Informações

Ao pressionar o botão referente às informações de um item da lista, a aba muda de estado apresentando os dados relevantes do ambiente e como acessá-lo. Na interface são apresentados dados do ambiente dentro da AWS, como ID único da Pilha Cloudformation, nome da mesma, informações do tempo de criação e estado atual da Pilha. Além desses dados, são informados o IP para acesso via SSH ao ambiente e o endereço para a aplicação no navegador.

No rodapé da página existem dois botões(Figura 15), o primeiro em vermelho faz a exclusão do ambiente apresentado, o qual será listado detalhadamente na próxima seção, o segundo em cinza tem papel de voltar a interface de listagem.

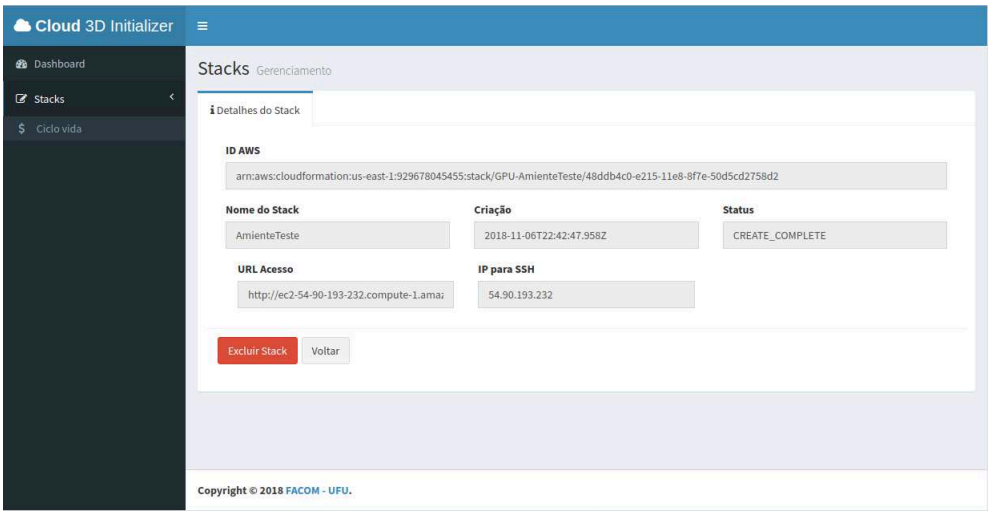

Figura 15 – Interface que tem o papel de apresentar as informações dos ambientes.

```
Serverless: GET /cf/AmbienteTeste (λ: getStack)
Serverless: [200] {"statusCode":200, "body":"{\"InstanceId\":\"i-00757fb2d9865cf84\",\"PublicIP\":\"54.196.241.138\",
\"WebsiteURL\":\"http://ec2-54-196-241-138.compute-1.amazonaws.com\"}"}
```

Figura 16 – Serviço de obtenção de informação de ambiente respondendo a requisição da interface do usuário.

6.2.4 Deleção

Ao pressionar o botão de deleção de ambiente na interface de obtenção de informações, é disparado uma mensagem para o usuário e uma requisição para o serviço de

deleção(Figura 17). Este serviço utiliza o AWS SDK para iniciar o processo de deleção da Pilha em questão(Figura 18). Após finalizar a deleção de todos os recursos, a interface apresenta uma mensagem e remove o ambiente da aba de listagem e dos painéis(Figura 19).

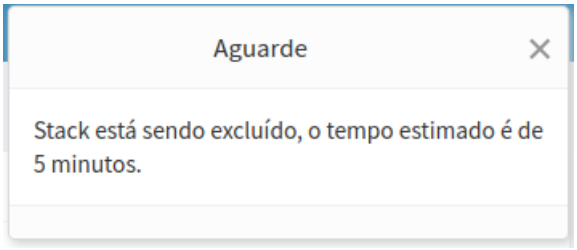

Figura 17 – Mensagem apresentada na interface do usuário devido ao processo de deleção do ambiente.

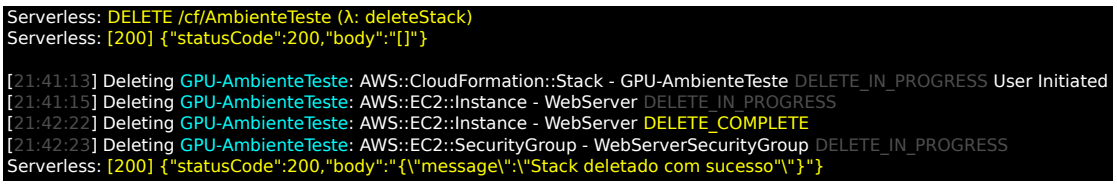

Figura 18 – Log do serviço de deleção após feita solicitação de exclusão do ambiente.

Figura 19 – Mensagem apresentada na interface do usuário após inicialização do ambiente.

6.3 Acesso ao Visualizer

Durante o ciclo de vida do ambiente, na tela de obtenção de informações, é disponibilizado um endereço o qual pode ser utilizado para acessar o Visualizer pelo navegador de Internet. Ao conectar no endereço pelo navegador, será apresentada a inicialização do Visualizer(Figura 20).

Figura 20 – Interface apresentada na abertura do Visualizer

Após a interface se conectar totalmente com servidor provisionado pelo ciclo de vida de ambiente, o navegador apresenta o Visualizer com uma paleta lateral de opções para análise de amostras científicas(Figura 21).

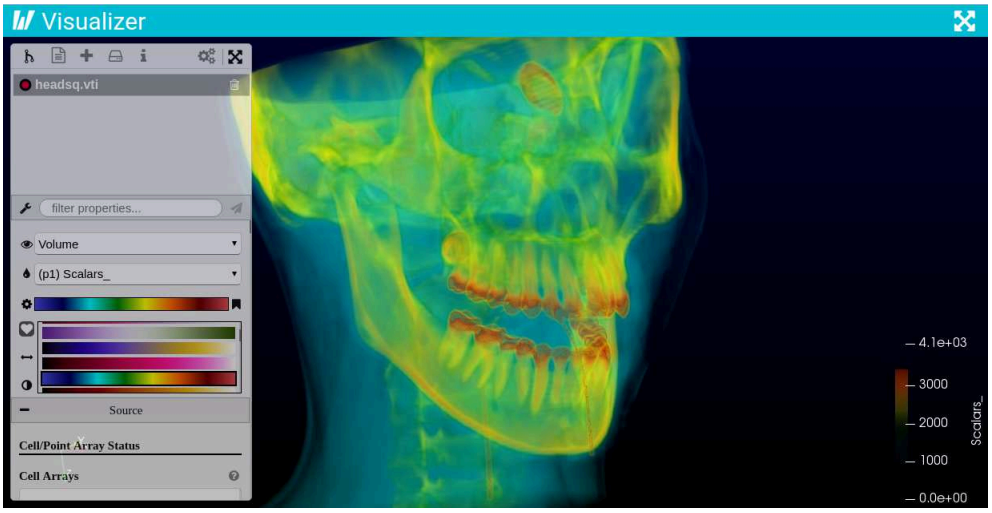

Figura 21 – Interface do Visualizer virtualizando um crânio humano.

7 Discussão

Ao fim da montagem da aplicação, gerenciado alguns ambientes e feitos testes sobre sua usabilidade, foram levantadas as vantagens e desvantagens de utilizar essa abordagem de disponibilizar serviços de visualização científica na nuvem sob demanda. Ao fim do capítulo também serão detalhadas as possíveis melhorias que podem ser feitas a partir do aplicativo desenvolvido neste trabalho.

7.1 Vantagens

Nesta seção serão demonstradas todas as vantagens levantadas após montagem e implementação do sistema na nuvem.

7.1.1 Investimento

A primeira grande vantagem vista na montagem do ambiente de virtualização na nuvem é o acesso a uma infraestrutura de alta qualidade com o custo reduzido, visto que a placa de vídeo NVIDIA GRID K520 conectada ao ambiente virtualizado neste projeto tem o valor de mercado de cerca de US$ 2.700,00. O valor pago é de 64 centavos de dólar por hora de utilização do sistema proposto no trabalho, mesmo que fosse comprado equipamento de qualidade mais baixa, o valor de investimento para aplicação na nuvem continua sendo muito mais vantajoso.

Além do acesso de qualidade a infraestrutura utilizada, a atualização destes recursos computacionais é sempre constante pelos provedores da nuvem. A arquitetura da nuvem desacopla ao máximo uma camada da outra, deixando mudanças de componentes ao longo do ciclo de vida do sistema algo com complexidade mais baixa que o cotidiano. A infraestrutura acaba tendo um custo baixo com recursos computacionais atualizados, de fácil do manutenção e atualização.

7.1.2 Elasticidade

A elasticidade do sistema dentro da nuvem é algo natural, a facilidade de adicionar e remover recurso em tempo de execução é simples. A característica elástica da nuvem além de ajudar na redução de custo, deixa o sistema robusto para enfrentar uma alta de usuários repentina.

Neste trabalho é abordada uma estrutura que tenta diminuir o custo o máximo possível, assim executando um ambiente separadamente por vez. Mas, caso a utilização

dos ambientes venha aumentar, existem outras abordagens para utilização de supercomputadores com contêineres Docker dentro da AWS (KITWARE, 2018b) (AWS, 2018c).

7.2 Desvantagens

A grande desvantagem vista nesta estrutura é a relação da infraestrutura do sistema com o provedor da nuvem. Caso fosse feita uma migração para outro provedor como Microsoft ou Google, uma força de trabalho seria inevitável, visto que detalhes como Modelo Cloudformation ou funções feitas com AWS SDK não iriam funcionar no novo ambiente. Ou seja, caso seja feita uma migração para outro provedor de nuvem, a refatoração do código fonte da aplicação será necessária.

Quando um projeto vai tomando tamanho, a utilização de recursos do provedor é muito benéfica, geralmente a camada PaaS da nuvem entrega serviços de alta qualidade com complexidade mais baixa. No entanto, a grande vinculação a esses serviços pode causar a necessidade de muitas alterações na aplicação e um momento de migração de provedor. Para evitar esses problemas a utilização de softwares de código aberto ao invés dos oferecidos pelos provedores pode ser uma solução, porém a complexidade aumentará, o que pode não ser interessante no inicio de um projeto.

8 Conclusão

É notório que a computação em nuvem traz a possibilidade de executar aplicações com várias finalidades e de qualquer grandeza. Com a implementação da aplicação de inicialização de ambientes para execução de sistemas de visualização científica, ficou claro que o projeto proposto nesta monografia foi proveitoso. Sistemas de visualização científica em ambientes com recursos computacionais impróprios podem causar lentidão, levando à desistência da utilização pelo o usuário final.

A aplicação proposta neste trabalho possibilita a execução de qualquer sistema a partir de um modelo de Infraestrutura como Código, sendo que o usuário final não precisa ter conhecimento algum sobre computação em nuvem para gerenciar o ciclo de vida dos ambientes gerenciados.

Mesmo com o objetivo do trabalho sendo atingindo, existem melhorias e novas implementações a serem feitas para evolução da aplicação de inicialização de ambientes na nuvem. São listados a seguir as possibilidades de trabalhos futuros para dar continuidade na aplicação proposta nesta monografia.

8.1 Trabalhos Futuros

Durante os estudos feitos para o desenvolvimento da aplicação e a execução do projeto proposto, foram levantadas possíveis melhorias que podem ser utilizadas em futuros trabalhos. Estas melhorias são listadas abaixo.

- O primeiro ponto de melhoria é na interface de usuário desenvolvida, apesar dela atender bem as necessidades propostas neste trabalho existem aprimoramentos que podem ser feitos. É possível evoluir a interface com a utilização de serviços da AWS tendo finalidade de autenticação, estatística de acessos, envio via e-mails informativos, entre outros recursos para melhorar a experiência final de utilização do cliente. Essa evolução pode ser rica também no uso do React, por se tratar de uma ferramenta de larga aceitação na comunidade de desenvolvedores, a interface pode ser aprimorada com técnicas avançadas de desenvolvimento de aplicações do lado do cliente.

- O aperfeiçoamento e criação de novos Modelos CloudFormation é outra área a ser explorada. Este trabalho propõem um Modelo base, no qual inicializa um ambiente contendo apenas um aplicativo de visualização científica e com apenas um tipo de instância EC2. Podem ser explorados os recursos do CloudFormation para habilitar

o usuário a personalizar mais o ambiente a ser inicializado, além da possibilidade de criação de novos Modelos para utilização de conceitos como de computação de alto desempenho e de outros softwares de visualização científica.

- Outro campo que pode ser avançado é na utilização da arquitetura sem servidor. O trabalho compõem de algumas funções para o gerenciamento do ciclo de vida dos ambientes do CloudFormation utilizando o AWS SDK e AWS Lambda. Além disso, apenas o Kitware Visualizer é utilizado como aplicação de visualização. No desenvolvimento de um software com finalidade mais específica, recursos como Banco de Dados por exemplo, podem ser necessários. O desenvolvimento deste tipo de aplicação pode ser feita em parte utilizando recursos da arquitetura sem servidor, em conjunto da implantação com Infraestrutura como Código auxiliada pelo CloudFormation.

- Por fim podem ser analisadas a troca de serviços AWS para ferramentas de código aberto, reduzindo a dependência da aplicação desenvolvida com o provedor de nuvem. A utilização de ferramentas de código aberto ajuda o desenvolvedor a explorar todos os provedores de nuvem, deixando a aplicação flexível para ser implantada a qualquer momento no ambiente que for mais vantajoso na situação atual.

- O portal Biowebvis(SILVA, 2007) é uma aplicação desenvolvida em Python que tem função de analise e visualização científica de amostras biológicas de diversos formatos pelo navegador de internet. O projeto atualmente é executado em ambiente Windows, porém sofre muito por falta de recursos computacionais adequados para sua finalidade. Durante o desenvolvimento desse projeto foram feitas várias tentativas de executar o sistema em contêineres Linux, porém todas elas sem sucesso. O Biowebvis utiliza dependências com versões descontinuadas. A evolução das dependências possibilitaria a execução da aplicação em contêineres, além da melhora da interface de usuário que podem ser pontos a serem levados em conta em trabalhos futuros. Com a evolução do Biowebvis, a implantação do sistema poderia ser facilitada junto ao projeto proposto nesta Monografia, deixando-o implantando na nuvem, com processamento mais rápido e sob demanda.

Referências

AMAZON. *Definição de preço do AWS Lambda*. 2015. <https://docs.aws.amazon.com/pt_br/lambda/latest/dg/limits.html>. Accessed:2018-06-13. Citado na página 22.

AMAZON. *O que é o AWS Lambda?* 2015. <https://docs.aws.amazon.com/pt_br/lambda/latest/dg/welcome.html/>. Accessed:2018-06-13. Citado na página 22.

AWS. *Como trabalhar com pilhas*. 2018. <https://docs.aws.amazon.com/pt_br/AWSCloudFormation/latest/UserGuide/stacks.html>. Accessed:2018-10-20. Citado na página 21.

AWS. *O que é o AWS CloudFormation?* 2018. <https://docs.aws.amazon.com/pt_br/AWSCloudFormation/latest/UserGuide/Welcome.html>. Accessed:2018-10-20. Citado 3 vezes nas páginas 5, 21 e 22.

AWS. *Orchestrating GPU-Accelerated Workloads on Amazon ECS*. 2018. <https://aws.amazon.com/pt/blogs/compute/orchestrating-gpu-accelerated-workloads-on-amazon-ecs/>. Accessed:2018-10-20. Citado na página 36.

AWS. *Perguntas frequentes sobre o Amazon EC2*. 2018. <https://aws.amazon.com/pt/ec2/faqs/>. Accessed:2018-10-20. Citado na página 21.

AWS. *Trabalhar com os modelos do AWS CloudFormation*. 2018. <https://docs.aws.amazon.com/pt_br/AWSCloudFormation/latest/UserGuide/template-guide.html>. Accessed:2018-10-20. Citado na página 21.

BREY, P. *Containers vs. Virtual Machines (VMs): What's the Difference?* 2018. <https://blog.netapp.com/blogs/containers-vs-vms/>. Accessed:2018-10-20. Citado 2 vezes nas páginas 5 e 18.

FACEBOOK. *React Github*. 2018. <https://github.com/facebook/react/>. Accessed:2018-10-20. Citado na página 23.

FISHER, B. *How was the idea to develop React conceived and how many people worked on developing it and implementing it at Facebook?* 2015. <https://www.quora.com/React-JS-Library/How-was-the-idea-to-develop-React-conceived-and-how-many-people-worked-on-developing-it-and-
answer/Bill-Fisher-17>. Accessed:2018-10-20. Citado na página 23.

GIENOW, M. *Serverless 101: How To Get Serverless Started In The Enterprise*. 2018. <https://thenewstack.io/serverless-101-how-to-get-serverless-started-in-the-enterprise/>. Accessed:2018-10-20. Citado na página 10.

GROUP, K. *OpenGL Overview*. 2018. <https://www.opengl.org/about/>. Accessed:2018-10-20. Citado na página 17.

HAT, R. *What is Docker*. 2018. <https://www.redhat.com/pt-br/topics/containers/what-is-docker>. Accessed:2018-20-04. Citado 2 vezes nas páginas 17 e 18.

INC., D. *Dockerfile reference*. 2018. <https://docs.docker.com/engine/reference/builder/>. Accessed:2018-10-20. Citado na página 19.

INC., D. *What is Docker*. 2018. <https://www.docker.com/resources/what-container>. Accessed:2018-12-03. Citado na página 17.

JOURDAIN, S.; AYACHIT, U.; GEVECI, B. Paraviewweb, a web framework for 3d visualization and data processing. In: *IADIS international conference on web virtual reality and three-dimensional worlds*. [S.l.: s.n.], 2010. v. 7, p. 1. Citado na página 16.

KITWARE. *Documentation of Visualizer*. 2018. <https://kitware.github.io/visualizer/docs/>. Accessed:2018-10-20. Citado na página 16.

KITWARE. *Launching Examples Supported "Out-of-the-box"*. 2018. <https://kitware.github.io/paraviewweb/docs/launching_examples.html>. Accessed:2018-10-20. Citado na página 36.

KITWARE. *ParaviewWeb Docker Github*. 2018. <https://github.com/Kitware/paraviewweb/tree/master/tools/docker>. Accessed:2018-10-20. Citado na página 26.

LEATHER, A. *Intel's New 28 Core Monster 5GHz Desktop Processor Is Most Powerful Ever*. 2018. <https://www.forbes.com/sites/antonyleather/2018/06/05/intels-new-28-core-monster-5ghz-desktop-processor-is-most-powerful-ever>. Accessed:2018-12-20. Citado na página 16.

MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States, 2011. Citado na página 12.

MIKOWSKI, M.; POWELL, J. *Single page web applications: JavaScript end-to-end*. [S.l.]: Manning Publications Co., 2013. Citado na página 20.

NVIDIA. *NVIDIA announces TESLA V100 with 5120 CUDA cores*. 2018. <https://videocardz.com/69378/nvidia-announces-tesla-v100-with-5120-cuda-cores>. Accessed:2018-12-20. Citado na página 16.

NVIDIA. *NVIDIA Docker Github*. 2018. <https://github.com/NVIDIA/nvidia-docker>. Accessed:2018-10-20. Citado 2 vezes nas páginas 5 e 19.

NVIDIA. *NVIDIA Docker: GPU Server Application Deployment Made Easy*. 2018. <https://devblogs.nvidia.com/nvidia-docker-gpu-server-application-deployment-made-easy/>. Accessed:2018-10-20. Citado na página 19.

ROBERTS, M. *Serveless Arquitetura*. 2016. <https://martinfowler.com/articles/serverless.html>. Accessed:2017-10-12. Citado na página 20.

SCHROEDER, W. J.; LORENSEN, B.; MARTIN, K. *The visualization toolkit: an object-oriented approach to 3D graphics*. [S.l.]: Kitware, 2004. Citado na página 16.

SERVERLESS. *Why Serverless?* 2018. <https://serverless.com/learn/>. Accessed:2018-06-13. Citado 2 vezes nas páginas 22 e 23.

SILVA, E. H. *BioWebVis - Ambiente web para citomorfometria utilizando imagens 3D*. Tese (Doutorado) — Universidade Federal de Uberlândia, https://repositorio.ufu.br/handle/123456789/20685, 8 2007. Citado na página 38.

SITAKANGE, J. *Infrastructure as Code: A Reason to Smile*. 2016. <https://www.thoughtworks.com/insights/blog/infrastructure-code-reason-smile>. Accessed:2018-10-20. Citado na página 20.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, v. 1, n. 1, p. 7–18, 2010. ISSN 1869-0238. Disponível em: <http://dx.doi.org/10.1007/s13174-010-0007-6>. Citado 6 vezes nas páginas 5, 10, 12, 13, 14 e 15.