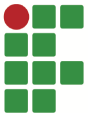

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS CAMPINA GRANDE
CURSO SUPERIOR BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

IGOR KÁDSON DE SOUZA OLIVEIRA

Uma Arquitetura Multiagente para Orquestração Automática em Infraestruturas OpenStack

CAMPINA GRANDE
2025

IGOR KÁDSON DE SOUZA OLIVEIRA

Uma Arquitetura Multiagente para Orquestração Automática em Infraestruturas OpenStack

Trabalho de Conclusão de Curso do CURSO SUPERIOR BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO submetido ao da INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA CAMPUS CAMPINA GRANDE para a obtenção do título de BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO.
Orientador: Prof.Dr. Igor Barbosa da Costa

CAMPINA GRANDE
2025

Catalogação na fonte:
Ficha catalográfica elaborada por Gustavo César Nogueira da Costa – CRB 15/479

O48a Oliveira, Igor Kádson de Souza.
Uma arquitetura multiagente para orquestração automática em infraestruturas OpenStack / Igor Kádson de Souza Oliveira. - Campina Grande, 2025.
24 f.: il.

Trabalho de Conclusão de Curso (Graduação em Bacharelado em Engenharia de Computação) - Instituto Federal da Paraíba, 2025.
Orientador: Prof. Dr. Igor Barbosa da Costa

1. Engenharia de computação. 2. Computação em nuvem. 3. OpenStack. 4. Sistemas multiagentes. 5. Inteligência artificial. I. Costa, Igor Barbosa da. II Título.

CDU 004.7

IGOR KÁDSON DE SOUZA OLIVEIRA

Uma Arquitetura Multiagente para Orquestração Automática em Infraestruturas OpenStack

Artigo apresentado à Coordenação do Curso de Bacharelado em Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Trabalho aprovado.

Banca Examinadora:

Prof.Dr. Igor Barbosa da Costa
Orientador

Prof. Dra. Daniella Dias Cavalcante da Silva
Avaliador

Prof. Dr. Petrônio Carlos Bezerra
Avaliador

CAMPINA GRANDE, , 12 de Dezembro de 2025.

Uma Arquitetura Multiagente para Orquestração Automática em Infraestruturas OpenStack

Igor Oliveira, Igor Costa

[1]Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB)
CEP – 58.432-300 – Campina Grande – PB – Brasil

igor.kadson@academico.ifpb.edu.br, igor.costa@ifpb.edu.br

Resumo. *Este trabalho investiga a viabilidade de utilizar um sistema multiagente para orquestrar recursos OpenStack de forma autônoma, por meio do desenvolvimento do* `ctrlStack`*, uma prova de conceito integrada ao plano de controle* `OpenStackMCPServer`*. A arquitetura proposta combina modelos de linguagem de grande porte com princípios de Sistemas Multiagentes (MAS), permitindo que agentes especializados interpretem intenções em linguagem natural, validem o contexto da infraestrutura e executem operações de provisionamento. O* `OpenStackMCPServer` *expõe funcionalidades do OpenStack por meio do protocolo Model Context Protocol (MCP), encapsulando operações complexas do SDK e fornecendo uma interface padronizada para os agentes. Resultados obtidos em um ambiente real de nuvem indicam que o sistema é capaz de realizar, de forma autônoma, fluxos completos de autenticação, descoberta de recursos e criação de instâncias, reduzindo a complexidade operacional quando comparado a métodos tradicionais de provisionamento. As limitações observadas também revelam desafios importantes relacionados à confiabilidade da tomada de decisão semântica, apontando direções para aprimoramentos futuros.*
Palavras-chave*: computação em nuvem; OpenStack; sistemas multiagentes; orquestração; modelos de linguagem.*

Abstract. *This work investigates the feasibility of using a multi-agent system to autonomously orchestrate OpenStack resources by developing* `ctrlStack`*, a proof of concept integrated with the* `OpenStackMCPServer` *control plane. The proposed architecture combines large language models with principles of Multi-Agent Systems (MAS), enabling specialized agents to interpret natural-language intentions, validate infrastructure context, and execute provisioning tasks. The* `OpenStackMCPServer` *exposes OpenStack operations through the Model Context Protocol (MCP), encapsulating SDK complexity and offering a standardized interface for agent interaction. Experiments conducted in a real OpenStack environment demonstrate that the system can autonomously perform complete workflows—including authentication, resource discovery, and virtual machine creation—while reducing operational complexity when compared to traditional provisioning methods. The observed limitations, particularly regarding semantic decision-making reliability, highlight challenges inherent to LLM-driven orchestration and point to opportunities for future improvements. Overall, the results support the potential of combining MAS, LLMs, and MCP as a foundation for intelligent cloud orchestration.*

1. Introdução

A computação em nuvem consolidou-se como um dos pilares da infraestrutura digital contemporânea, ao permitir o provisionamento dinâmico e escalável de recursos computacionais sob demanda (Mell and Grance, 2011). No modelo *Infrastructure as a Service* (IaaS), provedores oferecem infraestrutura virtualizada como serviço, viabilizando ambientes flexíveis e altamente configuráveis, ao mesmo tempo em que reduzem a necessidade de investimentos em hardware físico.

Entre as plataformas de código aberto para implementação de IaaS, o OpenStack destaca-se pela arquitetura modular e pelo ecossistema consolidado de serviços de computação, redes, armazenamento e autenticação (Denton, 2019). Apesar dessa robustez, a operação de ambientes OpenStack permanece desafiadora, em razão da complexidade de configuração, da integração entre múltiplos componentes e da dependência de equipes altamente especializadas para o uso correto de APIs, *dashboards* e ferramentas de automação.

Nesse cenário, a orquestração de recursos em nuvem assume papel central ao automatizar processos de implantação, escalonamento e monitoramento de cargas de trabalho. Ferramentas como Terraform, Heat e Ansible permitem descrever ambientes de forma declarativa, reduzindo erros manuais e padronizando a infraestrutura. No entanto, a utilização dessas soluções ainda exige domínio de linguagens específicas, compreensão detalhada de topologias de serviços e interação direta com scripts e arquivos de configuração complexos, o que restringe sua acessibilidade a perfis altamente técnicos e aumenta a barreira de adoção em contextos corporativos e acadêmicos.

Diante disso, emerge o seguinte problema de pesquisa: **como permitir que usuários descrevam, em linguagem natural, intenções de alto nível sobre a infraestrutura, e que essas intenções sejam convertidas, de forma autônoma e confiável, em operações concretas de orquestração em ambientes OpenStack?** Apesar da existência de ferramentas maduras de automação declarativa, ainda faltam soluções que integrem, em uma única arquitetura, compreensão semântica, tomada de decisão distribuída e execução automatizada sobre plataformas de nuvem.

A integração entre Inteligência Artificial (IA) e Sistemas Multiagentes (MAS) surge como alternativa promissora para enfrentar esse desafio. A IA conversacional tem transformado a interação entre humanos e sistemas computacionais ao possibilitar o uso de linguagem natural em substituição a interfaces estritamente técnicas (Jurafsky and Martin, 2023), democratizando o acesso a tecnologias complexas. Em paralelo, o paradigma de sistemas multiagentes oferece um modelo distribuído no qual múltiplos agentes de software autônomos percebem o ambiente, tomam decisões e coordenam ações por meio de interações sociais estruturadas (Wooldridge, 2009; Jennings, 2001). Essa combinação é especialmente adequada a problemas que exigem descentralização, adaptabilidade e coordenação entre múltiplas funções em infraestruturas de larga escala.

Neste contexto, este trabalho propõe uma **arquitetura multiagente** implementada no **sistema** `ctrlStack`, voltado à orquestração autônoma de recursos

em ambientes OpenStack. O sistema é estruturado sobre um plano de controle denominado `OpenStackMCPServer`, que adota o padrão aberto *Model Context Protocol* (MCP) como mecanismo de integração entre agentes e serviços externos. Nesse arranjo, o MCP é aplicado como camada de controle e coordenação entre o framework `CrewAI` e a API do OpenStack, permitindo que agentes autônomos interpretem intenções em linguagem natural, distribuam tarefas e executem operações de maneira contextualizada e reativa.

O objetivo geral deste trabalho é propor e validar uma arquitetura multiagente capaz de realizar orquestração autônoma em ambientes OpenStack. De forma específica, pretende-se:

- desenvolver o plano de controle `OpenStackMCPServer` baseado no protocolo MCP, expondo ferramentas de gerenciamento de recursos OpenStack a agentes inteligentes;
- integrar o framework `CrewAI` para coordenação autônoma de agentes responsáveis pela interpretação de intenções, validação de contexto e execução de operações; e
- avaliar a viabilidade da abordagem e caracterizar, de forma qualitativa, os ganhos de automação em relação a métodos tradicionais de provisionamento declarativo.

Como principais contribuições, este trabalho apresenta:

- uma arquitetura multiagente baseada em MCP para orquestração autônoma em ambientes OpenStack;
- a implementação do protótipo `ctrlStack`, integrando o `CrewAI` a um plano de controle distribuído (`OpenStackMCPServer`) responsável por encapsular a complexidade da API do OpenStack; e
- a validação preliminar da abordagem em um cenário realista de provisionamento de máquinas virtuais, evidenciando aumento do grau de automação e redução da complexidade operacional percebida em relação a abordagens tradicionais.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica; a Seção 3 revisa os principais trabalhos relacionados; a Seção 4 descreve a metodologia e a arquitetura do sistema proposto; a Seção 5 apresenta os resultados; a Seção 6 discute criticamente esses resultados; e, por fim, a Seção 7 apresenta as conclusões e direções para trabalhos futuros.

2. Fundamentação Teórica

Esta seção apresenta os conceitos que sustentam o desenvolvimento deste trabalho. Inicialmente, são discutidos os fundamentos da computação em nuvem e dos mecanismos de orquestração de recursos. Em seguida, abordam-se os princípios da Inteligência Artificial e dos Sistemas Multiagentes (MAS). Por fim, é explorado o padrão *Model Context Protocol* (MCP), que fornece a base conceitual e técnica para o plano de controle adotado na arquitetura proposta.

2.1. Computação em Nuvem e Orquestração

A computação em nuvem pode ser definida como um modelo para acesso ubíquo, conveniente e sob demanda a um conjunto de recursos computacionais configuráveis

(por exemplo, redes, servidores, armazenamento e aplicações), que podem ser rapidamente provisionados e liberados com mínima intervenção humana ou interação com o provedor de serviços (Mell and Grance, 2011). Entre os modelos de serviço usualmente adotados, destacam-se *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS), sendo o primeiro o foco deste trabalho.

No modelo IaaS, o provedor oferece recursos de infraestrutura virtualizada como serviço, permitindo que usuários criem, configurem e gerenciem máquinas virtuais, redes e volumes de armazenamento de forma flexível. Essa abordagem reduz custos de capital, aumenta a elasticidade do ambiente e favorece a automação de tarefas de operação e manutenção.

O OpenStack surge como uma das principais plataformas de código aberto para implementação de IaaS. Sua arquitetura é composta por serviços especializados, como Nova (computação), Neutron (redes), Cinder e Swift (armazenamento em bloco e em objetos) e Keystone (autenticação e autorização), entre outros (Denton, 2019). A integração desses serviços fornece uma base robusta para construção de nuvens privadas e públicas, porém introduz desafios significativos de configuração, operação e integração entre componentes heterogêneos (Villari et al., 2016).

Nesse contexto, a orquestração de recursos em nuvem desempenha papel estratégico. Diferentemente de mecanismos simples de automação, a orquestração refere-se à coordenação de múltiplas operações interdependentes, envolvendo implantação, escalonamento, monitoramento e recuperação de serviços distribuídos. Ferramentas consolidadas como OpenStack Heat, Terraform e Ansible permitem descrever estados alvo da infraestrutura por meio de modelos declarativos, automatizando a criação e a configuração de recursos (Villari et al., 2016).

Com o amadurecimento de ambientes de nuvem e *edge*, surgiram abordagens de orquestração mais inteligentes, capazes de reagir a eventos do ambiente e ajustar dinamicamente políticas de alocação (Dang et al., 2025). Conceitos como *intent-based orchestration* buscam traduzir objetivos de alto nível em políticas e fluxos de execução, reduzindo a necessidade de intervenção manual. Ainda assim, essas soluções permanecem fortemente acopladas a linguagens formais e descrições estruturadas.

2.2. Inteligência Artificial e Sistemas Multiagentes

A Inteligência Artificial (IA) tem avançado significativamente em tarefas de compreensão e geração de linguagem natural (Jurafsky and Martin, 2023). Modelos de linguagem de grande porte (*Large Language Models* — LLMs) expandiram esse potencial, viabilizando aplicações que vão desde assistentes virtuais até agentes capazes de raciocinar sobre sequências de ações em ambientes complexos.

Paralelamente, a área de Sistemas Multiagentes (MAS) fornece um arcabouço conceitual para o desenvolvimento de sistemas compostos por múltiplos agentes de software autônomos (Wooldridge, 2009). Quando organizados em sistemas cooperativos, esses agentes podem coordenar-se por meio de mecanismos de comunicação e negociação (Jennings, 2001).

Diversos modelos de agentes são descritos na literatura, incluindo agentes reativos, deliberativos e arquiteturas híbridas. Em particular, o modelo de agentes

situados enfatiza a interação contínua com o ambiente, na qual o comportamento emerge da relação entre percepções, objetivos e ações.

Estudos recentes apontam o potencial da integração entre IA e MAS em data centers e ambientes de nuvem. (Li et al., 2024) apresentam um levantamento de soluções baseadas em agentes para gerenciamento de recursos. O framework AgentFlow propõe uma arquitetura multiagente híbrida (nuvem + *edge*) capaz de realizar eleição dinâmica de serviços e ajustar políticas de roteamento (Singh et al., 2025). De forma complementar, *frameworks* de agentes colaborativos, como o `CrewAI` (Inc., 2025), abstraem a criação de equipes de agentes especializados, coordenados em torno de um objetivo comum.

2.3. Model Context Protocol (MCP)

Embora LLMs e MAS ofereçam uma base poderosa para raciocínio e coordenação, esses sistemas precisam de mecanismos padronizados para interagir com serviços externos, expor ferramentas e manter contexto entre múltiplas chamadas. O *Model Context Protocol* (MCP) surge como um padrão aberto de integração entre agentes inteligentes e sistemas externos.

De forma geral, o MCP define um modelo de comunicação em que *clients* interagem com *servers* que expõem recursos, ferramentas e fluxos de dados de forma padronizada. Cada servidor MCP é responsável por publicar um conjunto de ferramentas (*tools*) e esquemas de dados.

No contexto deste trabalho, o `OpenStackMCPServer` implementa o padrão MCP como camada intermediária entre o framework `CrewAI` e a nuvem OpenStack. Esse servidor expõe, como ferramentas MCP, operações típicas de gerenciamento, tais como autenticação, listagem de *flavors*, redes e imagens, e criação e remoção de máquinas virtuais. Cada ferramenta é descrita de forma estruturada, permitindo que os agentes entendam quais parâmetros são necessários e quais respostas podem ser esperadas (esquemas completos nos Apêndices A–C).

3. Trabalhos Relacionados

Ferramentas como OpenStack Heat, Terraform e Ansible permitem descrever ambientes de forma declarativa (Villari et al., 2016). Contudo, continuam exigindo conhecimento técnico detalhado.

Com o amadurecimento de ambientes de nuvem e *edge*, surgiram propostas de *intent-based orchestration*, como CAMINO (Antonakoglou et al., 2025) e OSM-VIM (Villari et al., 2016). Ainda assim, essas abordagens permanecem acopladas a linguagens formais.

Pesquisas em MAS aplicados a nuvem têm apresentado resultados promissores (Li et al., 2024; Singh et al., 2025). Mais recentemente, a combinação entre LLMs e *frameworks* multiagentes (e.g., `CrewAI`) viabilizou arquiteturas capazes de decompor objetivos e acionar ferramentas externas (Inc., 2025). Este trabalho diferencia-se ao propor o `ctrlStack`, integrando `CrewAI` e `OpenStackMCPServer` via MCP para orquestração OpenStack.

Tabela 1. Comparação entre abordagens de orquestração em nuvem

Solução	Linguagem natural	Coordenação multiagente	Orquestração OpenStack	Tipo de abordagem
OpenStack Heat	Não	Não	Sim (nativo)	Modelo declarativo; templates YAML
Terraform	Não	Não	Sim (via provider)	Infraestrutura como código (IaC); declarativo
Ansible	Não	Não	Sim (via módulos)	Automação procedural baseada em *playbooks*
CloudSim-Agents	Não	Sim (simulação)	Não	Simulador de nuvem com agentes autônomos
AgentFlow	Não	Sim	Não	Coordenação distribuída; eleição dinâmica de serviços
ctrlStack	Sim (via LLM + MCP)	Sim	Sim (via MCP + SDK)	Orquestração autônoma baseada em IA conversacional e MAS

4. Metodologia

Esta seção descreve a metodologia adotada. Inicialmente, apresenta-se uma visão geral da arquitetura e componentes. Em seguida, detalha-se o fluxo operacional (para facilitar a leitura junto à Figura 1) e, por fim, discutem-se as tecnologias e ferramentas empregadas.

4.1. Visão Geral da Arquitetura do Sistema

A metodologia adotada fundamenta-se na concepção e implementação de um sistema multiagente capaz de interagir com uma infraestrutura de nuvem OpenStack por meio de comandos expressos em linguagem natural. O objetivo central é demonstrar a viabilidade de automatizar tarefas complexas de gerenciamento em nuvem utilizando uma arquitetura distribuída baseada em agentes autônomos.

A arquitetura proposta é composta por três agentes principais: `UserInterface`, `Validator` e `Deployment`. Esses agentes foram implementados utilizando o framework `CrewAI` (Inc., 2025). O sistema é sustentado por um servidor intermediário denominado `OpenStackMCPServer`, implementado com o framework `FastMCP` (et al., 2024) e desenvolvido em Python, que atua como camada de abstração entre os agentes e a nuvem OpenStack. Essa camada realiza chamadas à API

do OpenStack por meio do OpenStack SDK (Foundation, 2023), encapsulando a complexidade das operações em uma interface exposta como *tools* MCP.

A Figura 1 apresenta o fluxo de interação entre usuário, agentes e infraestrutura, com o `OpenStackMCPServer` mediando a execução.

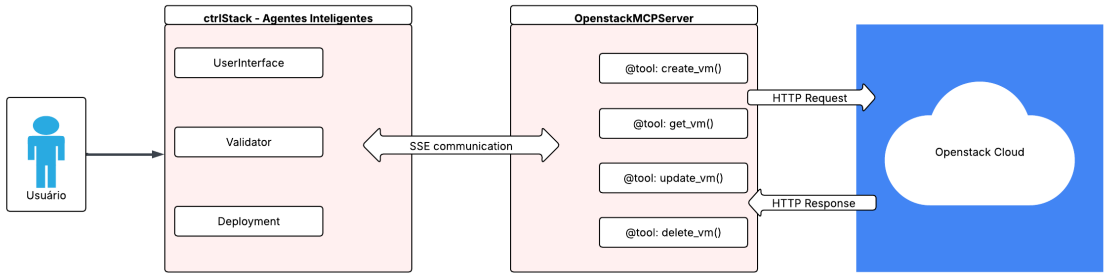

Figura 1. Fluxo de interação entre usuário, agentes e infraestrutura OpenStack.

Detalhamento dos módulos (referência à Figura 1).

- **`UserInterface`**: recebe a intenção em linguagem natural, consolida o objetivo (e.g., "criar VM com Ubuntu") e aciona o fluxo de *tasks*.
- **`Validator`**: autentica no OpenStack, descobre recursos (imagens, redes, *flavors*) e valida pré-condições para provisionamento.
- **`Deployment`**: seleciona parâmetros finais (com base em políticas) e executa operações de criação/remoção de instâncias via MCP.
- **`OpenStackMCPServer`**: encapsula as operações do SDK e expõe ferramentas MCP. No escopo deste trabalho, as ferramentas principais são:
  - `login`: cria sessão autenticada e retorna `session_id`;
  - `list_flavors`, `list_images`, `list_networks`: retornam recursos disponíveis para a sessão atual;
  - `create_server`: provisiona instância (com `SERVER_NAME`, `IMAGE`, `FLAVOR`, `NETWORK_NAME`, `KEYPAIR_NAME`);
  - `delete_server`: remove instância (operação sensível, discutida na Seção 6).

O que significa "retorno estruturado". As ferramentas MCP retornam respostas em formatos previsíveis (e.g., JSON), facilitando o consumo pelos agentes e reduzindo ambiguidades. Por exemplo, uma resposta típica de `login` é:

```
{"session_id":"01d2818e-2cff-4b69-b0b8-c0e67cc4e447"}
```

4.2. Estratégia de Implementação e Fluxo Operacional

O processo de implementação seguiu uma estratégia modular e iterativa. O fluxo de execução do sistema é definido a partir de arquivos YAML, que descrevem as *tasks*. O `CrewAI` interpreta essas descrições e executa cada etapa conforme a ordem definida. Cada *task* é associada a um agente responsável e a uma ferramenta registrada no `OpenStackMCPServer`.

De forma simplificada, o fluxo completo ocorre conforme as etapas a seguir:

1. **Inicialização do fluxo**: o comando `crewai run` inicia o fluxo de *tasks* descrito em `tasks.yaml`.
2. **Autenticação na nuvem**: o agente `Validator` executa `login_to_cloud`, invocando a ferramenta MCP `login`. O `OpenStackMCPServer` armazena o `session_id` e o reutiliza nas operações subsequentes.
3. **Descoberta e validação de recursos**: o agente `Validator` executa `list_flavors`, `list_networks` e `list_images`.
4. **Formatação dos resultados**: o `Validator` pós-processa as respostas e as traduz em formato legível e estável para a LLM.
5. **Criação da instância**: o agente `Deployment` executa `create_server`. Nesta prova de conceito, adota-se a política: menor *flavor* disponível, imagem preferencial *Ubuntu 22.04* e rede `provider`. O nome segue o padrão `crewai-2025-id`.

4.3. Tecnologias e Ferramentas Utilizadas

A implementação do sistema foi realizada em Python. O ambiente de desenvolvimento consistiu em um computador Dell XPS 13, com Intel Core i7 (10^{a} geração, 1,30 GHz), 16 GB RAM e 1 TB SSD, executando Fedora 41.

O framework `CrewAI` foi adotado para criação e coordenação dos agentes. Como modelo de linguagem utilizado pelo `CrewAI`, adotou-se o `gpt-4o-mini`, visando equilíbrio entre desempenho e capacidade semântica.

O `FastMCP` foi empregado na construção do `OpenStackMCPServer`, responsável pela comunicação com o OpenStack. O componente implementa um canal reativo via *Server-Sent Events* (SSE), mantendo persistência de contexto quando aplicável.

A comunicação com a nuvem utiliza o OpenStack SDK oficial, garantindo compatibilidade com distribuições OpenStack e abstrações para manipulação de instâncias, redes, volumes, *flavors* e imagens.

4.4. Critérios de Avaliação

A avaliação empírica da solução (Seção 5) considera:

- **Autonomia de orquestração**: execução do fluxo completo sem intervenção humana após credenciais.
- **Robustez do plano de controle**: manutenção de sessões, tratamento de erros e adaptação de consultas.
- **Confiabilidade em operações sensíveis**: observação do comportamento em operações destrutivas (e.g., deleção).

Detalhes técnicos complementares constam nos Apêndices A–C.

5. Resultados

Esta seção apresenta a análise empírica do funcionamento do `ctrlStack` em um ambiente real de nuvem OpenStack, com base nos registros de execução gerados pelo `CrewAI` e pelo `OpenStackMCPServer`. O objetivo é avaliar, de forma sistemática, (i) a capacidade dos agentes em interpretar intenções e coordenar ações de orquestração, (ii) a robustez da integração mediada pelo plano de controle MCP e (iii) o grau de autonomia alcançado na realização de operações típicas de provisionamento. Todos os experimentos foram conduzidos na nuvem pública do Laboratório de Sistemas Distribuídos (LSD-UFCG), utilizando credenciais e recursos de um ambiente de produção.

Os resultados discutidos a seguir derivam de uma execução completa do fluxo de *tasks* definido pelo sistema, englobando autenticação, descoberta de recursos, provisionamento de máquina virtual e tentativa de remoção. A análise fundamenta-se exclusivamente nos *logs* reais gerados pelos componentes, o que permite caracterizar com precisão o comportamento emergente da arquitetura multiagente em relação aos critérios definidos na Seção 4.

5.1. Configuração Experimental

O ambiente utilizado consiste em um cluster OpenStack com suporte a múltiplas redes, *flavors* e imagens de produção. A execução foi conduzida em sessão autenticada por meio de credenciais do tipo *application credential*, fornecidas ao agente em linguagem natural. Para fins de registro, o conjunto de parâmetros iniciais assumiu a forma:

```
URL do OpenStack: cloud.lsd.ufcg.edu.br
User: b2976d2af6e94b53af20f1aea7a20299
Senha: ********
Domain: lsd
```

Uma vez recebidos esses parâmetros, o sistema iniciou o fluxo de *tasks* definido no arquivo `tasks.yaml`, sem qualquer intervenção humana subsequente. Os agentes envolvidos foram: *UserInterface*, responsável por captar a intenção e iniciar o fluxo; *Validator*, encarregado da autenticação e coleta de recursos; e *Deployment*, responsável pelas operações de criação e deleção de instâncias.

5.2. Autenticação e Estabelecimento de Sessão

A primeira etapa do fluxo consistiu na autenticação no ambiente OpenStack. O agente *Validator* invocou a ferramenta MCP `login`, produzindo como resposta um identificador de sessão único:

```
{"session_id": "01d2818e-2cff-4b69-b0b8-c0e67cc4e447"}
```

O *log* do `OpenStackMCPServer` confirma a autenticação bem-sucedida via Keystone:

```
Login bem-sucedido.
Sessão criada: 01d2818e-2cff-4b69-b0b8-c0e67cc4e447
```

Esse resultado demonstra que o servidor MCP é capaz de estabelecer e manter contexto entre múltiplas requisições subsequentes, funcionando como plano de controle persistente ao longo de todo o *workflow* multiagente.

5.3. Descoberta Autônoma de Recursos

Após a autenticação, o agente *Validator* realizou a coleta de informações essenciais para o processo de provisionamento, invocando as ferramentas `list_flavors`, `list_images` e `list_networks`.

A listagem de *flavors* retornou, via `OpenStackMCPServer`, um conjunto de recursos reais, entre os quais:

```
general.small (1 vCPU, 2 GB RAM)
general.medium
general.large
general.xlarge
```

Para imagens, a consulta ao endpoint correspondente retornou mais de quinze imagens disponíveis, incluindo *snapshots* de produção e distribuições Linux, destacando-se:

```
ubuntu-22.04
ubuntu-24.04
inventario-prod
site-lsd
test
```

A etapa de descoberta de redes foi particularmente reveladora: a rede `provider` não foi localizada na primeira tentativa (`404 NetworkNotFound`), mas o `OpenStackMCPServer` realizou, de forma transparente, uma consulta alternativa utilizando filtro por nome, resultando na identificação correta do recurso:

```
GET /networks/provider -> 404
GET /networks?name=provider -> sucesso
```

Esse comportamento evidencia a resiliência operacional do plano de controle e sua capacidade de adaptar consultas para recuperar informações relevantes, mesmo diante de respostas inesperadas da API.

5.4. Provisionamento Autônomo de Instância

Com as informações coletadas, o agente *Deployment* realizou a etapa de decisão, definindo automaticamente *flavor*, imagem e rede com base em regras declarativas configuradas no arquivo de *tasks*. O sistema selecionou:

- o *flavor* mais econômico disponível: `general.small`;
- a imagem preferencial: `ubuntu-22.04`;
- a rede pré-configurada: `provider`;
- o nome da instância conforme convenção definida: `crewai-2025-1`.

A chamada MCP correspondente foi registrada da seguinte forma:

```
Using Tool: create_server
SERVER_NAME=crewai-2025-1
IMAGE=62dee28f-987d-40f5-a308-051d59991da8
FLAVOR=general.small
NETWORK_NAME=provider
```

O `OpenStackMCPServer` encaminhou a requisição ao OpenStack, criando a instância e monitorando seu progresso:

```
Still waiting for resource ... state is BUILD
state is ACTIVE
IP: 10.5.8.8
```

O provisionamento da máquina virtual ocorreu com sucesso, consolidando a funcionalidade essencial da arquitetura quanto à orquestração autônoma de recursos computacionais. Como forma de facilitar a compreensão da interação entre os componentes internos responsáveis pelo provisionamento da instância, a Figura 2 apresenta o fluxo de mensagens completo observado durante o provisionamento desta, conforme registrado na execução experimental.

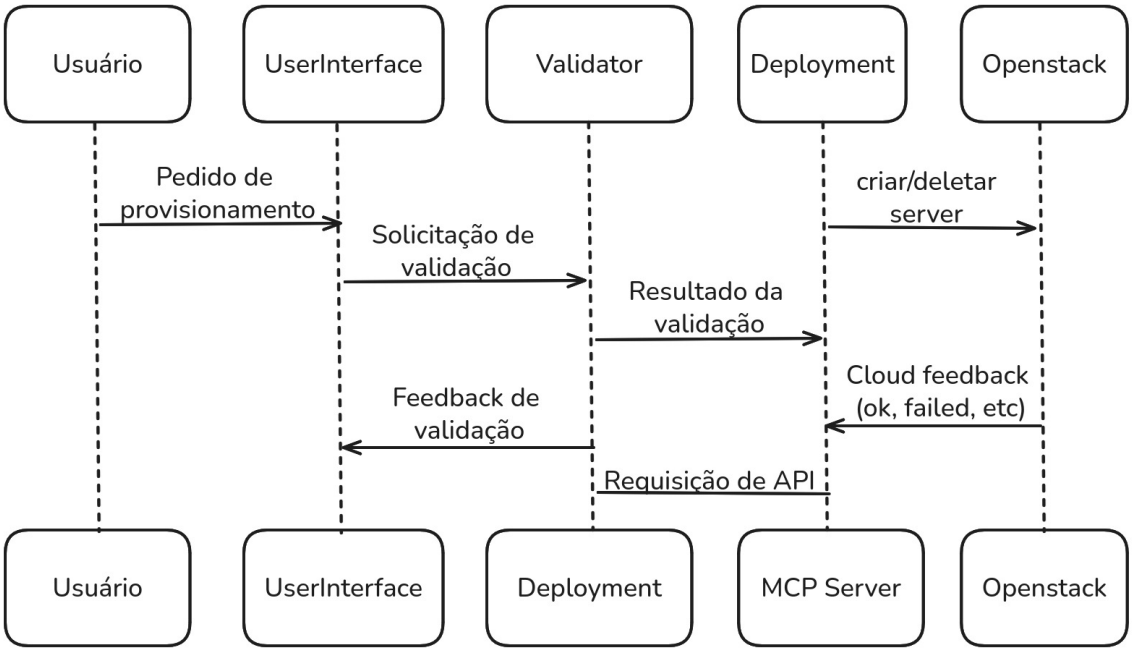

Figura 2. Fluxo detalhado de mensagens entre os agentes e o `OpenStackMCPServer` durante o provisionamento.

A figura evidencia, de forma clara, como o fluxo multiagente se organiza durante o provisionamento: o agente *UserInterface* inicia a requisição com base na intenção expressa pelo usuário; o agente *Validator* realiza autenticação, coleta e validação de recursos; e o agente *Deployment* conduz a etapa de criação da instância. O `OpenStackMCPServer` atua como mediador central, traduzindo cada solicitação

em chamadas diretas ao SDK do OpenStack e retornando respostas estruturadas aos agentes. Essa visualização reforça a interpretação dos resultados apresentados, demonstrando que a lógica de decisão distribuída se apoia em um encadeamento ordenado de mensagens, no qual cada agente contribui com uma etapa do processo. O diagrama também permite observar como a arquitetura incorpora tanto elementos reativos (respostas a eventos) quanto deliberativos (decisões tomadas pela LLM), constituindo uma cadeia de raciocínio autônoma alinhada ao modelo de agentes situados.

5.5. Avaliação da Tentativa de Deleção de Instância

Após o provisionamento, a *task* final previa a remoção da instância criada. Entretanto, os *logs* revelam um comportamento relevante do ponto de vista de confiabilidade: a LLM associou incorretamente o nome `crewai-2025-1` ao identificador de outra instância preexistente:

```
Using Tool: delete_server
SERVER_ID=549e60da-e31b-4691-846d-9b699e909a48
```

Esse identificador corresponde à instância `ministack-compute`, um recurso de produção que não tem relação com o experimento. A tentativa de remoção não surtiu efeito prático, conforme indicado pelo retorno da API, que manteve o estado do recurso inalterado. Ainda assim, o episódio evidencia uma limitação importante de sistemas de tomada de decisão baseados em LLM: a dificuldade em mapear, de forma determinística e segura, nomes e identificadores em ambientes complexos. As implicações dessa limitação são discutidas em maior detalhe na Seção 6.

Com base nos *timestamps* registrados, foi possível estimar a latência das operações:

- autenticação: aproximadamente 1 s;
- descoberta de recursos: entre 3 e 5 s;
- criação da instância: entre 7 e 10 s;
- tentativas de deleção: latência desprezível.

Os valores observados são compatíveis com o comportamento típico de infraestruturas OpenStack em ambientes de produção, caracterizados por tempos de provisionamento superiores aos de ambientes de simulação.

5.6. Síntese Observacional

Os resultados obtidos permitem sintetizar que:

1. o fluxo de criação de uma instância é executado de forma autônoma e bem-sucedida, sem intervenção humana direta após a configuração inicial;
2. a arquitetura multiagente apresenta comportamento consistente com o modelo proposto, distribuindo adequadamente tarefas entre interpretação, validação e execução;
3. o `OpenStackMCPServer` desempenha papel central na abstração da complexidade da API, tratando erros e adaptando consultas quando necessário;

4. a etapa de deleção evidencia limitações operacionais relevantes associadas à tomada de decisão semântica da LLM, com potencial risco para ambientes produtivos.

Em conjunto, a execução experimental valida a viabilidade da abordagem em relação aos critérios definidos na Seção 4, ao mesmo tempo em que evidencia pontos de atenção que orientam refinamentos arquiteturais discutidos na seção seguinte.

A Tabela 2 apresenta um comparativo entre Terraform, Ansible e o `ctrlStack`, evidenciando diferenças práticas quanto ao nível de abstração, esforço operacional, requisitos técnicos e capacidades de automação.

Tabela 2. Comparativo entre Terraform, Ansible e `ctrlStack`

Dimensão	**Terraform**	**Ansible**	**ctrlStack**
Modo de interação	HCL declarativo	Playbooks declarativos	Linguagem natural
Nível aprendizado	Alto	Alto	Baixo
Esforço operacional	Alto (arquivos extensos)	Alto (templates longos)	Baixo (1–2 frases)
Busca por recursos	Manual	Manual	Automática via MCP
Autonomia	Baixa	Baixa	Alta (agentes + LLM)
Nível de abstração	Baixo (declarativo)	Baixo–médio	Alto (intenção → ação)
Integração com Openstack	Via provider	Nativa (API)	SDK + MCP (abstração direta)

A comparação evidencia que o `ctrlStack` opera em um nível de abstração significativamente superior ao Terraform e ao Ansible, reduzindo substancialmente o esforço cognitivo e operacional necessário para a realização de tarefas básicas de provisionamento. Enquanto as abordagens declarativas exigem que o operador conheça sintaxe específica, identifique manualmente recursos e mantenha arquivos de configuração extensos, o `ctrlStack` centraliza essas responsabilidades nos agentes, que realizam descoberta automática de recursos, validação de contexto e resolução de parâmetros.

Além disso, diferentemente das ferramentas tradicionais, o `ctrlStack` incorpora um componente semântico capaz de interpretar intenções em linguagem natural e traduzir tais intenções em ações concretas. Essa característica reduz a barreira de entrada e permite que usuários com menor familiaridade técnica interajam com a infraestrutura de forma mais intuitiva. Por fim, a utilização do MCP como plano de controle contribui para reduzir latências intermediárias e simplificar o fluxo de chamadas ao OpenStack, proporcionando um mecanismo de orquestração mais ágil e adaptável.

6. Discussão

A análise dos resultados obtidos evidencia que o `ctrlStack`, em conjunto com o plano de controle `OpenStackMCPServer`, é capaz de executar de forma autônoma um fluxo completo de orquestração em um ambiente OpenStack real. As etapas de autenticação, descoberta de recursos e criação de instâncias mostraram-se robustas e coerentes com o modelo arquitetural proposto, confirmando a hipótese de que uma arquitetura multiagente, apoiada por MCP, pode traduzir intenções em linguagem natural em operações concretas de provisionamento.

Ao mesmo tempo, os experimentos revelam limitações importantes, tanto do ponto de vista da autonomia dos agentes quanto da confiabilidade operacional em infraestruturas de produção. Esta seção discute criticamente tais aspectos, à luz dos critérios de avaliação definidos na Seção 4, destacando implicações e oportunidades de aprimoramento.

A execução experimental demonstrou que os agentes são capazes de interpretar corretamente intenções expressas em linguagem natural e decompor essas intenções em ações concretas de orquestração. As decisões relacionadas à escolha do *flavor*, da imagem e da rede — fundamentadas nas regras declarativas presentes nas *tasks* — foram corretamente inferidas pela LLM, resultando na criação autônoma de uma instância em conformidade com as expectativas. Esse comportamento reforça o potencial de combinação entre LLMs e sistemas multiagentes para elevar o nível de abstração na interação com infraestruturas de nuvem, atendendo ao critério de autonomia de orquestração.

Por outro lado, a etapa de deleção da instância expõe uma limitação fundamental das arquiteturas baseadas em modelos de linguagem: a dificuldade em realizar mapeamentos determinísticos entre entidades simbólicas (nomes, rótulos, descrições) e identificadores técnicos presentes no ambiente operacional. Embora a VM criada estivesse corretamente registrada no ambiente OpenStack, a LLM associou esse recurso a outra instância preexistente, resultando em uma tentativa de deleção indevida. Trata-se de um exemplo de *alucinação operacional*, em que o modelo produz uma inferência semanticamente plausível, porém incompatível com o estado real da infraestrutura — fenômeno já apontado na literatura como um desafio em contextos que exigem alta precisão operacional.

Esse tipo de inconsistência reforça a necessidade de incorporar mecanismos de verificação formal no plano de controle. Delegar exclusivamente às LLMs a decisão final sobre a associação entre nomes e VMs representa um risco significativo para ambientes de produção, sobretudo quando tais operações possuem potencial destrutivo. A ausência de mecanismos de confirmação ou validação cruzada torna o sistema vulnerável a decisões incorretas, mesmo quando o fluxo de raciocínio aparenta coerência superficial. Nesse sentido, o critério de confiabilidade em operações sensíveis evidencia que a autonomia obtida ainda não é suficiente para uso desassistido em contextos críticos.

Os experimentos também evidenciaram o papel central do `OpenStackMCPServer` na estabilidade global da arquitetura. O plano de controle encapsulou com sucesso chamadas REST complexas, lidou com inconsistências

internas da API (como a necessidade de consultas alternativas para redes) e preservou o contexto entre requisições por meio do gerenciamento de sessões. Todavia, a tentativa de remoção indevida expõe uma lacuna crítica: o `OpenStackMCPServer` executou a operação solicitada pelo agente sem realizar verificações de segurança adicionais em nível de política.

Embora, no experimento, a nuvem tenha rejeitado implicitamente a deleção por razões ambientais, nada impede que, em outro cenário, uma operação semelhante resulte na exclusão real de recursos sensíveis. Observa-se, portanto, que a arquitetura atual delega ao MCP a orquestração técnica, mas não impõe garantias de integridade ou políticas de segurança que limitem o escopo das operações autorizadas aos agentes.

A ausência de mecanismos internos de validação — tais como verificação de conformidade entre `SERVER_NAME` e `SERVER_ID`, checagem de padrões de nomenclatura, confirmações explícitas em operações destrutivas ou restrições contextuais que impeçam ações fora do escopo da sessão — representa uma limitação arquitetural que deve ser abordada em versões futuras. A introdução de *guardrails* no `OpenStackMCPServer`, combinada a estratégias de *tooling* mais restritivas (por exemplo, separação entre ferramentas de leitura e de escrita, ou exigência de múltiplas evidências antes de operações destrutivas), tende a mitigar tais riscos e fortalecer a robustez do plano de controle.

De forma geral, a execução experimental confirma que o `ctrlStack` atinge seu objetivo principal de demonstrar a viabilidade de um sistema multiagente para orquestração autônoma de recursos OpenStack, integrando LLMs, MCP e o SDK oficial da plataforma. Ao mesmo tempo, a análise crítica dos resultados mostra que a autonomia proporcionada pelo modelo deve ser acompanhada de controles rigorosos, a fim de evitar comportamentos inadequados ou potencialmente danosos.

Assim, a principal contribuição empírica deste estudo consiste não apenas em validar a operação do `ctrlStack`, mas em explicitar — com base em evidências reais — os desafios e limitações inerentes à integração entre sistemas multiagentes, modelos de linguagem e infraestrutura de nuvem. Essa discussão fundamenta as direções propostas para trabalhos futuros, abordadas na próxima seção.

7. Conclusão e Trabalhos Futuros

O presente trabalho investigou a viabilidade de uma arquitetura multiagente orientada à interpretação semântica e à orquestração autônoma de recursos em ambientes OpenStack, resultando no desenvolvimento do `ctrlStack` e do plano de controle `OpenStackMCPServer`. A proposta integrou técnicas contemporâneas de Inteligência Artificial, em particular modelos de linguagem de grande porte, com os princípios dos Sistemas Multiagentes (MAS) e com os mecanismos de provisionamento disponibilizados pelo SDK oficial do OpenStack. Essa convergência foi explorada com o objetivo de reduzir a complexidade operacional associada ao gerenciamento de infraestrutura em nuvens privadas, elevando o nível de abstração disponível para os usuários.

Os resultados empíricos demonstraram que a arquitetura concebida é capaz de executar, de forma integralmente autônoma, um fluxo completo de orquestração composto por autenticação, descoberta de recursos e criação de máquinas virtuais em um ambiente real de produção. A interação coordenada entre os agentes mostrou-se consistente com os princípios teóricos subjacentes aos sistemas multiagentes, permitindo que tarefas complexas fossem decompostas, validadas e executadas com grau significativo de autonomia. Esse comportamento confirma que o uso combinado de LLMs e MAS constitui uma alternativa promissora para automatizar processos tradicionalmente dependentes de intervenção humana altamente especializada.

O plano de controle implementado no `OpenStackMCPServer` revelou-se um componente central para a efetividade da arquitetura. Sua capacidade de encapsular operações do SDK, manter sessões autenticadas, tratar exceções e oferecer uma interface unificada para os agentes reforça a ideia de que camadas intermediárias desse tipo são fundamentais para viabilizar fluxos autônomos de orquestração. A padronização proporcionada pelo protocolo MCP confere interoperabilidade e flexibilidade adicionais, tornando o sistema mais modular e alinhado às tendências modernas de integração entre agentes inteligentes e serviços externos.

Por outro lado, os experimentos também evidenciaram limitações importantes, cuja análise crítica oferece insumos valiosos para a evolução da proposta. O caso observado na tentativa de deleção de uma instância — em que o modelo de linguagem associou incorretamente um nome a um identificador de recurso — revelou um problema estrutural na forma como LLMs manipulam referências simbólicas em ambientes dinâmicos e sensíveis. Embora o fluxo de criação da instância tenha funcionado adequadamente, a falha de mapeamento na operação destrutiva destaca a necessidade de tratar com rigor as fronteiras entre interpretação semântica e ações de impacto sobre o ambiente físico ou virtual. Esse resultado converge com discussões recentes na literatura sobre as limitações de confiabilidade das LLMs em contextos que exigem precisão operacional elevada, reforçando a importância de mecanismos de validação rigorosos, especialmente em operações potencialmente destrutivas.

A partir das evidências empíricas obtidas, torna-se claro que a autonomia proporcionada pelos agentes deve ser complementada por políticas de controle robustas implementadas no plano de controle. Embora o `OpenStackMCPServer` já ofereça abstrações relevantes, a ausência de verificações explícitas sobre a correspondência entre nomes e identificadores, de restrições baseadas em padrões

de nomenclatura e de confirmações adicionais antes da execução de ações sensíveis representa riscos concretos que precisam ser mitigados. Sem esses mecanismos, a responsabilidade por evitar comportamentos indesejados recai excessivamente sobre a lógica probabilística da LLM, o que não é desejável em ambientes críticos.

Nesse sentido, as contribuições deste trabalho podem ser sintetizadas em dois eixos principais: (i) a demonstração da viabilidade técnica de integrar agentes autônomos com sistemas de provisionamento em nuvem, construindo um fluxo operacional completo e funcional; e (ii) a explicitação, com base em evidências reais, dos desafios inerentes à adoção dessa abordagem em contextos de produção, incluindo riscos, limitações e requisitos adicionais de segurança e confiabilidade. Assim, o `ctrlStack` não apenas valida um conceito, mas também abre caminho para discussões mais amplas sobre governança, verificação e controle em sistemas de orquestração inteligentes.

Como desdobramentos futuros, destacam-se algumas direções prioritárias:

- **Fortalecimento do plano de controle**: introdução de mecanismos de validação formal no `OpenStackMCPServer`, capazes de impedir ações incompatíveis com o estado real do ambiente ou com as políticas definidas pelo administrador, em especial para operações destrutivas;
- **Camadas adicionais de verificação semântica**: desenvolvimento de verificadores simbólicos ou módulos auxiliares que atuem como filtros para decisões produzidas pela LLM, com foco na interpretação de identificadores e na confirmação de intenções antes da execução de ações de alto impacto;
- **Ampliação dos fluxos suportados**: extensão da arquitetura para contemplar cenários mais sofisticados, tais como escalonamento automático, realocação dinâmica de recursos, detecção de anomalias e recuperação autônoma de falhas, explorando o potencial dos agentes em tarefas contínuas de otimização;
- **Avaliações em cenários ampliados**: realização de experimentos adicionais em ambientes com maior carga e diversidade de recursos, bem como análises comparativas com orquestradores tradicionais, incluindo métricas de esforço operacional, tempo de provisionamento e impacto em ambientes multiinquilino sensíveis.

Em conjunto, tais avanços tendem a consolidar a proposta apresentada como um arcabouço sólido para orquestração inteligente em nuvens privadas, aproximando a operação de ambientes OpenStack das possibilidades emergentes trazidas pela Inteligência Artificial e pelos sistemas multiagentes. Espera-se que os resultados e reflexões apresentados neste trabalho sirvam de base para pesquisas posteriores na área, contribuindo para o desenvolvimento de soluções mais seguras, confiáveis e alinhadas às necessidades de infraestruturas modernas.

A. Exemplo de Task no `tasks.yaml`

Este apêndice apresenta um trecho simplificado do arquivo `tasks.yaml`, ilustrando como o fluxo de orquestração é descrito de forma declarativa e associado aos agentes definidos no `ctrlStack`.

```
crew:
  name: ctrlStack Crew
  agents:
    - user_interface_agent
    - validator_agent
    - deployment_agent

tasks:
  - name: login_to_cloud
    description: >
      Autenticar no OpenStack usando credenciais fornecidas
         pelo usuario.
    agent: validator_agent
    tools:
      - login

  - name: discover_resources
    description: >
      Listar flavors, redes e imagens dispon veis para a
         sess o atual.
    agent: validator_agent
    tools:
      - list_flavors
      - list_networks
      - list_images

  - name: create_server
    description: >
      Criar uma nova inst ncia seguindo as pol ticas
         definidas
      (menor flavor, imagem Ubuntu 22.04, rede provider).
    agent: deployment_agent
    tools:
      - create_server
```

Listing 1. Trecho simplificado do arquivo tasks.yaml

O fragmento do List. 1 evidencia que o fluxo de trabalho é descrito como uma sequência de *tasks* associadas a agentes específicos e às ferramentas MCP expostas pelo `OpenStackMCPServer`. Essa estrutura facilita a extensão do sistema com novos cenários de orquestração.

B. Pseudocódigo do Fluxo de Decisão da LLM

Este apêndice apresenta um pseudocódigo de alto nível que resume a lógica de decisão empregada pela LLM ao selecionar *flavor*, imagem, rede e nome da instância durante

o provisionamento automático.

```
procedure PROVISIONAR_INSTANCIA(session_id):
    # 1. Obter recursos dispon veis via MCP
    flavors  <- list_flavors(session_id)
    images   <- list_images(session_id)
    networks <- list_networks(session_id)

    # 2. Escolher o menor flavor dispon vel
    flavor_escolhido <- argmin(flavors, por_ram_vcpu_disk)

    # 3. Selecionar imagem preferencial
    if existe_imagem_com_nome(images, "ubuntu-22.04"):
        imagem_escolhida <- "ubuntu-22.04"
    else:
        imagem_escolhida <- escolher_imagem_generica(images)

    # 4. Selecionar rede provider
    if existe_rede_com_nome(networks, "provider"):
        rede_escolhida <- "provider"
    else:
        rede_escolhida <- escolher_rede_padrao(networks)

    # 5. Definir nome da inst ncia com sufixo incremental
    instancias_existentes <- list_servers(session_id)
    proximo_id <- calcular_proximo_id(instancias_existentes,
       prefixo="crewai-2025-")
    nome_instancia <- "crewai-2025-" + proximo_id

    # 6. Invocar tool MCP de cria  o
    create_server(
        session_id     = session_id,
        SERVER_NAME    = nome_instancia,
        IMAGE          = imagem_escolhida,
        FLAVOR         = flavor_escolhido,
        NETWORK_NAME   = rede_escolhida,
        KEYPAIR_NAME   = "crewai-key"
    )
end procedure
```

Listing 2. Pseudocódigo do fluxo de decisão para criação de instância

O pseudocódigo do List. 2 não reflete a implementação literal, mas resume a política de decisão usada pela LLM a partir das respostas do MCP, evidenciando que o modelo opera sobre dados estruturados e aplica regras simples de seleção antes de acionar a ferramenta `create_server`.

C. Esquemas Simplificados de Mensagens MCP

Por fim, este apêndice apresenta esquemas JSON simplificados que ilustram o formato das mensagens trocadas entre os agentes do `ctrlStack` e o `OpenStackMCPServer`.

Os exemplos não cobrem todos os campos da implementação real, mas capturam os elementos essenciais utilizados nos experimentos.

Requisição de Login

```
{
  "tool": "login",
  "args": {
    "cloud_provider_url": "https://cloud.lsd.ufcg.edu.br",
    "app_creds": "ID_DA_APPLICATION_CREDENTIAL",
    "app_secret": "SEGREDO_DA_APPLICATION_CREDENTIAL",
    "domain_name": "lsd"
  }
}
```

Listing 3. Formato simplificado da requisição de login

Resposta de Login

```
{
  "session_id": "01d2818e-2cff-4b69-b0b8-c0e67cc4e447"
}
```

Listing 4. Formato simplificado da resposta de login

Resposta da Tool `list_flavors`

```
[
  {
    "name": "general.small",
    "vcpus": 1,
    "ram": 2048,
    "disk": 20
  },
  {
    "name": "general.medium",
    "vcpus": 2,
    "ram": 4096,
    "disk": 40
  }
]
```

Listing 5. Formato simplificado de retorno de flavors

Requisição para `create_server`

```
{
  "tool": "create_server",
  "args": {
    "session_id": "01d2818e-2cff-4b69-b0b8-c0e67cc4e447",
    "SERVER_NAME": "crewai-2025-1",
    "IMAGE": "ubuntu-22.04",
    "FLAVOR": "general.small",
    "NETWORK_NAME": "provider",
    "KEYPAIR_NAME": "crewai-key"
  }
}
```

Listing 6. Formato simplificado da requisição de criação de instância

Os esquemas dos Lists. 3–6 ilustram como o `OpenStackMCPServer` expõe operações de alto nível em formato JSON simples, o que facilita o consumo das tools pelos agentes do `ctrlStack` e reforça o papel do MCP como camada de integração padronizada entre a LLM e a API nativa do OpenStack.

Referências

Konstantinos Antonakoglou, Ioannis Mavromatis, Saptarshi Ghosh, Mark Rouse, and Konstantinos Katsaros. Camino: Cloud-native autonomous management and intent-based orchestrator, 2025. arXiv preprint.

Y. Dang et al. Multi-agent collaboration via evolving orchestration, 2025. arXiv preprint.

James Denton. *Learning OpenStack Networking (Neutron)*. Packt Publishing, Birmingham, 2019.

J. Lowin et al. Fastmcp: Building mcp servers in python. Documentation and GitHub repository, 2024. `https://github.com/jlowin/fastmcp` (accessed Oct 2025).

OpenInfra Foundation. Openstack sdk user guide. Documentation, 2023. `https://docs.openstack.org/openstacksdk/latest/` (accessed Oct 2025).

CrewAI Inc. Crewai: A lean python framework for autonomous multi-agent orchestration. GitHub repository and documentation, 2025. `https://github.com/crewAIInc/crewAI` (accessed Oct 2025).

Nicholas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001. doi: 10.1145/374308.374353.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 3rd (draft) edition, 2023. URL `https://web.stanford.edu/~jurafsky/slp3/`.

X. Li, H. Wang, and Y. Zhang. A review of ai and multi-agent systems for cloud performance and security. *Journal of Cloud Computing*, 13(2), 2024.

Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical Report SP 800-145, National Institute of Standards and Technology (NIST), 2011. URL `https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf`.

R. Singh et al. Agentflow: Resilient adaptive cloud-edge framework for multi-agent coordination, 2025. arXiv preprint.

Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajkumar Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, 2016. doi: 10.1109/MCC.2016.132.

Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2 edition, 2009.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Campina Grande - Código INEP: 25137409
R. Tranqüilino Coelho Lemos, 671, Dinamérica, CEP 58432-300, Campina Grande (PB)
CNPJ: 10.783.898/0003-37 - Telefone: (83) 2102.6200

Documento Digitalizado Ostensivo (Público)

Versão final TCC

Assunto:	Versão final TCC
Assinado por:	Igor Oliveira
Tipo do Documento:	Anexo
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Igor Kadson de Souza Oliveira**, **DISCENTE (202111250003) DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO - CAMPINA GRANDE**, em 04/03/2026 13:52:45.

Este documento foi armazenado no SUAP em 04/03/2026. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1788339
Código de Autenticação: 779cb39418