INSTITUTO FEDERAL DE SANTA CATARINA

HELENLUCIANY CECHINEL

Escalonamento de aplicações com suporte a GPU utilizando orquestração de contêineres

São José - SC

Dezembro/2018

ESCALONAMENTO DE APLICAÇÕES COM SUPORTE A GPU UTILIZANDO ORQUESTRAÇÃO DE CONTÊINERES

Trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Ederson Torresini

São José - SC

Dezembro/2018

Helenluciany Cechinel
Escalonamento de aplicações com suporte a GPU utilizando orquestração de contêineres/ Helenluciany Cechinel. – São José - SC, Dezembro/2018-
37 p. : il. (algumas color.) ; 30 cm.

Orientador: Ederson Torresini

Monografia (Graduação) – Instituto Federal de Santa Catarina – IFSC
Campus São José
Engenharia de Telecomunicações, Dezembro/2018.

1. Palavra-chave1. 2. Palavra-chave2. 2. Palavra-chave3. I. Orientador. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Título

HELENLUCIANY CECHINEL

ESCALONAMENTO DE APLICAÇÕES COM SUPORTE A GPU UTILIZANDO ORQUESTRAÇÃO DE CONTÊINERES

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 03 de dezembro de 2018:

Ederson Torresini, Me.
Orientador
Instituto Federal de Santa Catarina

Professor Roberto Wanderley da Nóbrega, Dr. Eng.
Instituto Federal de Santa Catarina

Professor Marcos Moecke, Dr. Eng.
Instituto Federal de Santa Catarina

RESUMO

A consolidação da computação em nuvem e o desenvolvimento de aplicações baseadas em microsserviços permitem uma melhor escalabilidade de recursos e processos, garantindo a modularidade do sistema. A Coordenadoria de Tecnologia e Comunicação do câmpus São José (CTIC) possui uma cultura de melhoria contínua. Sendo assim, optou-se por uma abordagem de infraestrutura como código, que visa diminuir a redundância de processamento de recursos e a redução dos custos com computadores, utilizando a virtualização por contêineres e escalonamento de aplicações. O intuito do trabalho é aperfeiçoar serviços que já são oferecidos na nuvem privada do Instituto Federal de Santa Catarina - campus São José (IFSC-SJ), propondo a melhoria de processamento de recursos em CPU através de unidade de processamento gráfico dedicado (GPU).

Palavras-chave: GPU. contêineres. escalonamento.

ABSTRACT

The consolidation of cloud computing and the applications development based on microservices allow high scalability of resources and process assuring system modularity. The Technology and Communications Coordination of campus São José have a culture of continuous improvement. Therefore, the chosen approach was infrastructure as code that aims to reduce process resources redundancy and the cost with computers using virtualization through containers and application staggering. The purpose of this research is to improve services that already exists in private cloud of Instituto Federal de Santa Catarina, suggesting the improvement of processing of resources in CPU through dedicated graphic processing unit, GPU.

Keywords: GPU. containers. scheduling.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

O conceito de computação em nuvem vem se consolidando cada vez mais nas organizações privadas e públicas. Esse termo pode ser definido como um conjunto de recursos: como capacidade de processamento, armazenamento, conectividade, plataformas, aplicações e serviços disponibilizados pela Internet (TAURION, 2009).

A difusão da computação em nuvem possibilitou a concepção de novas arquiteturas de *software* que visam o provisionamento de recursos computacionais no qual podem executar aplicações distintas (THEISGES, 2018). No cenário tecnológico atual, a migração de arquiteturas monolíticas para microsserviços vem aumentando de forma significativa, devido a escalabilidade dos serviços, interoperabilidade entre processos e da infraestrutura sob demanda. De acordo com Carvalho e Anjos (2017), uma arquitetura baseada em microsserviços possui a vantagem de desenvolver uma única aplicação como uma suíte de serviços, cada um rodando em seu próprio processo e se comunicando através de mecanismos leves, geralmente através de métodos *Hypertext Transfer Protocol* (HTTP) (FIELDING et al., 1999). Diante do exposto, a literatura de Alves (2017) enfatiza três características fortemente presentes na computação em nuvem: escalabilidade no provisionamento ou liberação de recursos, o modelo de pagamento sob demanda de uso e a virtualização.

Segundo Kang et al. (2017), a virtualização plena é um dos serviços de maior relevância que contribuem para a existência da computação em nuvem. Geralmente dominada pelas tecnologias de máquinas virtuais com uso de *hypervisor*[1], que utilizam o *software* para emular as funcionalidades de *hardware*, permitindo que vários sistemas operacionais e aplicações diversas sejam executadas em um mesmo ambiente de *hardware físico*. Contudo, a virtualização baseada em contêineres têm surgido para substituir o conceito de virtualização por *hypervisor*. Um contêiner permite a criação de instâncias de processamentos, sendo possível obter abstração e isolamento de recursos dentro de *namespaces*[2] entre aplicações ou subsistemas. Os ambientes virtuais são criados utilizando recursos e processos presentes no núcleo do próprio sistema operacional. Portanto, a diferença fundamental entre esses dois tipos de virtualização é que o *hypervisor* emula o *hardware* ao passo que o contêiner é a virtualização de aplicações a nível de sistema operacional, agilizando a portabilidade e velocidade das aplicações. Conforme Silva (2017), o uso de contêineres é considerado como uma alternativa leve se comparada ao ambiente de *hypervisor*, pois tem como objetivo a redução de sobrecarga do *hardware*, maior tolerância a falhas em função da configuração, maior agilidade nas atualizações e correção de erros de *software*.

O Instituto Federal de Santa Catarina câmpus São José (IFSC-SJ) visa atender a comunidade acadêmica composta por professores, alunos e técnicos administrativos, oferecendo-os acesso as aplicações armazenadas local e remotamente. O IFSC-SJ detém uma estrutura de salas e laboratórios que possuem ferramentas de programação, aplicações de processamento de sinais digitais e simuladores para o uso dos docentes e discentes, dos cursos com ênfase em Telecomunicações. Recentemente, uma nova modalidade de laboratórios experimentais vem sendo estudada e discutida pela instituição de ensino: a proposta de implementação de laboratórios remotos controlados via Internet como apoio ao ensino presencial, podendo também ser usado em Ensino a Distância (EaD). Os laboratórios remotos permitem que experimentos reais de um laboratório físico sejam controlados remotamente através da nuvem privada. Diante desta realidade e da preocupação no que tange a restrição de acesso aos laboratórios físicos em determinados

[1] *Hypervisor* é uma camada de *software* entre o *hardware* e o sistema operacional.
[2] *Namespaces* é um recurso do *kernel* do *Linux* que particiona os recursos do *kernel* e permite isolamento dos recursos.

horários e outros fatores limitantes ao uso de aplicações em máquinas locais, que a Coordenadoria de Tecnologia de Informação e Comunicação (CTIC) propôs a adoção da computação na nuvem. Atualmente, o ambiente de computação é híbrido, ou seja, parte das aplicações rodam em laboratórios físicos e parte na nuvem privada. Isto é, a CTIC disponibiliza remotamente os mesmos aplicativos que são executados nas máquinas físicas. A justificativa da utilização do ambiente ser híbrido, deve-se a preocupação em relação ao armazenamento e processamento dos projetos e experimentos realizados pelos alunos na nuvem, a incompatibilidade de versões dos aplicativos e principalmente a restrição de processamento da aplicações em ambiente remoto.

Para atingir uma boa eficiência global, a CTIC vem trabalhando com o conceito de microsserviços para melhor distribuir os recursos de processamento e memória. A equipe possui a intenção de executar todos os serviços ofertados pelo IFSC-SJ através de aplicação como serviço, entregando ao usuário final somente a aplicação desejada via HTTP. Desta forma o objetivo primário do projeto é atender a maior quantidade de acessos simultâneos a essas aplicações, utilizando recursos ociosos da CTIC, como placas de vídeo e a infraestrutura em nuvem do IFSC-SJ. Portanto, este trabalho visa otimizar o escalonamento das aplicações que utilizam poder de processamento intensivo através de recursos dedicados de unidade de processamento gráfico ou placas *Field Programmable Gate Array* (FPGA). Outro objeto de pesquisa deste trabalho é a investigação do uso de GPU, ao invés de CPU, para aprimorar a experiência de uso dos aplicativos por parte do usuário final, analisando também a melhoria da eficiência do ambiente global de forma a atender uma quantidade maior de usuários na infraestrutura de nuvem.

1.1 Motivação

A computação em nuvem vem sendo bastante discutida e aderida nas empresas e instituições públicas. No IFSC-SJ, alguns fatores fizeram com que a CTIC tenha migrado parte das aplicações para uma infraestrutura em nuvem privada. Dentre esses fatores, estão a restrição de horários para alunos utilizarem as instalações físicas com recursos computacionais do Instituto Federal de Santa Catarina (IFSC), a necessidade de ter os professores presentes em sala de aula para que alunos possam usufruir dos equipamentos, bem como estudos para implantação de laboratórios remotos em parceria com o curso de Refrigeração e Climatização (RAC). Dentro dessa conjectura, a motivação do trabalho está no interesse da CTIC em ofertar computação dedicada para projetos futuros de inteligência artificial, mineração de dados, *cloud gaming* e outros. Outro interesse para o desenvolvimento do trabalho foi existirem recursos ociosos na instituição, como por exemplo GPU, que poderiam estar sendo utilizadas e agregando valor na área de pesquisa e ensino da instituição.

1.2 Objetivos

O objetivo principal deste trabalho é o de aperfeiçoar serviços que já são oferecidos na nuvem privada de contêineres do IFSC-SJ, propondo a melhoria de processamento de recursos através de unidade de processamento gráfico dedicado (GPU).

1.2.1 Objetivo Específico

- Disponibilizar um manual de como publicar aplicações através de objetos Kubernetes com suporte a GPU.

1.3 Organização do texto

O texto está organizado da seguinte forma: no Capítulo 2 é apresentado o referencial bibliográfico onde são descritos conceitos relevantes ao desenvolvimento do projeto. E no Capítulo 3 é apresentado sobre a metodologia e cronograma para a execução do trabalho no próximo semestre.

2 FUNDAMENTAÇÃO TEÓRICA

O presente capítulo discorre sobre os principais conceitos relevantes para o desenvolvimento do trabalho. O estudo bibliográfico realizado abrange desde a computação em nuvem, os tipos de implementações de serviço, virtualização, orquestração de contêineres e processamento em GPU.

2.1 Computação em Nuvem

Para Fernandez (2018), o conceito de nuvem é uma categoria de terceirização de serviços. Nos dias atuais o valor do negócio está em oferecer um produto ou serviço acessível através de uma aplicação web, ao invés de oferecer serviços de infraestrutura de Tecnologia da Informação (TI). A nuvem também permite a possibilidade de aumentar o poder de atendimento da aplicação de acordo com um custo por unidade computacional.

De acordo com Veras (2012), a computação em nuvem (do inglês *cloud computing*) é um conjunto de recursos em ambiente virtual de fácil utilização e que permite acesso remoto a *hardware*, *software*, plataformas de desenvolvimento e serviços. O termo computação em nuvem vem ganhando popularidade, entretanto a definição do termo foi definido em 1997 no meio acadêmico pelo Dr. Ramnath Chellappa durante uma palestra (SILVA, 2017).

Segundo Petcov (2018), as empresas que utilizam computação em nuvem consideram tanto os aspectos econômicos quanto técnicos. Pois o uso de aplicações em nuvem reduz a complexidade técnica da infraestrutura e agrega valor a empresa, garantindo economia em relação a investimentos de equipamentos, equipe e *softwares*. Empresas como a Amazon[1], VMware[2], Microsoft[3] e Google[4] já dispõem de serviços de computação em nuvem comumente utilizados (PEREIRA et al., 2016).

2.1.1 Características

A literatura de Mell e Grance (2011) e Dornelas e Souza (2016) citam o vasto acesso a rede, agrupamento de recursos, atendimento sob demanda, rápida elasticidade e serviço mensurável como propriedades que caracterizam a computação em nuvem. Mas autores como Körbes e Wildner (2016) defendem diferentes características como: a redução de custos na aquisição de infraestruturas das empresas, a flexibilidade de gerenciar e escalonar recursos computacionais a partir de uma demanda e a facilidade de abstração dos usuários em relação ao acesso aos serviços.

A partir da descrição de Oliveira, Junior e Santos (2011) e Mell e Grance (2011), é possível obter melhor entendimento dessas características.

- **Amplo Acesso a Rede**: os recursos estão disponíveis por meio da Internet e são acessados através de mecanismos padronizados, que permitem o uso de plataformas heterogêneas devido a versatilidade de dispositivos eletrônicos existentes no mercado. Essa característica garante que não exista um limitante físico mensurável de armazenamento, uso de processamento e acesso ao serviço disponibilizado, garantindo alta disponibilidade dos recursos.

[1] <https://aws.amazon.com/>
[2] <https://cloud.vmware.com/>
[3] <https://azure.microsoft.com/>
[4] <https://cloud.google.com/>

- **Agrupamento de Recursos**: na computação em nuvem, os recursos são agrupados a fim de atender diversos usuários e diferentes demandas do meio físico e virtual. Desta forma, recursos como processamento, memória e largura de banda de rede são alocados e realocados dinamicamente segundo a necessidade do consumidor.
- **Atendimento sob Demanda**: o serviço ofertado deve atender as necessidades de usuários, e as tecnologias adaptadas a uma demanda particular, podendo ser caracterizada portanto como sob medida.
- **Rápida Elasticidade**: é a capacidade de expansão ou delimitação de um serviço de acordo com as exigências dos clientes, sem interrupções e em tempo de execução, resolvendo a questão de variação de carga na nuvem.
- **Serviço Mensurável**: por meio do uso de métricas é possível realizar o monitoramento do serviço disponibilizado. Do ponto de vista do provedor é interessante identificar e mensurar essas métricas para elaborar diferentes modelos de negócio. Os modelos são baseados no uso do serviço e não sobre o custo do dispositivo físico. Pode-se mensurar por exemplo, o tempo de uso do serviço, transferência de dados e armazenamento de arquivos.

2.1.2 Modelo de Implantação

Neste modelo, a classificação da nuvem acontece em quatro tipos. A partir do texto de Mell e Grance (2011), as nuvens podem ser identificadas como pública, privada, híbrida e comunitária. O IFSC-SJ atualmente dispõe de uma nuvem privada que somente é utilizada pela própria organização do campus São José.

- **Nuvem Privada**: esse tipo de nuvem é caracterizada por fornecer serviços exclusivamente a um grupo de usuários de uma organização, cuja a infraestrutura é própria da instituição (VERAS, 2012). Segundo Damasceno (2015), as nuvens privadas podem ser classificadas em internas ou externas. O modelo interno possibilita que a organização tenha controle total sobre a infraestrutura, configurações de recursos, aplicativos, dados e mecanismos de segurança. Em contrapartida, o modelo externo, opta por terceirizar a implementação da nuvem privada em um provedor de serviço em nuvem já existente no mercado.
- **Nuvem Pública**: na nuvem pública, um conjunto de recursos e diretórios são disponibilizados para o público geral a partir de uma modalidade conhecida como *pay-as-you-go*, o que torna o modelo de nuvem pública economicamente viável, devido a cobrança por utilização dos serviços. (SOUZA, 2010).
- **Nuvem Híbrida**: esse modelo é composto pelas duas nuvens privada e pública. Assim, uma empresa pode armazenar dados locais e sigilosos em uma nuvem privada e fazer a transferência deles entre ambas as nuvens que se disponibilizam em aplicações e portabilidade de dados (SOUZA, 2010).
- **Nuvem Comunitária**: o ambiente de Nuvem é provisionado para uso exclusivo por uma determinada comunidade de consumidores compartilhando interesses comuns (por exemplo, requisitos de segurança e monitoramento) (SILVA, 2016).

2.2 Arquitetura da Computação em nuvem

As camadas que constituem uma arquitetura do ambiente de forma genérica são: camada de *hardware*, camada de infraestrutura, camada de plataforma e camada de aplicação.

- **Camada de *Hardware***: Braga, Silva e Barros (2012) afirma que nesta camada é realizado todo o gerenciamento e configuração dos recursos físicos da nuvem, como servidores, roteadores, *switches* e sistemas de energia.

- **Camada de Infraestrutura**: nesta camada é implementada a parte de virtualização, desta forma são criados recursos computacionais por meio de particionamento de recursos físicos usando tecnologias de virtualização por *hypervisor*, ou particionando o sistema operacional por meio de virtualização por contêineres. Esta camada permite a alocação dinâmica de recursos e armazenamento no *cluster*[5] (BRAGA; SILVA; BARROS, 2012).

- **Camada de Plataforma**: é constituída por sistemas operacionais e *frameworks* de aplicações. Esta camada tem a finalidade de facilitar a implantação de aplicações, em máquinas virtuais ou contêineres. Por exemplo, a camada oferece *frameworks* para interfaces de aplicações, para banco de dados e implementação de armazenamento (SANTOS, 2016).

- **Camada de Aplicação**: Santos (2016) e Braga, Silva e Barros (2012) consideram esta camada como a de mais alto nível, onde estão dispostas as aplicações de computação em nuvem. Isso significa que aplicações diferenciadas fazem uso do conceito de elasticidade e distribuição de carga que a nuvem oferece para atingir uma melhor eficiência e reduzir custos.

2.3 Modelo de serviço

Braga, Silva e Barros (2012) afirma que na computação em nuvem é possível aplicar um modelo de mercado a partir de uma demanda baseada em serviços. Há três categorias de serviços: *Infrastructure as a Service* (IaaS), *Plataform as a Service* (PaaS) e *Software as a Service* (SaaS), as quais estão ilustradas na Figura 1.

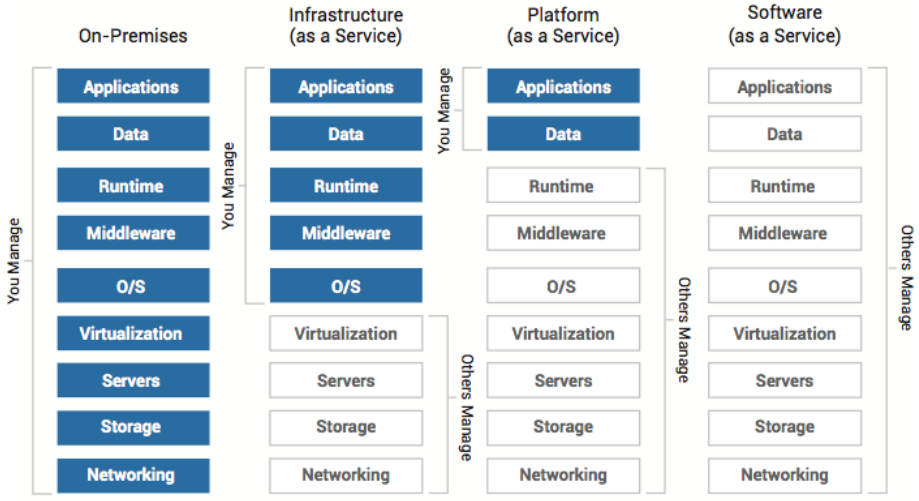

Figura 1 – Camadas de cada modelo de serviço
(FORSEC, 2016)

O modelo de serviço IaaS atua entre a camada de *hardware* e infraestrutura da arquitetura de computação em nuvem. Santos (2016) define que a infraestrutura com serviço deve prover recursos computacionais sob demanda. Portanto, o provedor do serviço deve operar no gerenciamento de processamento computacional, armazenamento, memória e rede para fornecer eficiência para o cliente. Silva

[5] Um *cluster* é um conjunto de computadores interconectados que funcionam como se fosse um só grande sistema.

(2016) complementa e caracteriza o modelo como um ambiente que possui um conjunto de recursos de computação virtualizados e hospedados em nuvem, que permite o compartilhamento dos recursos entre diversos usuários em uma mesma máquina física. A partir da Figura 1, é possível observar as camadas de responsabilidade do provedor referente aos tipos de modelos de serviços ofertados. Na infraestrutura como serviço, a responsabilidade está nas camadas de virtualização, sistema operacional, servidores e rede.

No PaaS é fornecida uma plataforma que fornece sistemas, linguagens de programação e ambientes para desenvolvimento, testes, implementação e hospedagem na nuvem (FILHO, 2012). Segundo Souza (2010), a plataforma como serviço auxilia na implementação de sistemas de *software*, já que contém ferramentas de desenvolvimento e colaboração entre desenvolvedores. No texto de Santos (2016) são mencionados alguns exemplos de serviços que são fornecidos a partir do modelo PaaS: *middleware*, *frameworks*, servidores de banco de dados e servidores de aplicação. Na Figura 1, é visível que o usuário do serviço somente tem responsabilidade acerca da camada dos dados e das aplicações - as demais camadas são atribuídas ao provedor.

O modelo SaaS consiste em disponibilizar aplicações sob demanda por meio da *Web*(BRAGA; SILVA; BARROS, 2012). Sendo assim, o provedor disponibiliza ao cliente uma aplicação que é executada em uma infraestrutura na nuvem. No texto de Alves (2017), o cliente não realiza gerenciamento ou controle da camada de infraestrutura composta por rede, servidores, sistemas operacionais e armazenamento. Conforme pode ser identificado na Figura 1, o usuário só precisa executar a aplicação na nuvem, todas as demais camadas do SaaS são responsabilidade do provedor. Silva (2017) afirma que os aplicativos são hospedados e entregues *online* através de plataforma *Web* que oferece funcionalidade tradicional, como por exemplo um *webmail*. Atualmente no IFSC-SJ algumas aplicações como o blog da CTIC já são entregues no modelo de serviço SaaS.

2.4 Virtualização

A forma de contratação de serviços virtuais entre usuários e provedores é baseada no consumo dos recursos na nuvem. A medição de uso dos recursos pode ser por tempo requerido, recursos utilizados, taxa de dados trafegados e quantidade de dados armazenados (SANTOS, 2016). Portanto, é importante o gerenciamento da nuvem e dos recursos ofertados ao clientes. Nesse contexto de IaaS, a virtualização é um conceito bem importante para que seja possível prover compartilhamento de recursos entre diversos usuários da nuvem. O IFSC-SJ possui modelo de implementação de nuvem privada e uma combinação de infraestrutura e aplicação como serviço.

O autor Oliveira (2013) descreve a virtualização como sendo uma técnica que permite a execução de diversas instâncias do sistema operacional serem executados concorrentemente em um processador. A virtualização plena segundo Silveira (2018), é um *software* que simula uma plataforma de *hardware*, sistema operacional, dispositivos de armazenamento ou recursos de rede.

A partir da literatura de Laureano e Maziero (2008), a camada de virtualização constrói uma outra interface de mesmo nível, segundo as necessidades dos componentes de sistema. Essa interface de sistema é denominada máquina virtual ou *hypervisor*. O *hypervisor* é um *software* base que gerencia o acesso ao *hardware* para as várias máquinas virtuais. Cada máquina virtual pode rodar um sistema operacional capaz de ser executado diretamente sobre o *hardware* (OLIVEIRA, 2013). O mercado atual disponibiliza algumas soluções para a implantação de máquinas virtuais, tais como: Hyper-V da Microsoft, Xen do Citrix, Virtual Box, Qemu, VMWare e outros (SILVEIRA, 2018).

Na Figura 2, é possível ver as camadas da arquitetura em nuvem com virtualização baseada em *hypervisor*. Camada de aplicação, sistema operacional visitante, *hypervisor*, sistema operacional

hospedeiro e *hardware*. A virtualização por máquinas virtuais utiliza técnicas como: virtualização total e a paravirtualização.

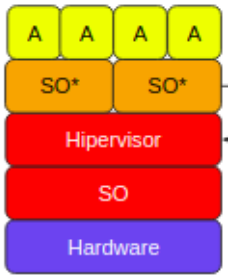

Figura 2 – Camadas da arquitetura em nuvem com virtualização por *hypervisor*.
(DRAGHICI, 2014)

2.4.1 Virtualização Plena

A técnica de virtualização total consiste em prover uma réplica virtual do *hardware* de forma que o sistema operacional e as aplicações podem executar como se tivessem executando diretamente sobre a máquina física original (CARISSIMI, 2009). Na virtualização plena, o *hypervisor* fornece uma máquina virtual completa, cuja a arquitetura é igual a da máquina hospedeira e os hóspedes podem executá-la sem ter que sofrer qualquer modificação. O *hypervisor* pode executar diferentes versões de sistemas operacionais (OLIVEIRA, 2013)

2.4.2 Paravirtualização

Laureano e Maziero (2008) afirma que a abordagem de paravirtualização permite melhorar o acoplamento entre os sistemas operacionais e o *hypervisor*, possibilitando máquinas virtuais mais eficientes. A técnica permite modificar o sistema hospedeiro para fazer uma *hypercall*[6] para a camada de *hypervisor* quando uma instrução ou ação sensível for executada (CARISSIMI, 2009). Além disso, Laureano e Maziero (2008) diz que a diferença entre a virtualização total e a paravirtualização pode ser melhor compreendida tomando como exemplo o acesso à memória: na virtualização plena, o *hypervisor* reserva um espaço de memória separado para cada sistema convidado; na paravirtualização, o *hypervisor* informa ao sistema operacional visitante quais áreas de memória é possível utilizar.

Silveira (2018) destaca que a virtualização pode ampliar e facilitar o poder computacional em nuvem, mas também pode trazer impactos negativos na sua infraestrutura quando não projetada corretamente. Hillesheim (2018) ressalta que mesmo com todos os benefícios que o uso de virtualização por máquinas virtuais oferece, a mesma vem sendo evitada em alguns cenários devido a grande sobrecarga em seu desempenho, conhecido como *overhead*.

2.5 Contêiner

A virtualização baseada em contêineres permite armazenar bibliotecas necessárias para executar uma aplicação, compartilhando o *kernel* com o hospedeiro, o que torna esse tipo de virtualização mais leve e eficiente (SILVA, 2017).

De acordo com o projeto do Docker[7], 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

[6] É uma chamada de sistema diretamente para o *hypervisor*.
[7] <https://www.docker.com/>

confiável, de um ambiente de computação para outro. Na literatura de Alves (2017), a empresa RedHat define contêineres como mantenedores de aplicações, possibilitando o isolamento delas por meio de um método de implantação baseado em imagens.

A diferença entre virtualização por contêineres e *hypervisor* pode ser observada na Figura 3. As máquinas virtuais incluem a aplicação, as dependências, os arquivos binários e sistema operacional visitante. Em contrapartida, os contêineres compartilham entre si o *kernel* do sistema operacional hospedeiro, não precisando ter uma camada de sistema operacional visitante (ALVES, 2017).

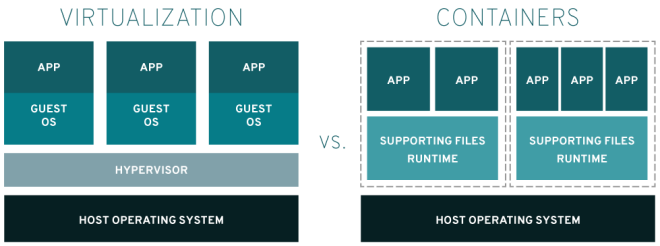

Figura 3 – Diferenças entre arquitetura de uma máquina virtual e de um contêiner. (REDHAT, 2018)

- **cgroups**: a literatura de (SILVA, 2017), discorre de *cgroups* ou grupo de controle, como uma implementação do *kernel* do Linux utilizada para implementar limitação do uso de recursos como: memória, CPU, leitura e gravação em disco ou rede, sendo o recurso de memória um dos mais impactantes no desempenho de uma aplicação.
- **namespaces**: para Alves (2017), *namespaces* permitem que um processo tenha um conjunto de usuários, inclusive com privilégios e seja restrito ao contexto de seu próprio contêiner. O trabalho de SILVA (2017) reafirma que *namespaces* podem ser utilizados para isolamento de processos em níveis de grupos diferentes.

Uma das principais plataformas de contêineres é o Docker, pois ele provê uma *Application Programming Interface* (API) simples para manipulação e isolamento de processos. Uma vez utilizando contêineres para separar os serviços de uma aplicação, é necessário coordenar a instanciação, escala e conexão destes contêineres (FILHO, 2016). Esse gerenciamento de contêineres pode ser feito através de várias ferramentas dentre elas, o Kubernetes.

2.6 Orquestração de Contêineres

Uma plataforma de orquestração de contêineres, permite gerenciar e coordenar o funcionamento de ambientes em nuvem, de aplicações baseadas em múltiplos contêineres. Em linhas gerais, a orquestração tem a preocupação com dois ou mais recursos computacionais, enquanto uma plataforma de execução está focada em um único recurso computacional (FILHO, 2016; ALVES, 2017).

2.6.1 Kubernetes

A empresa Google utiliza contêineres há alguns anos e lançou o sistema Kubernetes para orquestrar contêineres em nuvem (NETTO et al., 2016). De acordo com Kubernetes (2018), o Kubernetes é um sistema de código aberto para gerencia de aplicativos rodando em contêiner em vários *hosts*. Para RedHat (2018), essa ferramenta de orquestração permite gerenciar contêineres em escala, tornando viável o escalonamento e gerência de integridade dos contêineres com o passar do tempo.

Através de uma *Application Programming Interface* (API), são criados objetos Kubernetes para especificar o tipo de aplicações, cargas de trabalho, imagens de contêineres, número de réplicas, recursos de disco e outras opções que auxiliam na manutenção do estado de um *cluster* (THEISGES, 2018). Depois da configuração realizada, o Painel de Controle do Kubernetes trabalha para garantir que o estado do *cluster* em execução seja equivalente ao estado configurado. Para isso, as ações de inicializar, reiniciar contêineres e escalar o número de réplicas de uma determinada aplicação são automáticas (HILLESHEIM, 2018).

Um *cluster* gerenciado pelo Kubernetes possui duas principais unidades operacionais: mestre e nós. Os componentes mestres disponibilizam os serviços por meio de uma API e são responsáveis por executar tarefas relacionadas ao escalonamento, agendamento e atualizações de aplicações (KUBERNETES, 2018). Nesses componentes mestres são executados os seguintes processos conforme consta na documentação do (KUBERNETES, 2018):

- **kube-apiserver**: processo servidor que visa validar e configurar dados para os objetos da API, que incluem *pods*, serviços, controladores de replicação e outros.
- **kube-controller-manager**: esse processo é o gerenciador controlador do Kubernetes, ou seja, um *loop* de controle que monitora o estado compartilhado do *cluster* através do processo apiserver e faz alterações para que o estado atual chegue o mais próximo do estado desejado.
- **kube-scheduler**: um dos processos mais relevantes para este trabalho, pois a partir de requisitos de recursos individuais e coletivos, requisitos de qualidade de serviço, restrições de *hardware*, *software* e políticas de afinidade, que é selecionado um nó físico específico para execução de uma aplicação solicitada por um usuário.

Os componentes dos demais nós são executados com o intuito de manter os *Pods* em execução e fornecer o ambiente de tempo de execução do Kubernetes. Os nós que não são mestres executam os processos:

- **kubelet**: para garantir que os contêineres estejam sendo executados em um *Pod*.
- **kube-proxy**: definido por ser o *proxy* de rede com o propósito de encaminhamento das conexões.

2.6.2 Pods

Quando um novo serviço é criado no *cluster*, o mestre insere contêineres nos nós, em unidades denominadas *Pods*. Para essa inserção são considerados fatores como requisitos de *hardware* de aplicação e carga de cada nó (ALBUQUERQUE et al., 2018)

Kubernetes (2018) define *Pod* como sendo um processo em execução no *cluster* e uma unidade mínima de implantação do Kubernetes, que possui uma única instância e pode conter uma aplicação com mais de um contêiner, recursos de armazenamento e outras opções que permitam controlar os contêineres.

Na literatura de Albuquerque et al. (2018), os *Pods* podem ser vistos como "caixas" que comportam grupos de contêineres e que devem trabalhar em conjunto para entregar um serviço. Por exemplo, em um servidor Web que necessita de um banco de dados, um *Pod* poderia conter dois contêineres (um para o servidor Web e outro para o servidor de banco de dados).

Os *Pods* são vistos externamente como um entidade única, pois dividem o mesmo domínio de rede e compartilham portas como se fossem uma única máquina. Os autores de Kubernetes (2018) mencionam três tipos de Controladores de *Pods*: *Deployment*, *StatefulSet* e *DaemonSet*.

- **Deployment:** permitem criar, gerenciar, monitorar e atualizar conjuntos de *pods* idênticos para prover um determinado serviço. É responsável por criar objetos que controlam de número de *pods* a serem criados, número mínimo de *pods* disponíveis, quantidade de *pods* que pode ser instanciados entre outros parâmetros (ALBUQUERQUE et al., 2018; KUBERNETES, 2018).

- **StatefulSet:** é o objeto da API usado para gerenciar aplicações com estado. Cada *pod* tem um identificador persistente que é mantido em qualquer reprogramação realizada (KUBERNETES, 2018).

- **DaemonSet:** garante que os nós executem uma cópia de um *Pod*. Conforme acontece a adição de nós ao *cluster*, os *Pods* são adicionados aos nós. E quando os nós são removidos do *cluster*, esses *Pods* são coletados como lixo, limpando os *Pods* criados.

2.6.3 Serviços

De acordo com Hillesheim (2018), os *Pods* são criados e destruídos a todo momento, e quando removidos nunca mais são recuperados. Sua criação e destruição é feita dinamicamente pelas *ReplicaSets*. Para solucionar tal problema, Albuquerque et al. (2018) afirma que os serviços Kubernetes podem tornar *Pods* visíveis de algumas formas e Theisges (2018), cita a exportação de portas de um contêiner como alternativa para que a aplicação que está sendo executada no *Pod* fique visível ao usuário.

2.7 Processamento Dedicado

Nos primórdios da computação, os sistemas possuíam apenas um núcleo de processamento e trabalhavam na forma sequencial de instruções. A cada nova geração de processadores, era notório o crescimento no desempenho dos *softwares*, o que motivava cada vez mais os desenvolvedores a aperfeiçoar suas aplicações, tornando-as mais robustas e acompanhando o poder computacional. Entretanto, as CPUs não conseguiram acompanhar o processo de crescimento do volume de dados e a crescente demanda por processamento dedicado (DOMINGOS, 2012).

Desta forma, abriu-se frente a uma nova forma de se realizar computação mais eficiente: através do paralelismo computacional. Para Jradi (2016), a computação paralela é uma técnica, onde múltiplas instruções são simultaneamente executadas utilizando diferentes unidades de processamento. A partir da documentação de Mourelle (2011), sistemas multiprocessadores contêm dois ou mais processadores com capacidades aproximadamente idênticas. Cada processador possui a sua memória local e dispositivos próprios, entretanto, a comunicação entre processadores se faz através das memórias compartilhadas ou de uma rede de interrupção.

Pela pesquisa de Jradi (2016), na década de 1990, Flynn propôs uma classificação para sistemas de computadores que acabou se tornando o padrão e ainda é amplamente utilizado até hoje. Ela é baseada na multiplicidade ou não do fluxo de instruções e de dados em um computador, conforme é ilustrado na Figura 4.

Flynn dividiu os computadores em quatro classes. Com apoio da literatura de Mourelle (2011), Jradi (2016) é possível descrever melhor essa classificação:

- *Single-Instruction Single-Data* (SISD): essa categoria é caracterizada por ter fluxo único de instrução e de dados. As instruções são executadas de forma sequencial, como por exemplo as máquinas de Von Neumann.

- *Single-Instruction Multiple-Data* (SIMD): possui fluxo único de instrução, no entanto, tem múltiplos fluxos de dados. Corresponde às máquinas paralelas que têm múltiplos elementos processadores supervisionados pela mesma unidade de controle, mas operam sobre diferentes conjuntos de dados.
- *Multiple-Instruction Single-Data* (MISD): tem fluxo múltiplos de instrução e o fluxo único de dados. Unidades processadoras que recebem instruções distintas, entretanto, operam sobre o mesmo conjunto de dados. Na prática não há máquinas dessa categoria viáveis.
- *Multiple-Instruction Multiple-Data* (MIMD): é conhecida por possuir múltiplos fluxos de instruções e de dados. Os MIMD possuem uma unidade de controle para cada processador e, portanto, podem executar instruções diferentes em diferentes conjuntos de dados. Corresponde aos sistemas multiprocessadores.

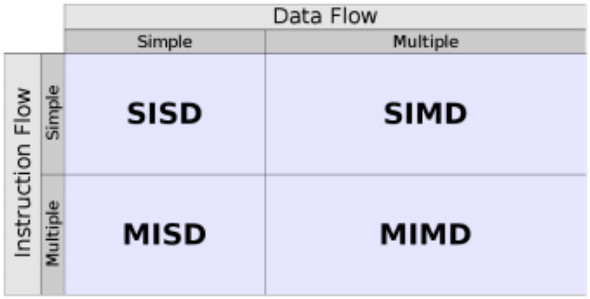

Figura 4 – Classificação de Flynn para sistema de computadores.
(JRADI, 2016)

2.8 Processamento em GPU

A unidade de processamento gráfico (GPU) em essência consiste em várias unidades especializadas para processamento de operações com ponto flutuante, que são bastante usadas para aceleração de renderização gráfica de informações bidimensionais e tridimensionais. Visto que a renderização precisa computar de forma independente os mesmos algoritmos para cada pixel, a GPU possui uma arquitetura distinta para essa computação paralela (JRADI, 2016; KANG et al., 2017).

Segundo, BREDER (2016), o avanço da capacidade de processamento da GPU tornou possível o processamento de outros tipos de estrutura de dados, tornando-a excelente alternativa para realizar computação paralela com um custo baixo. Sendo assim, recentemente a GPU vem sendo utilizada não somente para renderização gráfica, mas também para propósitos gerais em varias áreas da pesquisa como: inteligência artificial, jogos, mineração de dados, genoma humano, ressonância eletromagnética, e diversas outras áreas em que são necessários reconhecimentos de padrões de imagens. (DOMINGOS, 2012).

2.8.0.1 Arquitetura de GPU

Como já mencionado, a GPU é formada por inúmeros processadores, organizados e projetados de forma a tornar ótimo o desempenho de paralelismo. Sua organização e forma de processamento leva à uma abordagem chamada SIMD (DOMINGOS, 2012). Segundo BREDER (2016), a GPU dedica mais o circuito na *Arithmetic Logical Units* (ALU) (*Arithmetic Logical Units*), o que a torna muito mais eficaz para cálculos matemáticos.

É possível observar na Figura 5, que a GPU tem um circuito bastante limitado de unidade de controle e memória, que está representado pelas pequenas caixas amarelas e laranjas. Mas para aplicações que demandam o uso de unidades lógicas aritméticas, o processamento em GPU é uma vantagem.

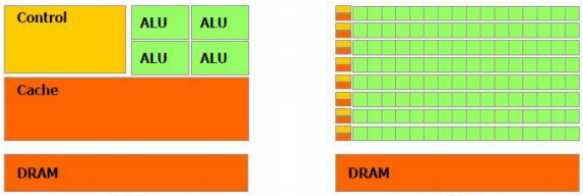

Figura 5 – Diferença entre arquitetura CPU e GPU.
Fonte: (BREDER, 2016)

SILVA (2013) também contribui para discussão da Figura 5. Ele afirma que a grande diferença no desempenho de CPUs e GPUs pode ser atribuída às diferenças existentes na arquitetura de ambas. Pois CPUs visam obter alta performance na execução de código sequencial. Em contrapartida, as GPUs são projetadas para execução paralela de instruções, sendo otimizadas para processar operações sobre vetores.

Toda GPU deve estar acoplada a uma CPU, pois precisa do controle da CPU para iniciar seu processamento Puc-Rio (2018). Nesse contexto, a CPU é designada como *host* e a GPU como dispositivo. O processamento de dados e código precisa ser enviado da CPU para GPU através do barramento principal para ser executado, e após a realização do cálculo, os dados resultantes são retornados para CPU por meio da memória principal, conforme Figura 6 (JRADI, 2016).

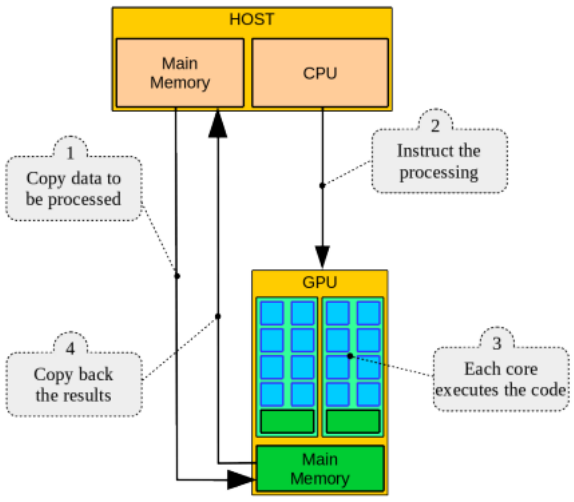

Figura 6 – Fluxo do processamento em GPU.
(JRADI, 2016)

A partida da literatura de Jradi (2016) e Puc-Rio (2018), os principais elementos do *hardware* da GPU são o *Stream Processor* (SP), o *Streaming Multiprocessor* (SM) e as distintas memórias disponíveis.

- SP: é conhecido também pela nomenclatura da Nvidia como núcleo *Compute Unified Device Architecture* (CUDA)[8], que são unidades de computação com capacidade de execução de algoritmos de alta eficiência. Duas partes cruciais para um processador de fluxo são a unidade lógica aritmética e a unidade de ponto flutuante.

- SM: são blocos multiprocessados, fortemente agrupados construídos por processadores de fluxos. Pela pesquisa em Puc-Rio (2018), o multiprocessador foi projetado para criar e executar *threads* concorrentes com um custo mínimo devido a *overhead* e sincronização. Cada SM possui vários

[8] Plataforma de computação paralela desenvolvida pela NVIDIA.

núcleos chamados de núcleo CUDA. E cada um dos núcleos CUDA possui um pipeline de operações aritméticas ALU e de ponto flutuante (BREDER, 2016).

- *Memory Hierarchy:* a GPU possui um espaço de memória própria, organizado hierarquicamente, e quando bem explorada, pode ser um elemento importante para o bom desempenho da aplicação (BREDER, 2016). Segundo o trabalho de Jradi (2016), existem quatro tipo de memória a serem trabalhadas em GPU. A **memória global** que é a memória principal e está disponível para todos os SPs para leitura e gravação. A **memória constante**, que está fisicamente localizada na memória principal do dispositivo. A **memória local** permite ser lido ou escrito por todos os SPs somente se dentro de um SM. E por fim, a **memória privada** que é restrita a utilização somente dentro de cada SP durante a execução do programa.

2.9 Ferramentas de Programação para Computação Paralela

Atualmente existem diversas APIs que possuem suporte a paralelismo no mercado, como por exemplo: Vulkan, Metal, *Open Graphics Library* (OpenGL), *Open Computing Language* (OpenCL), CUDA e outras. A escolha de um *framework* para implementação de computação paralela é um passo importante para o desenvolvimento de qualquer sistema computacional multiprocessado (FERREIRA, 2016). Pelo fato das placas de vídeo GPU disponíveis no IFSC-SJ serem da fabricante *Nvidia Corporation*, foram escolhidas duas plataformas para possível utilização no trabalho.

2.9.1 OpenCL

Inicialmente proposto pela Apple e depois desenvolvido pelo grupo Khronos, o OpenCL é uma plataforma de código aberto compatível com distintos tipos de arquiteturas e que possui uma abstração de baixo nível do *hardware* (PINTO, 2011; FERREIRA, 2016). A plataforma possui um ambiente de execução, uma API e bibliotecas OpenCL. É uma ferramenta que está em constante crescimento e inclui arquiteturas de processamento como CPU, GPU e FPGA, sendo bastante utilizada para fins de aceleração gráfica.

2.9.2 CUDA

É uma plataforma proprietária, desenvolvida pela nVidia para desenvolver computação paralela. Possui uma API e modelos de programação específicos para acelerar aplicações com GPU (BRAZ, 2015). A plataforma CUDA tem duas interfaces de programação: *Driver API* e *Runtime API*. A primeira permite configurar diretamente a placa GPU, controlar a troca de contexto, carregar módulos e bibliotecas. Já a segunda permite um nível de abstração mais alto, o uso de códigos para indicar quais funções serão executadas com GPU ou em CPU. Essa interface viabiliza a utilização e integração de GPU com aplicações já existentes, de uma maneira mais prática e fácil. (KANG et al., 2017). CUDA é baseada em uma extensão da linguagem C e possui bibliotecas com suporte C/C++ e Python. (NVIDIA, 2015a).

3 PROPOSTA

O intuito da CTIC do IFSC-SJ é migrar todos os serviços disponíveis aos usuários para a nuvem privada. A intenção é entregar à comunidade acadêmica todos os serviços oferecidos pela CTIC diretamente por aplicações *Web*, ou seja, o usuário só precisaria acessar o navegador e inserir a *Uniform Resource Locator* (URL) do serviço que está disponível. O blog[1] e a Wiki[2] já funcionam em um modelo de serviço de SaaS. Outro modelo de serviço implantado pela CTIC é o IaaS, pois provê toda uma infraestrutura em nuvem privada para rodar aplicações como: Matlab, Octave, Quartus. Atualmente o *cluster* da instituição conta com somente processamento em CPU, ou seja, não possui um ambiente que realize processamento intensivo com auxílio de GPU. Diante do exposto, o presente trabalho visa suprir uma demanda latente, e disponibilizar o serviço de escalonamento de aplicações com suporte a GPU utilizando orquestração de contêineres, para à comunidade acadêmica.

A proposta consiste em trabalhar desde a camada de hardware até as aplicações presentes nos contêineres. Na camada de *hardware* estarão dispostas as placas GPU da Nvidia em alguns servidores físicos do *cluster*. Na camada de *software* será trabalhada a configuração dos *drivers* CUDA ou OpenCL, e a integração deles com o sistema operacional hospedeiro. Acima dessa camada, estará disposta uma ferramenta de virtualização baseada em contêineres. A Nvidia disponibiliza um *plugin* para o docker, chamado `nvidia-docker`, onde é possível fazer o armazenamento e isolamento em contêiner, de aplicativos acelerados e implementá-los em qualquer infraestrutura que tenha suporte a unidade de processamento gráfico. Cada contêiner possui em seu ambiente de usuário uma camada de sistema operacional, bibliotecas que permitem a instalação e execução da plataforma CUDA ou OpenCL, para realizar códigos que acelerem GPU em determinadas aplicações. Por fim, a orquestração dos contêineres por meio da ferramenta de Kubernetes será a etapa de finalização da implantação proposta e que terá um foco maior no trabalho. Através de objetos Kubernetes serão gerenciadas as solicitações de multiusuários a recursos GPU disponíveis na nuvem, a partir de uma política de escalonamento por afinidades de grupo de máquina.

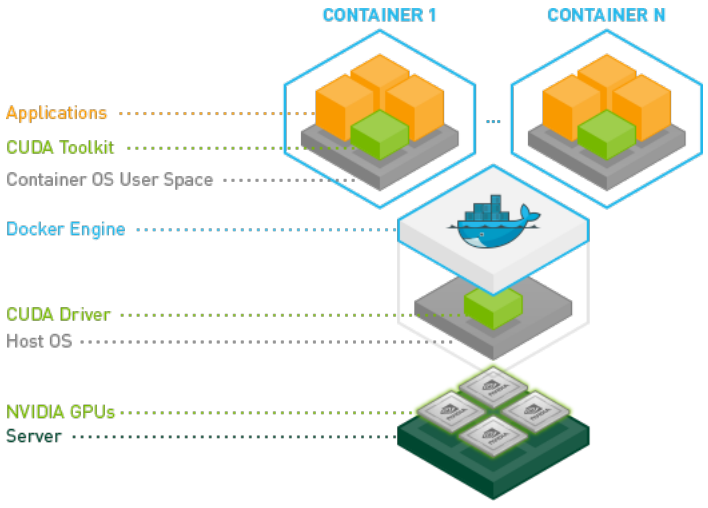

Figura 7 – Camadas de implantação da solução.
(NVIDIA, 2015b)

[1] <https://wordpress.sj.ifsc.edu.br/ctic-sje/>
[2] <https://wiki.sj.ifsc.edu.br>

3.1 Ferramentas

Abaixo estão dispostas algumas das ferramentas e aplicações que serão utilizadas para o desenvolvimento do trabalho.

- CUDA
- Docker
- Docker-nvidia
- Kubernetes
- Para validação de cenários serão utilizadas outras ferramentas de apoio ao trabalho, como: Octave, Python e TensorFlow.

3.2 Metodologia

Para o bom desenvolvimento do projeto, se faz necessário a especificação da metodologia do trabalho. A metodologia consiste em 5 etapas fundamentais, as quais estão dispostas nos itens abaixo.

3.2.1 Levantamento da Infraestrutura Atual

Nesta etapa são analisados os serviços que são fornecidos pela CTIC à comunidade acadêmica. Após análise inicial, surge a necessidade de identificar quais desses serviços requerem intensivo poder de processamento e memória. A partir dos dados obtidos, é feita uma avaliação de quais aplicações poderiam obter suporte ao uso de processamento gráfico dedicado para garantir maior eficiência de processamento global. Estudos de melhoria das aplicações *Web* no quesito eficiência já foram pensados, entretanto, aplicações como as de *desktop* que demandam uso intensivo de CPU, ainda não foram estudados.

No primeiro momento é realizado um mapeamento da infraestrutura da nuvem privada atual do IFSC-SJ. É contabilizado o número de equipamentos ativos e ociosos que são relevantes ao projeto proposto, como por exemplo servidores e placas de vídeo dedicadas. Também é estudada a especificação de cada equipamento, buscando informações de capacidade de processamento, quantidade de núcleos, memória, eficiência energética, largura de banda e sistema operacional.

Para obter melhor compreensão da estrutura de nuvem privada implementada pela CTIC, uma análise da arquitetura e dos serviços ofertados à instituição é realizada por meio de entrevista com membros da equipe e com leitura de documentação dos equipamentos que compõe a estrutura da nuvem do IFSC-SJ, para que assim, seja possível entender melhor o cenário real e levantar os requisitos para o trabalho.

3.2.2 Levantamento de Requisitos

Esta parte da metodologia é caracterizada pela análise de requisitos do projeto, identificando as métricas que definem o bom desempenho e funcionamento do sistema já existente e elencando as possíveis restrições de implementação da solução proposta. Essa etapa possui a finalidade de definir esses indicadores para melhor atender a necessidade dos usuários da nuvem privada. Dentre essas métricas podem ser destacadas o desempenho de processamento gráfico, desempenho de CPU, memória e outros valores que possam contribuir para definição de uma política de escalonamento dessas aplicações.

Também é proposto um estudo das ementas das disciplinas do curso de Engenharia de Telecomunicações, buscando encontrar as aplicações que são mais executadas na infraestrutura de nuvem e que

possuem maiores números de chamados em aberto, pela decorrência de atraso de execução. Tendo em vista a adição de novos componentes à estrutura do IFSC-SJ, o levantamento dos requisitos é realizado em conjunto com a equipe da CTIC do IFSC-SJ, afim de remodelar a estrutura da nuvem privada.

3.2.3 Estudo de Ferramentas para Implantação

Esta etapa do trabalho consiste no aprofundamento do estudo de ferramentas e tecnologias utilizadas para a implementação do escalonamento de aplicações. Portanto, o projeto demanda que sejam feitas leituras conceituais sobre definição e funcionamento de contêineres, escalonamento de processos, aplicações em microsserviços, distribuição de carga, processamento gráfico dedicado e como funciona a concorrência de uso de uma mesma GPU por vários usuários. Além desses estudos conceituais, são realizados laboratórios para melhor compreender o comportamento de tecnologias como *Docker*, *Docker-nvidia*, CUDA e *Kubernetes*.

3.2.4 Implantação da Solução

O trabalha consiste em aplicar na prática um conjunto de ferramentas e tecnologias para obtenção de maior eficiência no uso de recursos de processamento das aplicações dos usuários finais que utilizam a nuvem privada do IFSC-SJ. Sendo assim, surge a demanda para configuração do *driver* de placa de vídeo da nVidia em que o mesmo deve ser compatível com a imagem do sistema operacional existente. Também é necessário definir e implantar uma plataforma responsável pela aceleração gráfica em aplicações, podendo ser CUDA ou OpenCL. Após realização das atividades descritas acima, se faz indispensável a instalação e execução do *Docker Engine* para orquestrar contêineres com suporte a GPU nVidia.

Como etapa final, é proposto estabelecer uma política de escalonamento baseada por afinidade de máquina ou grupo de máquina com suporte a aceleração gráfica, utilizando *Kubernetes*. Esse escalonamento permite distribuir a carga entre recursos disponíveis em um *Cluster*, otimizando o processamento na nuvem privada da instituição. A política de escalonamento das aplicações depende da disponibilidade de recurso de processamento e memória. Levando em consideração as métricas levantadas na etapa de levantamento de requisitos. Métricas de rede são desconsideradas para elaboração dessa política, pois não está sendo tratado de nuvem multi-região.

3.2.5 Testes e Validação

Posterior ao processo de implantação, tem-se a demanda para realização de testes de desempenho e funcionamento da nuvem privada, para garantir que o serviço de escalonamento de aplicações com uso de GPU utilizando contêineres, atenda às necessidades da comunidade acadêmica. As verificações do funcionamento da solução proposta consistem em teste de aplicação, *benchmarking*, testes em nuvem com inserção de erros para verificar a tolerância a falhas do serviço e testes de replicação em outros nós, testando a migração da aplicação para outras máquinas.

Para finalizar a parte de testes, é preciso elaborar códigos que explorem o potencial das aplicações que possibilitem o uso de computação paralela, instigando a utilização de recursos dedicados. Portanto, para auxílio na criação desses testes, se faz necessário a assessoria de um professor com experiência na área de processamento de sinais e renderização. Além dos testes de validação, são elaborados questionários para avaliar a experiência do usuário final em relação ao tempo de resposta das aplicações local e remotas. Desta forma, é possível agrupar dados que justifiquem a necessidade de processamento com uso de GPU para aplicações utilizadas em disciplinas do cursos de Telecomunicações.

3.3 Cronograma

	Fev	Mar	Abr	Mai	Jun	Jul
Levantamento da Infraestrutura Atual	X					
Levantamento de Requisitos		X				
Estudo de Ferramentas para Implantação		X	X			
Implantação da Solução Proposta			X	X	X	
Testes e Validação				X	X	X

REFERÊNCIAS

ALBUQUERQUE, F. et al. Integração de replicação máquina de estados no kubernetes. *Workshop de Testes e Tolerância a Falhas (WTF_SBRC)*, v. 19, 2018. ISSN 2595-2684. Disponível em: <http://portaldeconteudo.sbc.org.br/index.php/wtf/article/view/2384>. Citado 2 vezes nas páginas 25 e 26.

ALVES, T. H. C. R. *Uma Arquitetura Baseada em Containers para Workflows de Bioinformática em Nuvens Federadas*. [s.n.], 2017. Disponível em: <https://goo.gl/U9yKuL>. Citado 3 vezes nas páginas 15, 22 e 24.

BRAGA, A. S.; SILVA, G. M.; BARROS, M. C. *Cloud Computing*. [s.n.], 2012. Disponível em: <http://www.ic.unicamp.br/~ducatte/mo401/1s2012/T2/G08-079713-079740-820650-t2.pdf>. Citado 2 vezes nas páginas 21 e 22.

BRAZ, C. de M. *Processamento de Imagens usando CUDA*. [s.n.], 2015. Disponível em: <https://www.ime.usp.br/~gold/cursos/2015/MAC5742/reports/ImageProcessingCUDA.pdf>. Citado na página 29.

BREDER, B. *Ordenação da Submissão de Kernels Concorrentes para Maximizar a Utilização dos Recursos da GPU*. [s.n.], 2016. Disponível em: <http://www2.ic.uff.br/PosGraduacao/Dissertacoes/740.pdf>. Citado 3 vezes nas páginas 27, 28 e 29.

CARISSIMI, A. *Virtualização: da teoria a soluções*. [s.n.], 2009. Disponível em: <http://www.jvasconcellos.com.br/unijorge/wp-content/uploads/2012/01/cap4-v2.pdf>. Citado na página 23.

CARVALHO, L. S. P.; ANJOS, M. Cézar Lopes dos. *Impacto dos padrões arquiteturais de Micro Serviço e Monolítico no desenvolvimento de softwares*. [s.n.], 2017. Disponível em: <https://goo.gl/6KxRu9>. Citado na página 15.

DAMASCENO, J. C. *UCLOUD: UMA ABORDAGEM PARA IMPLANTAÇÃO DE NUVEM PRIVADA PARA A ADMINISTRAÇÃO PÚBLICA FEDERAL*. [s.n.], 2015. Disponível em: <https://goo.gl/6ZnHej>. Citado na página 20.

DOMINGOS, D. P. *Utilização de GPU para Sistemas de Paralelismo Massivo*. [s.n.], 2012. Disponível em: <http://www.ic.unicamp.br/~cortes/mo601/trabalho_mo601/diego_domingos_gpu/artigo_v2.pdf>. Citado 2 vezes nas páginas 26 e 27.

DORNELAS, J. S.; SOUZA, K. R. R. d. *Cloud Computing: em busca da compreensão de seu uso em organizações públicas*. [s.n.], 2016. Disponível em: <http://revista.apsi.pt/index.php/capsi/article/view/484/438>. Citado na página 19.

DRAGHICI, R. *Docker vs Virtualization*. [s.n.], 2014. Disponível em: <https://monkeyvault.net/docker-vs-virtualization/>. Citado na página 23.

FERNANDEZ, G. P. *Orquestração de Contêineres na Nuvem: Um Modelo de Segurança*. [s.n.], 2018. Disponível em: <http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/1346>. Citado na página 19.

FERREIRA, J. C. dos S. *CLEM & OCEAN: Dois Compiladores OpenCL para as Arquiteturas Manycore METAL e ArachNoC*. [s.n.], 2016. Disponível em: <http://repositorio.ufpi.br/xmlui/bitstream/handle/123456789/415/dissertacaoJonatasFerreira2016.pdf?sequence=1>. Citado na página 29.

FIELDING, R. T. et al. *Hypertext Transfer Protocol – HTTP/1.1*. [S.l.], 1999. <http://www.rfc-editor.org/rfc/rfc2616.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc2616.txt>. Citado na página 15.

FILHO, A. A. G. M. *Infraestrutura de redes de computadores focada em SaaS*. [s.n.], 2012. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/images/4/4b/TCC_AlexandreAugusto.pdf>. Citado na página 22.

FILHO, A. C. de A. *Estudo comparativo entre Docker Swarm e Kubernetes para orquestração de contêineres em arquiteturas de software com microsserviço*. [s.n.], 2016. Disponível em: <http://www.cin.ufpe.br/~tg/2016-2/acaf.pdf>. Citado na página 24.

FORSEC. *Qual é a diferença entre IaaS, SaaS e PaaS?* [s.n.], 2016. Disponível em: <http://forsec.com.br/site/qual-e-a-diferenca-entre-iaas-saas-e-paas>. Citado na página 21.

HILLESHEIM, M. *Chatbot para monitoramento e controle de orquestrador de nuvem de contêineres*. [s.n.], 2018. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/images/7/7c/Proposta.pdf>. Citado 3 vezes nas páginas 23, 25 e 26.

JRADI, W. A. R. *Application of GPU Computing to Some Urban Traffic Problems*. [s.n.], 2016. Disponível em: <https://goo.gl/7tjYB5>. Citado 4 vezes nas páginas 26, 27, 28 e 29.

KANG, D. et al. *ConVGPU: GPU Management Middleware Container Based Virtualized Environment*. [s.n.], 2017. Disponível em: <http://dx.doi.org/10.1109/CLUSTER.2017.17>. Citado 3 vezes nas páginas 15, 27 e 29.

KUBERNETES. *Kubernetes Documentation*. [s.n.], 2018. Disponível em: <https://kubernetes.io/docs/home/>. Citado 3 vezes nas páginas 24, 25 e 26.

KöRBES, L. H.; WILDNER, M. C. S. *A UTILIZAÇÃO DE APLICATIVOS DA COMPUTAÇÃO EM NUVEM NO ENSINO SUPERIOR*. [s.n.], 2016. Disponível em: <http://www.univates.br/revistas/index.php/destaques/article/viewFile/1229/1093>. Citado na página 19.

LAUREANO, M. A. P.; MAZIERO, C. A. *Virtualização: Conceitos e Aplicações em Segurança*. [S.l.: s.n.], 2008. Citado 2 vezes nas páginas 22 e 23.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg, MD, 2011. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>. Citado 2 vezes nas páginas 19 e 20.

MOURELLE, L. *Arquiteturas de Alto Desempenho*. [s.n.], 2011. Disponível em: <http://www.eng.uerj.br/~ldmm/Arquiteturas_de_Alto_Desempenho/Paralelismo.pdf>. Citado na página 26.

NETTO, H. et al. *Replicação de Máquinas de Estado em containers no Kubernetes: uma Proposta de Integração*. [s.n.], 2016. Disponível em: <http://www.sbrc2016.ufba.br/downloads/SessoesTecnicas/152295.pdf>. Citado na página 24.

NVIDIA, T. A. of. *CUDA Education & Training*. [s.n.], 2015. Disponível em: <https://developer.nvidia.com/cuda-education-training>. Citado na página 29.

NVIDIA, T. A. of. *NVIDIA Container Runtime for Docker*. [s.n.], 2015. Disponível em: <https://github.com/NVIDIA/nvidia-docker>. Citado na página 31.

OLIVEIRA, D.; JUNIOR, T. A. V.; SANTOS, N. *Computação em Nuvem para Gerenciamento de Rede de Ensino*. [s.n.], 2011. Disponível em: <https://goo.gl/kDKmFo>. Citado na página 19.

OLIVEIRA, J. d. *VIRTUALIZAÇÃO E O USO DE SOFTWARE LIVRE*. [s.n.], 2013. Disponível em: <https://goo.gl/Sa4EJb>. Citado 2 vezes nas páginas 22 e 23.

PEREIRA, A. L. et al. *COMPUTAÇÃO EM NUVEM: A SEGURANÇA DA INFORMAÇÃO EM AMBIENTES NA NUVEM E EM REDES FÍSICAS*. [s.n.], 2016. Disponível em: <http://periodicos.ufes.br/BJPE/article/viewFile/EO02_2016/pdf>. Citado na página 19.

PETCOV, R. *Aplicativos em nuvem*. Editora Senac São Paulo, 2018. (Universitária). ISBN 9788539612291. Disponível em: <https://books.google.com.br/books?id=G4ZIDwAAQBAJ>. Citado na página 19.

PINTO, V. G. *Ambientes de Programação Paralela Híbrida*. [s.n.], 2011. Disponível em: <http://www.inf.ufrgs.br/~vgpinto/publicacoes/ti1ufrgs.pdf>. Citado na página 29.

PUC-RIO. *Unidades de Processamento Gráfico – GPUs*. [s.n.], 2018. Disponível em: <https://www.maxwell.vrac.puc-rio.br/24129/24129_6.PDF>. Citado na página 28.

REDHAT. *RedHat Documentation*. [s.n.], 2018. Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes>. Citado na página 24.

SANTOS, R. C. M. d. *Implantação de Infraestrutura como Serviço em uma Nuvem Computacional Privada*. [s.n.], 2016. Disponível em: <http://www.bdm.unb.br/bitstream/10483/17042/1/2016_RafaelCesarDosSantos_tcc.pdf>. Citado 2 vezes nas páginas 21 e 22.

SILVA, A. T. D. *OTIMIZAÇÃO DE PATHFINDING EM GPU*. [s.n.], 2013. Disponível em: <https://goo.gl/1XRM4e>. Citado na página 28.

SILVA, F. H. R. e. *Avaliação de Desempenho de Contêineres Docker para Aplicações do Supremo Tribunal Federal*. [s.n.], 2017. Disponível em: <https://goo.gl/FFmXWC>. Citado 2 vezes nas páginas 15 e 22.

SILVA, H. B. d. *UMA INVESTIGAÇÃO SOBRE O PROCESSO MIGRATÓRIO PARA A PLATAFORMA DE COMPUTAÇÃO EM NUVEM NO BRASIL*. [s.n.], 2016. Disponível em: <https://goo.gl/hvB8fB>. Citado 2 vezes nas páginas 20 e 22.

SILVA, J. C. Q. d. *USO DE NUVENS HÍBRIDAS NAS EMPRESAS BRASILEIRAS*. [s.n.], 2017. Disponível em: <https://goo.gl/98YuT9>. Citado na página 19.

SILVA, W. F. D. *UTILIZANDO VIRTUALIZAÇÃO BASEADA EM CONTAINERS PARA CRIAÇÃO DE LABORATÓRIOS PRÁTICOS DE DISCIPLINAS NA ÁREA DE TI*. [s.n.], 2017. Disponível em: <http://www.repositorio.ufc.br/bitstream/riufc/29549/1/2017_tcc_wfsilva.pdf>. Citado 2 vezes nas páginas 23 e 24.

SILVEIRA, M. F. G. d. *A VIRTUALIZAÇÃO E OS EFEITOS NA MÉTRICA DE PUE*. [s.n.], 2018. Disponível em: <https://www.riuni.unisul.br/bitstream/handle/12345/4810/AD6.pdf?sequence=1&isAllowed=y>. Citado 2 vezes nas páginas 22 e 23.

SOUZA, F. R. C. *Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios*. [S.l.: s.n.], 2010. Citado 2 vezes nas páginas 20 e 22.

TAURION, C. *Computação em Nuvem. Transformando o mundo da Tecnologia da Informação*. [S.l.: s.n.], 2009. ISBN 9788574524238. Citado na página 15.

THEISGES, M. L. *Mineração de bancos de dados distribuídos usando inteligência artificial para análise de log de nuvem privada de contêineres*. [S.l.: s.n.], 2018. Citado 3 vezes nas páginas 15, 25 e 26.

VERAS, M. *Cloud Computing. Nova Arquitetura TI*. Editora Brasport Rio de Janeiro, 2012. Disponível em: <https://books.google.com.br/books?id=G4ZIDwAAQBAJ>. Citado 2 vezes nas páginas 19 e 20.