Provisionamento Dinâmico de Clusters para o Ensino na Computação

PEDRO IVO KUHN[1]; GABRIEL MULLER FISCHER[2]; GERSON GERALDO H. CAVALHEIRO[3]; RAFAEL BURLAMAQUI AMARAL[4];

[1]Universidade Federal de Pelotas – pikuhn@inf.ufpel.edu.br
[2]Universidade Federal de Pelotas – gmfischer@inf.ufpel.edu.br
[3]Universidade Federal de Pelotas – gerson.cavalheiro@inf.ufpel.edu.br
[4]Universidade Federal de Pelotas – rafael.amaral@inf.ufpel.edu.br

1. INTRODUÇÃO

Este trabalho se insere na área do conhecimento de Tecnologia na Educação, com foco no provimento de infraestrutura de hardware para aprimorar os processos de ensino-aprendizagem em tópicos avançados da Computação. O assunto central é a disponibilização de clusters computacionais sob demanda, projetados para o ensino de diversas práticas computacionais em um ambiente acadêmico, tais como o uso de ferramentas para programação paralela e distribuída (MPI). A problematização que norteia este estudo parte da dependência da infraestrutura provida pela instituição para que a aprendizagem não se limite apenas em simulações. Contudo, a alocação de recursos dedicados por disciplina é uma solução de alto custo e difícil gerenciamento, por esse motivo, se faz necessário uma solução flexível e compartilhada, capaz de alocar essa infraestrutura de forma eficiente e sob demanda. Portanto, se busca oferecer uma plataforma virtualizada para que os estudantes tenham acesso a uma plataforma de experimentação, sem necessitar abordar questões técnicas relacionadas a configuração e manutenção de um cluster. O objetivo geral deste trabalho é projetar e desenvolver uma plataforma web que provisiona, sob demanda, ambientes virtuais com múltiplas máquinas, oferecendo a cada usuário, um cluster privado, seguro e acessível, capaz de atender as necessidades de um ambiente acadêmico com a finalidade de ser uma ferramenta prática em uma rede local, como a de uma sala de aula.

2. METODOLOGIA

A fundamentação teórica que embasa esta pesquisa combina estudos sobre a aplicação de tecnologias de conteinerização e orquestração no contexto educacional. A tecnologia Docker (DOCKER, 2025) emergiu como um padrão para a criação de ambientes isolados e portáteis. Autores como Merkel (2014) destacam as vantagens dos contêineres, como a leveza e a rapidez de inicialização. Contudo, para gerenciar um conjunto de contêineres interconectados como um sistema coeso, uma camada de orquestração é necessária. Para este fim, adotamos o Kubernetes (KUBERNETES, 2025), plataforma que automatiza o escalonamento e a gestão de redes de aplicações conteinerizadas, tornando viável a criação de clusters dinâmicos.

Este trabalho está baseado em uma arquitetura de microsserviços orientado a eventos, combinada com princípios de Infraestrutura como Código (IaC) para orquestrar ambientes virtuais. A abordagem foi escolhida por sua capacidade de gerenciar de forma automatizada e replicável todo o ciclo de vida desses ambientes, sendo capaz de alocar os recursos necessários, configurar o ambiente

e, ao final, desalocá-lo de maneira eficiente. A solução está estruturada em três pilares principais: a arquitetura da solução, o fluxo de interação do usuário e o modelo de segurança e isolamento.

2.1. Arquitetura da Solução

A solução foi projetada a partir de uma arquitetura de três camadas interconectadas, para operar em ambientes de rede local, como os laboratórios de informática da instituição. Sendo a primeira camada a infraestrutura, um cluster Kubernetes flexível, capaz de operar tanto em múltiplos nós (máquinas) físicas quanto em um único nó para fins de validação e desenvolvimento. A segunda camada é o backend (servidor), o núcleo da aplicação desenvolvido em Node.js (NODE.JS, 2025) e com o framework Express.js (EXPRESS.JS, 2025), que atua como o orquestrador central: gerenciando um stream de dados entre cliente e servidor pela biblioteca Socket.IO (SOCKET.IO, 2025); traduzir as solicitações em comandos para a API @kubernetes/client-node (KUBERNETES, 2025) e gerar os recursos isolados para cada sessão de usuário. A terceira camada é o frontend (cliente), uma página web construída com HTML, CSS e JavaScript, cuja responsabilidade é capturar a solicitação do usuário sobre o tamanho do cluster e renderizar o terminal interativo utilizando a biblioteca Xterm.js (XTERM.JS, 2025). Na Figura 1 é representada todos os processos que cada camada é responsável, a partir do momento em que o usuário se conecta a partir da rede da UFPel.

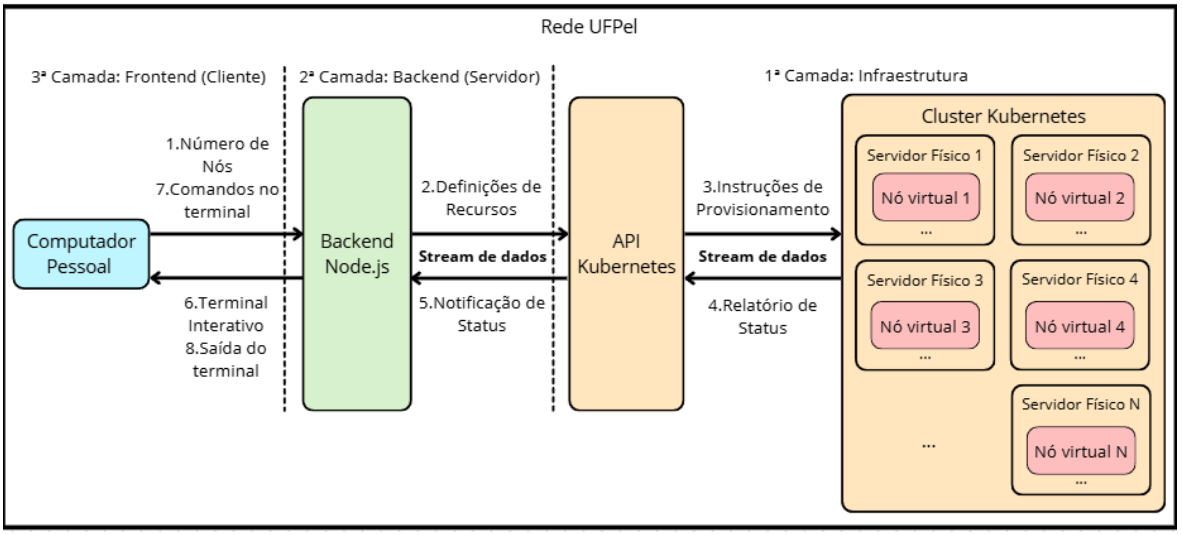

Figura 1 - Funcionamento da Plataforma

2.2. Fluxo de Interação e Usabilidade

O fluxo de interação do usuário foi projetado para que o acesso à complexa infraestrutura de cluster seja totalmente abstraído. Toda a interação ocorre por meio da interface web do frontend (Camada 3), que consiste em duas telas principais, como mostra a Figura 2. A primeira tela serve para a configuração do ambiente, onde se define o número de nós virtuais. Após a submissão, o backend (Camada 2) provisiona o ambiente em tempo real e a interface transita para a segunda tela: um painel de controle com a lista de máquinas virtuais e o terminal de acesso ao nó mestre, pronto para uso.

Figura 2 - Interface do Usuário

2.3. Modelo de Segurança

A segurança e o isolamento entre as sessões dos usuários são garantidos por dois mecanismos principais. Primeiramente, para cada nova sessão, um identificador único é gerado e utilizado para nomear todos os recursos Kubernetes correspondentes, prevenindo qualquer colisão ou acesso cruzado entre os ambientes dos usuários. Em segundo lugar, e mais importante, é implementado um isolamento criptográfico: um par de chaves SSH é gerado dinamicamente para cada pool computacional utilizando o node-forge (NODE-FORGE, 2025). Essas chaves são distribuídas de forma segura apenas para os nós daquela sessão, garantindo criptograficamente que apenas os contêineres pertencentes ao mesmo pool possam se comunicar entre si.

3. RESULTADOS E DISCUSSÃO

Para validar a funcionalidade da orquestração e a capacidade da plataforma em gerenciar processos paralelos, foi executado um algoritmo de multiplicação de matrizes com MPI (2048x2048), esta escolha se justifica por ser um problema clássico, onde é computacionalmente intensivo, facilmente paralelizável e de alta comunicação entre os processos. Os testes foram realizados em um ambiente de desenvolvimento de nó único (físico), utilizando a distribuição kubernetes inclusa no Docker Desktop, e possuindo as seguintes configurações: 6 cores, 12 threads e 16GB de memória RAM. Embora este ambiente não demonstre o ganho de performance de um cluster com múltiplos nós físicos. A execução foi dividida em dois cenários. No primeiro, utilizando um único nó virtual, o tempo de execução foi de 196 segundos. No segundo, a carga de trabalho foi distribuída entre quatro nós virtuais executando em paralelo, resultando em um tempo de 71 segundos.

A redução no tempo de execução, mesmo em um único nó físico, resultou em um ganho de performance de aproximadamente 2,76 vezes, assim comprovando que a solução orquestrou com sucesso os processos paralelos através da rede do

Kubernetes e que a comunicação MPI está funcionando como esperado. Este resultado confirma a viabilidade da arquitetura como uma ferramenta funcional para o ensino acadêmico, cuja performance pode ser escalada ao ser implantada no cluster físico de múltiplos nós do laboratório.

4. CONCLUSÕES

Este trabalho apresentou o projeto e desenvolvimento de uma plataforma web que utiliza a virtualização para criar ambientes computacionais sob demanda. O objetivo de validar a viabilidade técnica de uma arquitetura capaz de orquestrar ambientes virtuais de múltiplos nós de forma isolada e sob demanda foi alcançado, resultando em um protótipo que comprova a eficácia deste modelo.

Conclui-se que o "motor" central da solução, responsável pela orquestração, gerenciamento de rede, configuração de segurança e disponibilização de acesso aos ambientes virtuais através de um terminal web, está completo e funcional. A validação em um ambiente de teste com um único nó físico, confirmou que todos os componentes lógicos e a comunicação MPI operam corretamente.

5. REFERÊNCIAS BIBLIOGRÁFICAS

MERKEL, Dirk et al. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, v. 239, n. 2, p. 2, 2014. Acesso em: 13 ago. 2025.

KUBERNETES. *@kubernetes/client-node – Kubernetes Official Node.js Client Library*. GitHub, 2025. Disponível em: https://github.com/kubernetes-client/javascript. Acesso em: 13 ago. 2025.

XTERM.JS. *Xterm.js – A Terminal for the Web*. 2025. Disponível em: https://xtermjs.org/. Acesso em: 14 ago. 2025.

NODE.JS. *Node.js – JavaScript Runtime Environment*. 2025. Disponível em: https://nodejs.org/pt. Acesso em: 23 ago. 2025.

EXPRESS.JS. *Express.js – Node.js Web Application Framework*. 2025. Disponível em: https://expressjs.com/. Acesso em: 23 ago. 2025.

SOCKET.IO. *Socket.io – Communication Between a Client And a Server*. 2025. Disponível em: https://socket.io/. Acesso em: 23 ago. 2025.

DOCKER. Docker – Empowering App Development for Developers. 2025. Disponível em: https://www.docker.com/. Acesso em: 13 ago. 2025.

NODE-FORGE. *node-forge – JavaScript Implementation of TLS, PKI, AES, SHA1, RSA, etc.* NPM, 2025. Disponível em: https://www.npmjs.com/package/node-forge. Acesso em: 23 ago. 2025.