Análise de Dependabilidade em Nuvens OpenStack Multi-Nó via Injeção de Falhas Geradas por IA

Guilherme Silva Duarte[1], **Erica Teixeira Gomes de Sousa**[1]

[1]Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)

guilherme.silvad@ufrpe.br, erica.sousa@ufrpe.br

Resumo. *A expansão das nuvens privadas distribuídas torna essencial a avaliação de dependabilidade, sobretudo quanto à propagação de erros entre nós. Este trabalho analisa de forma experimental essa propagação em uma nuvem OpenStack multi-nó. As falhas são injetadas de forma automatizada e geradas por Inteligência Artificial (IA). Cada falha é uma mutação semântica derivada de um* bug *real. Os resultados indicam que 94,4% das execuções produziram* Falhas Cinzentas Distribuídas*: os serviços permaneceram ativos no monitoramento, enquanto a infraestrutura ficou funcionalmente incapaz de provisionar instâncias. Esse comportamento revela uma classe de ameaças invisível à observabilidade tradicional.*

1. Introdução

A Computação em Nuvem consolidou-se como base operacional de cargas científicas e empresariais, e o OpenStack ocupa posição de destaque entre as plataformas de nuvem privada de código aberto. Oferecendo Infraestrutura como Serviço (IaaS), a plataforma gerencia mais de 40 milhões de núcleos em produção [OpenInfra Foundation 2023] e é adotada em centros de computação de Alto Desempenho (HPC), como o CERN. Sua relevância é reforçada pelos 72% de adoção de nuvem híbrida reportados pelo *State of the Cloud Report 2024* [Flexera 2024].

Em produção, ambientes OpenStack operam de forma distribuída e multi-nó, com os serviços de controle, computação e rede segregados em hosts físicos distintos, comunicando-se por mensageria assíncrona via RPC e por APIs REST. Essa arquitetura introduz a propagação de erros entre nós (*cross-node error propagation*), na qual uma falhas migra silenciosamente entre serviços acoplados [Cotroneo et al. 2019]. O resultado pode ser uma Falha Cinzenta (*Gray Failure*): o serviço aparenta operacional, mas está funcionalmente inoperante para o *tenant* [Natella et al. 2013].

A Engenharia do Caos investiga a injeção deliberada de falhas como prática de validação [Basiri et al. 2016, Ojdanic et al. 2023], e trabalhos recentes empregam Modelos de Linguagem de Grande Escala (LLMs) para gerá-las [Cotroneo and Liguori 2024, Liguori et al. 2024]. Esses estudos, contudo, concentram-se em ambientes monolíticos, suprimindo justamente os mecanismos de propagação entre nós.

Diante desse cenário, este trabalho avalia experimentalmente a propagação de falhas em uma infraestrutura OpenStack distribuída multi-nó. A avaliação emprega injeção automatizada de falhas semânticas geradas por IA. Para isso, utiliza-se a ferramenta CIMut (*Cloud Injection Mutator*), proposta em trabalho anterior dos autores [Duarte et al. 2026]. A contribuição do presente trabalho não é a ferramenta em si,

mas a sua aplicação a um ambiente distribuído realista e a análise empírica da propagação de erros resultante. São contribuições deste trabalho: (i) a caracterização empírica de *Falhas Cinzentas Distribuídas* em uma nuvem OpenStack multi-nó; (ii) a identificação de três padrões recorrentes de propagação de falhas entre nós; e (iii) evidências de que mutações semânticas geradas por IA produzem falhas invisíveis ao monitoramento tradicional.

A Seção 2 apresenta o referencial teórico e os trabalhos relacionados. A Seção 3 descreve a ferramenta CIMut. A Seção 4 detalha o estudo de caso. A Seção 5 discute os resultados obtidos. A Seção 6 apresenta as conclusões e os trabalhos futuros.

2. Referencial Teórico e Trabalhos Relacionados

A taxonomia de dependabilidade de [Avizienis et al. 2004] estabelece os atributos fundamentais de sistemas confiáveis, mas em ambientes distribuídos precisa acomodar a propagação de erros entre componentes acoplados por comunicação assíncrona [Natella et al. 2016]. A Engenharia do Caos formaliza a injeção deliberada de falhas como prática de validação [Basiri et al. 2016], tendo a mutação de código como técnica predominante [Jia and Harman 2011]. Como os operadores sintáticos clássicos não capturam *bugs* reais [Ojdanic et al. 2023], cresce o uso LLMs para gerá-los [Fan et al. 2023].

A injeção de falhas no OpenStack remonta a [Ju et al. 2013], que alvejaram a comunicação entre serviços. [Cotroneo et al. 2019] expandiram essa abordagem com mutação de código em ambiente *all-in-one* (todos os serviços em um único *host*) e revelaram que a maioria das falhas se propagava silenciosamente entre componentes. Posteriormente, propuseram o ThorFI [Cotroneo et al. 2022], restrito à camada de rede virtual. Já [Cotroneo and Liguori 2024] demonstraram que falhas semânticas geradas por IA são mais representativas que as sintáticas, e [Liguori et al. 2024] que o *Context-Aware Prompting*, enriquecer a instrução com contexto do código-alvo, maximiza essa geração. Esses estudos, no entanto, foram conduzidos predominantemente em ambientes monolíticos, suprimindo os mecanismos de propagação entre nós, lacuna que este trabalho aborda.

3. Cloud Injection Mutator (CIMut)

A CIMut foi proposta em trabalho anterior dos autores [Duarte et al. 2026]. A ferramenta automatiza a geração e a aplicação de mutações semânticas. Uma mutação semântica é uma alteração que preserva a validade sintática do código, mas modifica seu comportamento lógico. A automação ocorre por meio de um *pipeline* de Geração Aumentada por Recuperação (*Retrieval-Augmented Generation*, RAG) acoplado a um agente de execução residente. Uma base vetorial mapeia os arquivos e as funções críticas do OpenStack. A partir de uma intenção em linguagem natural (*User Query*), o sistema recupera os cinco contextos arquiteturais mais relevantes. Esses contextos enriquecem o *prompt* enviado ao LLM por meio de *Context-Aware Prompting* [Liguori et al. 2024]. O resultado é uma cadeia de um a cinco passos. Cada passo é executado em dois estágios. O primeiro, *Read*, extrai o código-fonte atual no contêiner Kolla e o resubmete ao LLM para determinar a injeção. O segundo, *Modify*, aplica a alteração no servidor.

4. Estudo de Caso

Este estudo de caso analisa a propagação de erros entre nós segregados em um ambiente OpenStack multi-nó, provisionado com Kolla-Ansible. As falhas semânticas foram geradas automaticamente pela CIMut, operada pelo modelo Claude Sonnet 4.6 (temperatura 0,5).

A topologia experimental segrega as funções em três nós virtuais hospedados em VirtualBox: o nó controller concentra o controle (Nova API/Conductor/Scheduler, Neutron Server, Glance, Keystone, Heat, Placement, MariaDB, RabbitMQ e Memcached); o network hospeda a rede (agentes Neutron OVS/DHCP/L3/Metadata, HAProxy e Keepalived); e o compute01 executa a computação (Nova Compute e libvirt). Os componentes Cinder e Swift não foram incluídos no escopo experimental.

A carga de teste partiu de *bugs* reais do Red Hat Bugzilla, selecionados pelo histórico de propagação entre serviços (*cross-service*). Cada *bug* foi reescrito como uma intenção estratégica curta, sem prescreve arquivo, linha ou técnica (o LLM decidiu de forma autônoma onde e como atacar). A Tabela 1 sintetiza os nove cenários em quatro grupos arquiteturais.

Tabela 1. Cenários de teste derivados do Red Hat Bugzilla.

ID	Grupo	Intenção estratégica
IN-01	A: Build/Spawn	Falha ao conectar interface de rede ao criar VM
IN-02	A: Build/Spawn	Falha no daemon libvirt durante *build* de instâncias
IN-03	A: Build/Spawn	Inconsistência na contagem de instâncias ativas
IN-04	B: Conductor/Scheduler	*Timeout* no agendamento impedindo seleção de *hosts*
IN-05	B: Conductor/Scheduler	Falha em *ResourceProvider* ao migrar instâncias
IN-07	C: Hypervisor	Condição de corrida durante exclusão de servidores
IN-08	C: Hypervisor	Travamento no *reboot* de instância multi-vCPU
IN-09	D: Cross-cutting	Degradação de *performance* na API Nova
IN-10	D: Cross-cutting	Falha de comunicação RPC no provisionamento

Cada cenário foi submetido em duas rodadas (R1 e R2), totalizando 18 execuções, com o ambiente restaurado ao estado inicial entre elas, o que permite uma avaliação preliminar de variância.

A avaliação baseou-se em quatro métricas, adaptadas da taxonomia de dependabilidade ao contexto multi-nó [Avizienis et al. 2004, Cotroneo et al. 2019]: (i) ocorrência de falha funcional, mesmo sem *crash* de processo; (ii) profundidade da cadeia, o número de mutações coordenadas pelo LLM; (iii) distribuição entre nós, o conjunto de nós atingidos; e (iv) alteração de *status*, indicador binário de *crash failure* observável pelo monitoramento dos contêineres Kolla [Cotroneo et al. 2019].

5. Análise de Resultados

A CIMut, operada pelo Claude Sonnet 4.6, gerou cadeias com média de 2,8 ($\pm$ 1,1) mutações por execução. Em todas as 18 execuções, as mutações induziram falhas na funcionalidade do provisionamento de instâncias. Todas as mutações foram semânticas (inversão de operadores, substituição de constantes, reordenação de parâmetros, troca de exceção), sem alucinações posicionais. A Tabela 2 consolida os resultados por rodada (R1 e R2).

Tabela 2. Síntese consolidada das 18 execuções por cenário.

Input	**Falha funcional** (R1/R2)	**Cadeia** (R1/R2)	**Nós atingidos**	***Status* alterado**
IN-01	Sim/Sim	3/4	Controller, Compute, Network	Não
IN-02	Sim/Sim	2/4	Controller, Network, Compute	Não
IN-03	Sim/Sim	2/1	Controller, Network	Não
IN-04	Sim/Sim	4/2	Controller, Network	Não
IN-05	Sim/Sim	2/2	Controller, Compute	Não
IN-07	Sim/Sim	4/3	Controller	Não
IN-08	Sim/Sim	1/1	Controller	Não
IN-09	Sim/Sim	3/3	Controller	**nova_scheduler: failed**
IN-10	Sim/Sim	2/2	Controller, Network	Não
Total	**18/18**	**2,8 ± 1,1**	—	**1/18**

As cadeias revelaram três padrões arquiteturais. O primeiro (IN-01, IN-02, IN-04) atravessa os três nós: inicia no plano de controle (Conductor, Glance API), propaga-se para a computação (*nova-compute*, libvirt) e atinge a rede (*neutron-l3-agent*, *neutron-openvswitch-agent*), tornando o estado irrecuperável sem intervenção manual. O segundo (IN-03, IN-05, IN-10) restringe-se a dois nós, explorando a interface RPC entre o Controlador e um nó dependente. O terceiro (IN-07, IN-08, IN-09) concentra as mutações no Controlador, compromete serviços compartilhados (Keystone, Nova API, Heat). O caso mais severo foi o IN-09: a saturação de *threads* WSGI (*Web Server Gateway Interface*) da Nova API derrubou o `nova_scheduler` (estado `failed`), o único *crash failure* observado.

O principal resultado é a ocorrência sistemática de *Falhas Cinzentas Distribuídas*. Em 17 das 18 execuções (94,4%), todos os 21 serviços do *deployment* permaneceram com *status* `active` no monitoramento, embora a infraestrutura estivesse funcionalmente incapaz de provisionar instâncias, uma classe de ameaça invisível aos *health checks*. Embora as cadeias de R1 e R2 não fossem idênticas, o efeito sistêmico foi consistente em todos os *inputs*, sugerindo que o *pipeline* RAG converge para soluções semanticamente equivalentes. Os cenários de propagação ampla saturaram a CPU do Controlador (90–100% em IN-05, IN-07, IN-10 e IN-04), efeito relevante para infraestruturas de HPC.

6. Conclusão

Este trabalho analisou a propagação de erros gerados por IA em ambientes OpenStack multi-nó, revelando três padrões consistentes: *cross-node* integral, propagação binária via RPC e concentração no plano de controle. Destaca-se a caracterização empírica das *Falhas Cinzentas Distribuídas*, em que mutações lógicas corromperam a regra de negócio dos serviços sem derrubar seus processos. Antes associado a falhas orgânicas em sistemas isolados [Natella et al. 2013], o fenômeno foi aqui reproduzido em um ambiente distribuído realista, com implicações diretas para a observabilidade semântica em HPC.

Como trabalhos futuros, pretende-se incluir Cinder e Swift no escopo; ampliar a topologia; comparar diferentes LLMs e aumentar o número de rodadas para uma análise de variância mais robusta.

Referências

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Basiri, A., Behnam, N., de Graaff, R., Hochstein, L., Kosewski, L., Reynolds, J., and Rosenthal, C. (2016). Chaos engineering. *IEEE Software*, 33(3):35–41.

Cotroneo, D. and Liguori, P. (2024). Neural fault injection: Generating software faults from natural language. In *Proceedings of the 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, pages 23–27.

Cotroneo, D., Simone, L. D., Liguori, P., Natella, R., and Bidokhti, N. (2019). How bad can a bug get? an empirical analysis of software failures in the OpenStack cloud computing platform. In *Proc. of ESEC/FSE*, pages 200–211.

Cotroneo, D., Simone, L. D., and Natella, R. (2022). ThorFI: A novel approach for network fault injection as a service. *Journal of Network and Computer Applications*, 201:103334.

Duarte, G. S., de Sousa, E. T. G., and e Silva, C. M. N. (2026). Uma análise comparativa entre LLMs proprietários e de pesos abertos na injeção de falhas de nuvens privadas. In *Anais do XXVII Workshop de Testes e Tolerância a Falhas (WTF)*, pages 219–231, Porto Alegre, RS, Brasil. SBC.

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., and Zhang, J. M. (2023). Large language models for software engineering: Survey and open problems. In *Proc. of ICSE-FoSE*, pages 31–53.

Flexera (2024). 2024 state of the cloud report.

Jia, Y. and Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678.

Ju, X., Soares, L., Shin, K. G., Ryu, K. D., and Silva, D. D. (2013). On fault resilience of OpenStack. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC)*, pages 2:1–2:16.

Liguori, P., Improta, C., Natella, R., Cukic, B., and Cotroneo, D. (2024). Enhancing AI-based generation of software exploits with contextual information. In *Proceedings of the 35th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 180–191.

Natella, R., Cotroneo, D., and Madeira, H. S. (2013). On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, 39(1):80–96.

Natella, R., Cotroneo, D., and Madeira, H. S. (2016). Assessing dependability with software fault injection: A survey. *ACM Computing Surveys*, 48(3):1–55.

Ojdanic, M., Barr, E. T., Grunske, L., and Papadakis, M. (2023). Syntactic versus semantic similarity of artificial and real faults in mutation testing studies. *IEEE Transactions on Software Engineering*, 49(7):3922–3938.

OpenInfra Foundation (2023). 2023 OpenInfra user survey.