CIMut: Ferramenta de Injeção de Falhas em Ambientes de Nuvens por Mutação

Guilherme Silva Duarte[1]*, Erica Teixeira Gomes de Sousa[2]

[1] guilherme.silvad@ufrpe.br. Universidade Federal Rural de Pernambuco (UFRPE), Brasil. * autor correspondente
[2] erica.sousa@ufrpe.br. Universidade Federal Rural de Pernambuco (UFRPE), Brasil.

Resumo
A injeção de falhas de software é uma técnica valiosa para avaliar a resiliência de sistemas de software complexos. Ao introduzir falhas em componentes específicos, os pesquisadores podem observar como essas falhas se propagam e impactam o comportamento geral do sistema. Este artigo apresenta a ferramenta CIMut, desenvolvida para injeção de falhas por meio da mutação do código-fonte. A mutação do código-fonte possibilita a introdução de falhas para análise de sistemas. Um estudo experimental abrangente realizado no OpenStack, uma plataforma de computação em nuvem de código aberto, demonstra a eficácia da ferramenta. Foram realizados mais de 62 experimentos no OpenStack, cada um injetando falhas em diferentes componentes do sistema. Os resultados do estudo foram promissores. Uma parcela significativa (até 96,7%) das falhas injetadas resultou em bugs, classificados como erros explícitos (travamentos, exceções) ou bugs com impacto funcional (comportamento incorreto, perda de dados). Esses resultados demonstram que a ferramenta CIMut é capaz de gerar falhas representativas que podem ser utilizadas para avaliar a resiliência de sistemas de software complexos como o OpenStack.
Palavras-chave: Ambiente de Nuvem; Dependabilidade; Injeção de falha de software; OpenStack; Sistemas tolerantes a falhas.

CIMut: A Mutation-Based Fault Injection Tool for Cloud Environments

Abstract
Software fault injection is a valuable technique for assessing the resilience of complex software systems. By deliberately introducing faults into specific components, researchers can observe how these faults propagate and impact the overall system behavior. This paper presents a comprehensive experimental study conducted on OpenStack, an open-source cloud computing platform. The study employed the CIMut tool, which was developed for mutation-based fault injection. Source code mutation enables the creation of realistic faults that simulate common programming errors. Over 62 experiments were performed on OpenStack, each injecting faults into different system components. The study results were promising. A significant portion (up to 96.7%) of injected faults resulted in observable bugs, classified as either explicit errors (crashes, exceptions) or bugs with functional impact (incorrect behavior, data loss). These findings demonstrate that the CIMut tool is capable of generating representative faults that can be used to evaluate the resilience of complex software systems like OpenStack.
Keywords: *Cloud Environment; Dependability; Software Fault Injection; OpenStack; Fault-Tolerant Systems.*

1 Introdução

A computação em nuvem revolucionou a forma como empresas e indivíduos utilizam recursos computacionais. Sua adoção abrange diversos setores, como negócios, redes sociais, computação científica e de alto desempenho (ABDERRAHIM, W.; CHOUKAIR, Z., 2016). A nuvem se caracteriza pela oferta de acesso ubíquo, conveniente e sob demanda a um conjunto compartilhado de recursos computacionais configuráveis. Esse modelo proporciona provisionamento e liberação rápidos de recursos, reduzindo o esforço de gerenciamento e a interação com o provedor de serviços (CORREIA, 2011).

A escalabilidade e flexibilidade inerentes à nuvem a tornaram uma tecnologia essencial para a economia moderna, facilitando a expansão de empresas em todo o mundo (CROUZET, Y.; KANOUN,

K., 2012). A migração para a nuvem está experimentando um crescimento significativo, e espera-se que essa tendência continue nos próximos anos (Li et al., 2020).

Apesar dos inúmeros benefícios, a computação em nuvem também enfrenta o desafio da ocorrência de falhas. Devido à interdependência entre os componentes da nuvem, uma falha em um único componente pode gerar impactos em cascata, afetando diversos outros. Os prejuízos anuais associados a falhas na nuvem são estimados em 700 bilhões de dólares (ABED, 2025).

Nesse contexto, a injeção de falhas se destaca como uma técnica fundamental para analisar a resiliência de sistemas de nuvem. Através da injeção controlada de falhas, é possível avaliar a capacidade do sistema em lidar com cenários adversos e desenvolver mecanismos de tolerância a falhas. A implementação desses mecanismos visa mitigar a ocorrência de falhas e garantir a continuidade das operações, proporcionando maior disponibilidade e confiabilidade do ambiente de nuvem (CROUZET, Y.; KANOUN, K., 2012).

Diante da importância da computação em nuvem e dos desafios impostos pela ocorrência de falhas, este trabalho introduz uma ferramenta para injeção de falhas por mutação em ambientes de nuvem. Essa ferramenta possibilita a análise do impacto das falhas nos serviços da nuvem, bem como a avaliação da resiliência e dependência entre seus módulos. Através da injeção controlada de falhas, a ferramenta permite identificar vulnerabilidades e desenvolver estratégias para mitigar seus efeitos, garantindo maior confiabilidade e disponibilidade do ambiente de nuvem.

A Seção 2 deste artigo traz os principais conceitos utilizados neste trabalho. Já a Seção 3 apresenta mais detalhes sobre trabalhos relacionados à injeção de falhas em ambientes de nuvem. A Seção 4 apresenta a ferramenta proposta, a Seção 5 detalha os experimentos que foram executados e a Seção 6 mostra a classificação de bugs. E a Seção 7 conclui este trabalho com considerações finais e trabalhos futuros.

2 Referencial teórico

Nesta seção serão definidos alguns conceitos importantes para o entendimento deste trabalho.

2.1 Computação em Nuvem

Computação em nuvem representa um modelo para disponibilizar recursos computacionais (computação, armazenamento e rede) de forma acessível e universal. Uma característica importante desse modelo é o provisionamento sob demanda, permitindo que os recursos sejam alocados de maneira prática conforme a necessidade (MELL, P.; GRANCE, T., 2011).

A computação em nuvem oferece diversos modelos de implantação para atender a diferentes necessidades de negócios. A nuvem privada garante exclusividade, com a infraestrutura provisionada para uma única organização, permitindo controle total sobre gerenciamento e operação, seja internamente ou por terceiros (CORREIA, 2011). Atualmente existem diversas alternativas de plataformas que permitem personalizar a configuração da nuvem e criar um ambiente customizável ideal para que o trabalho seja desenvolvido. Plataformas como OpenShift (HAT, 2025), Apache CloudStack (DIVYA, D.; HARSHITHA, K., 2025), OpenNebula (CHOWDHURY, 2025), Cloudify (VANKAYALAPATI, 2025) e Eucalyptus (SOLÍS et al., 2025) podem ser adotadas para criar um ambiente de nuvem privada.

O OpenStack é uma plataforma de nuvem open source que capacita os usuários a construir e gerenciar seus próprios ambientes de nuvem privada. Amplamente utilizada por grandes empresas como Blizzard Entertainment, Daum Communication, LINE Corporation e Adobe, o OpenStack oferece uma estrutura modular para atender a diversas necessidades da nuvem. O Nova permite a criação e gerenciamento de instâncias (máquinas virtuais), enquanto o Neutron gerencia as redes virtuais. O Cinder e o Swift gerenciam o armazenamento em volumes e objetos, respectivamente. Além disso, o OpenStack conta com funcionalidades como monitoramento (Ceilometer), automação (Heat) e gerenciamento de identidade (Keystone), fornecendo uma solução completa para a implementação e operação de nuvens privadas (OPENSTACK, 2025).

2.2 Injeção de Falhas

A dependabilidade é uma característica essencial para sistemas computacionais, especialmente em aplicações críticas. Um sistema confiável é aquele que executa suas funções corretamente, mesmo em condições adversas ou na presença de falhas. A injeção de falhas, ao simular essas condições, permite identificar vulnerabilidades, contribuindo para o desenvolvimento de sistemas mais robustos e resilientes. Ao testar e avaliar a capacidade de um sistema em lidar com falhas, é possível aprimorar sua arquitetura, seus mecanismos de detecção e recuperação de erros, e, consequentemente, aumentar sua confiabilidade geral (MACIEL, 2022).

Os métodos de injeção de falhas em hardware atuam diretamente nos componentes físicos, como pinos de chip e circuitos internos. Essa abordagem, embora precisa, exige equipamentos especializados, acesso físico ao sistema e pode ser complexa e cara de implementar. Por outro lado, os métodos de software introduzem falhas diretamente no estado do software, como a alteração de valores em variáveis ou a interrupção de fluxos de execução. Essa técnica oferece maior flexibilidade, menor custo e menor intrusão no sistema. Falhas de software podem ser introduzidas de duas maneiras: modificando o código-fonte ou alterando o comportamento do sistema em execução. Isso pode ser feito usando recursos existentes ou desabilitando processos (DUTERTRE; CLÉDIÈRE, 2025).

A mutação de código altera o código-fonte do programa para simular erros de programação. Por exemplo, um operador aritmético pode ser trocado (+ por -) ou uma instrução condicional pode ser invertida. A mutação permite avaliar a robustez do código contra erros lógicos e falhas de implementação, contribuindo para a construção de sistemas mais confiáveis (BARAZA et al., 2008).

2.3 Orthogonal Defect Classification

A Classificação Ortogonal de Defeitos (ODC) representa uma metodologia estruturada para análise de defeitos de software, combinando aspectos quantitativos e qualitativos. Seu objetivo é fornecer um mecanismo eficaz para identificar e corrigir falhas de programação de forma precisa e eficiente.

A metodologia ODC, conforme descrito por (BUTCHER et al., 2002), se organiza em duas categorias: classes abertas e fechadas. Cada classe possui atributos específicos que indicam a gravidade dos defeitos e orientam as ações necessárias para sua representação ou correção.

As classes abertas auxiliam na identificação de defeitos, analisando as atividades em andamento quando a falha foi descoberta, os gatilhos que a manifestam e seu impacto potencial nos usuários. Já as classes fechadas, utilizadas após a correção, focam na natureza exata do defeito e no escopo da correção. Elas categorizam o alvo da correção, o tipo de defeito corrigido, qualificadores para detalhar o defeito, a idade da correção (em qual estágio foi implementada) e a fonte do código (desenvolvimento internamente, reutilizado, terceirizado, etc).

3 Trabalhos Relacionados

A injeção de falhas tem se consolidado como uma técnica fundamental para estudos sobre falhas em ambientes de nuvem. Sua capacidade de introduzir falhas controladas permite avaliar a confiabilidade e os mecanismos de recuperação desses sistemas sob condições adversas. A injeção de falhas também é empregada no desenvolvimento de ferramentas especializadas. Diversas ferramentas foram desenvolvidas para esse propósito, cada uma com características e funcionalidades específicas. A AFEX (BANABIC, 2012) (MEIKLEJOHN, 2024), por exemplo, é uma ferramenta de propósito geral que simula o efeito de testes de caixa preta, injetando falhas em software. Já o DEFINE (KAO, 2012) (DAI, Y.; LIU, S.; LIU, H., 2025), foca em ambientes distribuídos, simulando falhas em hardware e corrompendo informações armazenadas ou transmitidas. A Eucabomber (SOUZA, 2013) (ARAUJO, J. et al., 2023), desenvolvida para plataformas de nuvem Eucalyptus, desativa processos da plataforma, simulando falhas em seu funcionamento.

Outras ferramentas de injeção de falhas em software incluem: a FIAT (SEGALL et al., 1995) (AMYAN, A. et al., 2024), que injeta falhas de forma semelhantes à execução de um trojan; a G-SWIFT (DURAES; MADEIRA, 2006) (SALIH.; GOPAL, 2024) e a PREFAIL (JOSHI 2011) (KHAN; ALI; NAZIR, 2024), ambas alteram o código binário do programa para simular falhas; e a

PROFIPY (COTRONEO, 2020) (COTRONEO; LIGUORI, 2024) oferece uma linguagem de domínio específico (DSL) para programar a injeção de falhas.

Além das ferramentas em software, existem simuladores de falhas em hardware, como o FIM-SIM (YALCIN, 2011) (ASADOVA; KERTESZ; LOVAS, 2023) que utiliza o simulador M5, e o FOCUS (CHOI, 1992) (AMYAN, A. et al., 2024) que utiliza o simulador SPLICE para injetar falhas em designs VLSI. A FERRARI (KANAWATI, 1992) (AMYAN, A. et al., 2024) automatiza o processo de injeção de falhas em arquiteturas SPARC, enquanto a FTAPE (TSAI, 1999) (AMYAN, A. et al., 2024) estressa os recursos de uma máquina alvo em computadores tolerantes a falhas Tandem Integrity S2.

Há também ferramentas que combinam hardware e software para injeção de falhas. A MESSALINE (ARIAT, 1990) (RUANO, O. et al., 2021), por exemplo, injeta falhas em diversos pontos simultaneamente em componentes físicos de plataformas específicas. A RIFLE (MADEIRA, 1994) (SALIH, et al., 2022) utiliza software para injetar falhas em nível de pinos (pin-level faults). Já a Xception (CARREIRA; MADEIRA; SILVA, 2001) (SALIH, et al., 2022) possui um sistema base com formato bem definido, permitindo a injeção de falhas em arquiteturas baseadas em processador, memória e barramentos.

Visto a importância dessa área, esse trabalho apresenta uma ferramenta de injeção de falhas baseada em mutação de código que seja capaz de executar essa tarefa em diversos tipos de ambientes de nuvem.

A Tabela 1 apresenta uma análise comparativa de 15 ferramentas de injeção de falhas em ambientes de nuvem. Dessas ferramentas, 10 utilizam software e 5 utilizam hardware para injetar falhas. Entre as ferramentas baseadas em software, 4 adotam a técnica de mutação de código. Dessas, apenas 2 ferramentas modificam o código fonte, enquanto as outras 2 alteram o código binário. Diferentemente das abordagens de alteração binária (G-SWIFT, PREFAIL) e DSL (PROFIPY), o CIMut atua diretamente no código-fonte via SSH, com recursos de verificação de conteúdo e rastreamento de mutações.

Tabela 1 – Ferramentas de injeção de falhas na nuvem

Nome da Ferramenta	**Tipos de Falhas Injetadas**	**Ambiente/Plataforma**
AFEX (BANABIC, 2012)	Software	Propósito Geral
DEFINE (KAO, 2012)	Software e simula falhas em hardware	Foco em ambientes distribuídos
Eucabomber (SOUZA, 2013)	Software	Plataforma de nuvem Eucalyptus
FIAT (SEGALL et al., 1995)	Software	Propósito Geral
G-SWIFT (DURAES; MADEIRA, 2006)	Software - Alterando código	Propósito Geral
PREFAIL (JOSHI 2011)	Software - Alterando código	Propósito Geral
PROFIPY (CONTRONEO, 2020)	Software - Alterando código	Propósito Geral
FIM-SIM (YALCIN, 2011)	Simulador de falhas de Hardware	Simulador M5

FOCUS (CHOI, 1992)	Simulador de falhas em Hardware	Simulador SPLICE
FERRARI (KANAWATI, 1992)	Software e simula falhas em hardware	Arquitetura SPARC
FTAPE (TSAI, 1999)	Software	Tandem Integrity S2 fault-tolerant computer
MESSALINE (ARIAT, 1990)	Hardware usando componente físicos	Testado em plataformas específicas
RIFLE (MADEIRA, 1994)	Hardware usando software	Propósito Geral
Xception (CARREIRA; MADEIRA; SILVA, 2001)	Hardware usando software	Sistema base com formato bem definido
CIMut (Próprio autor)	Software - Alterando código	Propósito Geral

Fonte: próprio autor

4 Cloud Injection Mutator

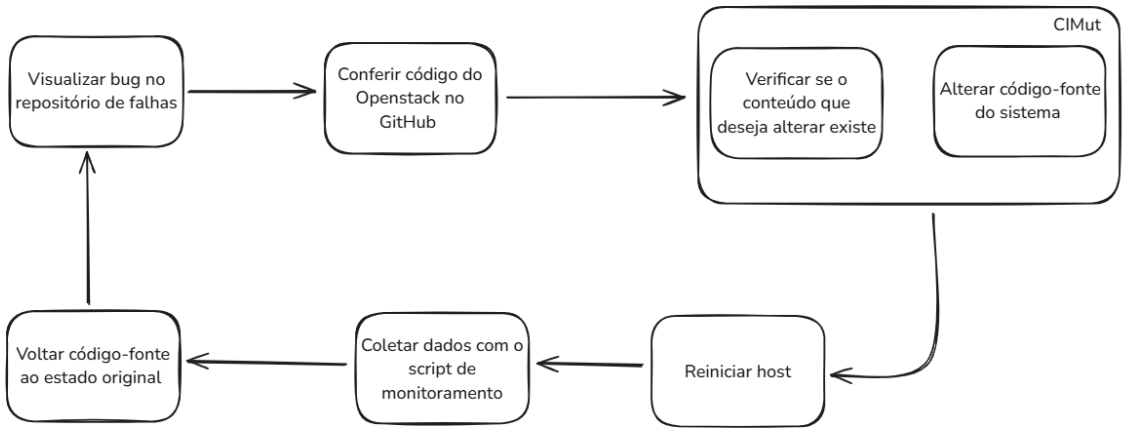
Esta seção apresenta a ferramenta de injeção de falhas proposta que utiliza a técnica de mutação de código para introduzir falhas de software de forma controlada em ambientes de nuvem. A ferramenta desenvolvida foi utilizada para realizar testes em um ambiente de nuvem configurado, envolvendo a injeção de diferentes tipos de falhas e a observação do comportamento do sistema. Esta ferramenta, desenvolvida em Python, auxilia na injeção de falhas em ambientes de nuvem através da mutação de código-fonte. Ela é composta por dois scripts: o script de verificação de conteúdo e o script de alteração de linha. O script de verificação de conteúdo incorpora um mecanismo de verificação de conteúdo, permitindo ao usuário validar a existência do conteúdo a ser alterado no arquivo e linha selecionados.

O script de alteração de linha permite ao usuário modificar o conteúdo de uma linha específica em um arquivo remoto. Para utilizá-lo, é necessário fornecer as credenciais de acesso SSH (endereço IP, porta, nome de usuário e senha), o caminho do arquivo remoto, o número da linha a ser alterada e o novo conteúdo desejado. A ferramenta estabelece uma conexão SSH com o host remoto, baixa o arquivo especificado, realiza a alteração na linha indicada e envia o arquivo modificado de volta para o host. Em seguida, ela registra a mutação em um arquivo de log. Para que as modificações surtam efeito, é necessário reiniciar o host para que o OpenStack recompile o código com o arquivo mutado.

Adicionalmente, um script de coleta de dados foi desenvolvido para alimentar um dashboard criado com Flask. Este dashboard apresenta informações como: módulo referente a cada serviço, nome do serviço, status, consumo de recursos do módulo, nó alvo e consumo médio de cada processo em seu respectivo nó.

A Figura 1 ilustra o fluxo de trabalho da injeção de falhas utilizando o CIMut. Neste exemplo, o CIMut foi utilizado com o OpenStack.

Figura 1 – Fluxo de trabalho para injeção de falhas



Fonte: próprio autor

Esta ferramenta automatiza a análise e modificação do Openstack, facilitando a identificação do impacto de bugs no desempenho do sistema. O fluxo de trabalho se divide em três etapas principais: Verificação, Mutação e Monitoramento.

O processo se inicia com uma inspeção minuciosa de um repositório de falhas (**Visualizar bug no repositório de falhas**). A partir da análise de um bug específico, o componente do OpenStack onde a falha se manifesta é identificado. Posteriormente, o código-fonte do OpenStack, disponível no repositório GitHub, é consultado para determinar o arquivo preciso a ser modificado com o objetivo de induzir a falha (**Conferir código do Openstack no GitHub**). Para auxiliar nesse processo, um script é empregado para verificar a existência do conteúdo a ser alterado na máquina virtual (VM) onde o OpenStack está provisionado. O script recebe como parâmetros o endereço IP da VM, a porta de comunicação, as credenciais de usuário e senha para autenticação, o caminho do arquivo e o número da linha para garantir a precisão da modificação (**CIMut: Verificar se o conteúdo que deseja alterar existe**).

Após a confirmação da existência do conteúdo desejado, um segundo script executa a mutação. Este script recebe o número da linha, o caminho do arquivo e o novo conteúdo que substituirá o original (**CIMut: Alterar código-fonte do sistema**). Após a mutação, a VM é reiniciada para que o Openstack recompile o código com as alterações realizadas (**Reiniciar host**). Em seguida, uma carga de trabalho, com a criação de uma instância, é executada para simular o uso do sistema sob a influência da mutação.

Por fim, um script de monitoramento captura dados dos serviços do OpenStack (**Coletar dados com o script de monitoramento**). As informações coletadas são agrupadas por módulo e apresentadas em um dashboard. Essa visualização facilita a análise do impacto das mutações no desempenho e funcionamento do OpenStack, permitindo uma compreensão mais profunda da relação entre bugs e a dependabilidade do sistema.

5 Estudo de Caso

Esta seção apresenta um estudo de caso com o objetivo de avaliar a dependabilidade do ambiente de nuvem configurado com a plataforma OpenStack, através de experimentos de injeção de falhas com mutação de código através da ferramenta CIMut. Para este estudo, o OpenStack foi configurado utilizando o DevStack (OPENSTACK, 2025), uma ferramenta que permite a implantação do OpenStack em um ambiente de desenvolvimento ou teste. Uma configuração all-in-one foi utilizada, onde uma única máquina virtual executa as funções de nó Controlador, nó de Computação e nó de Rede. Essa abordagem simplifica a implantação e permitiu a criação de um ambiente de teste completo sem a necessidade de uma infraestrutura complexa. A Tabela 2 descreve as especificações da máquina utilizada.

Tabela 2 – Configuração da máquina virtual

	Processador	**Memória**	**Disco**	**Sistema Operacional**	**Módulos OpenStack**
(Compute, Controller, Network) Node	AMD Ryzen 3 4300U	8 GB	1 TB	Ubuntu 22.04.4 LTS (Jammy Jellyfish)	Nova, Neutron, Cinder, Swift, Glance, Keystone

Fonte: próprio autor

No ambiente Devstack, encontramos diversos serviços que representam os componentes do OpenStack. O Nova, responsável pelo gerenciamento de instâncias virtuais, conta com serviços como: devstack@n-api.service, que atua como a API principal, processando solicitações de gerenciamento de instâncias; devstack@n-cpu.service, que executa as instâncias na máquina host; e devstack@n-sch.service, que agenda a execução das instâncias em hosts apropriados. O devstack@n-api-meta.service fornece metadados para as instâncias, enquanto os serviços devstack@n-cond-cell1.service, devstack@n-novnc-cell1.service e devstack@n-super-cond.service são utilizados em implantações com múltiplas células, auxiliando na comunicação e no acesso remoto ao console das instâncias.

O Neutron, responsável pelas redes virtuais, utiliza o serviço devstack@q-svc.service como seu serviço principal. O devstack@q-ovn-metadata-agent.service, quando o plugin OVN é empregado, fornece informações de rede para as instâncias. O Glance, que gerencia imagens de disco, utiliza o serviço devstack@g-api.service para sua API. O Cinder, responsável pelos volumes de disco, possui o devstack@c-sch.service como agendador, o devstack@c-api.service para sua API e o devstack@c-vol.service para gerenciar a criação e exclusão de volumes nos backends de armazenamento. O Placement, que gerencia o inventário de recursos, utiliza o serviço devstack@placement-api.service para sua API.

Além dos serviços específicos de cada componente, o Devstack também utiliza serviços gerais. O devstack@dstat.service coleta e exibe estatísticas de desempenho, o devstack@etcd.service atua como um banco de dados distribuído para armazenar dados de configuração e estado, e o devstack@keystone.service gerencia a autenticação e autorização dentro do OpenStack.

Os experimentos têm como objetivo injetar falhas através da ferramenta CIMut na nuvem privada configurada por meio da mutação do código-fonte, o que envolve a alteração de trechos específicos do código, potencialmente gerando erros que podem resultar em falhas. Neste experimento, as falhas foram injetadas no Nova, o módulo do OpenStack responsável pela computação, que abrange o gerenciamento de recursos e a criação de instâncias. A introdução de erros nos serviços deste módulo impacta diretamente a administração das máquinas virtuais.

Para direcionar a escolha dos arquivos a serem modificados, foram consultados os relatórios de bugs classificados como crash, hang, degradação de desempenho e funcionalidade incorreta reportados no Bugzilla (BUGZILLA, 2024), plataforma dedicada ao reporte de bugs relacionados ao

OpenStack. Após a análise das descrições dos bugs conforme a ODC (BUTCHER et al., 2002), os arquivos de código-fonte relevantes mencionados nos relatórios foram localizados no repositório GitHub do OpenStack. Em seguida, foi realizada uma análise minuciosa do código para identificar as modificações específicas que estavam relacionadas aos reportes dos usuários da plataforma OpenStack. Esse processo sistemático garantiu que as modificações fossem embasadas em problemas reais relatados pela comunidade de usuários e desenvolvedores do OpenStack, aumentando a relevância e a eficácia da injeção de falhas.

Cada cenário foi testado separadamente utilizando o CIMut apresentado na Seção 4 (Figura 1) e cada bug foi replicado duas vezes, totalizando 62 experimentos. Além do script, uma carga de trabalho foi usada baseada na criação/exclusão de máquinas virtuais da menor configuração possível (1 vCPU e 1 GB de RAM) e sem volume de armazenamento. Essas máquinas virtuais foram adotadas apenas para gerar trabalho de alocação de recursos de computação e rede. O recurso de armazenamento foi desconsiderado na instanciação da máquina virtual uma vez que esse recurso é de responsabilidade de outro módulo do OpenStack, o Cinder, que não foi selecionado para teste neste trabalho.

6 Classificação de Bugs

Nos testes, foram considerados apenas bugs que envolvem o módulo Nova, totalizando 31 bugs, distribuídos da seguinte forma: 12 de crash, 3 de hang, 3 de degradação de desempenho e 13 de funcionalidade incorreta. Conforme descrito na Tabela 3.

Tabela 3 – Bugs classificados

Id	Componente	Título	Activity	Trigger	Impacto	Severidade
1906941	openstack-nova	Volume attached at two instances after failover because compute outage	System Test	Software Configuration	Crash	High
1908405	openstack-nova	Nova is out of sync with ironic as hypervisor list in nova does not agree with ironic list after reboot of the nodes	System Test	Software Configuration	Crash	Unspecified
1894829	openstack-nova	[rhosp16] [rt-kvm] Failure in instances build as libvirtd enters uninterruptible state	System Test	Software Configuration	Crash	Urgent
1774332	openstack-nova	resizing from flavor with swap to one without swap puts instance into Error status	System Test	Blocked Test	Crash	High

1851417	openstack-nova	[RFE] [NFV] Support live migration of a VM with an SRIOV PF port	System Test	Software Configuration	Crash	High
1852110	openstack-nova	nova host-evacuation returns erroneous pci addresses and an error: Unable to correlate PCI slot	System Test	Startup/Restart	Crash	High
1899468	openstack-nova	Provide a workaround option and/or nova-manage command to force the refresh of connection_info during a hard reboot	System Test	Startup/Restart	Crash	High
1700390	openstack-nova	KVM-RT guest with 10 vCPUs hangs on reboot	System Test	Startup/Restart	Crash	High
1860808	openstack-nova	[OSP 17][RFE] Allow admins to cold evacuate instances from down compute hosts	Function Test	Test Interaction	Crash	Medium
1860904	openstack-nova	[RFE][OSP 17] Persist disk addresses in the bdms of an instance	Design Review	Backward Compatibility	Crash	Medium
1899581	openstack-nova	MessagingTimeout while in nova.compute.api.API._create_volume_bdm can leave orphaned BDM records in the database	Code Inspection	Concurrency	Crash	Medium
1851490	openstack-nova	Revert migration does not work when destination host is faulty	System Test	Recovery/Exception	Crash	Medium

1679420	openstack-nova	Unable to provision a instance after nova.conf is configured for ssl = true in [oslo_messaging _rabbit] section	System Test	Software Configuration	Hang	High
1774243	openstack-nova	When trying to migrate VMs from a failed compute node, we receive a ResourceProvide rCreationFailed traceback	System Test	Blocked Test	Hang	Urgent
1668318	openstack-nova	[RFE][TestOnly] - For inclusion of ed25519 type key pairs in nova	System Test	Recovery/Ex ception	Hang	Low
1590472	openstack-nova	RFE: QEMU VFIO based block driver for NVMe devices (OSP)	System Test	Blocked Test	Performance Degradation	Unspecified
1834451	openstack-nova	[OSP16.0][nova-compute] error occurred while updating compute node resource provider status to "enabled"	System Test	Startup/Resta rt	Performance Degradation	Low
1856426	openstack-nova	Following an update from 16z2 to 16.1 multiple tempest tests fail with internal server error	Code Inspection	Backward Compatibility	Performance Degradation	High
1569039	openstack-nova	Live Migration fails	System Test	Blocked Test	Incorrect Functionalit y	Unspecified
1907392	openstack-nova	[OSP 17] n-api should reject requests to attach a volume to an instance *if* an active	System Test	Software Configuration	Incorrect Functionalit y	Unspecified

		bdm record exists				
1904525	openstack-nova	Overcloud node Power State is NOSTATE after FFU and nova stop	System Test	Software Configuration	Incorrect Functionality	Medium
1878749	openstack-nova	tempest.api.compute.servers.test_servers_negative DELETE race	Design Review	Design Conformance	Incorrect Functionality	Urgent
1767797	openstack-nova	When unshelving an SR-IOV instance, the binding profile isnt reclaimed or rescheduled, and this might cause PCI-PT conflicts	System Test	Recovery/Exception	Incorrect Functionality	High
1820202	openstack-nova	Live migration of non-NUMA non-pinned instances does not update cpu_shared_set mapping	Code Inspection	Logic/Flow	Incorrect Functionality	High
1614280	openstack-nova	[RFE] Nova should refuse to launch overcommit guests on hosts where cpu_dedicated_set overlaps with isolcpus	Code Inspection	Backward Compatibility	Incorrect Functionality	Medium
1643487	openstack-nova	[RFE] Provide the ability to create ephemeral disk with no filesystem	Function Test	Test Interaction	Incorrect Functionality	Medium
1728298	openstack-nova	nova virtio-scsi drivers turned to virtio-blk after rescue mode	System Test	Hardware Configuration	Incorrect Functionality	Medium
1872314	openstack-nova	[OSP 16.1] Cant launch instance if name ends with a number	Design Review	Design Conformance	Incorrect Functionality	Medium

1877289	openstack-nova	[OSP 17][RFE][rbd] Create a unique rbd user for each host volume attachment	Design Review	Side Effects	Incorrect Functionality	Unspecified
1808416	openstack-nova	Unshelved instance cant migrate and resize	System Test	Blocked Test	Incorrect Functionality	Medium
1852299	openstack-nova	Snapshot cannot be deleted when the instance is shutdown after multiple snapshots creation if cinder nfs backend is used	System Test	Startup/Restart	Incorrect Functionality	Medium

Fonte: próprio autor

A Tabela 3 apresenta uma lista dos bugs utilizados nos experimentos. Cada bug é detalhado em colunas específicas, fornecendo informações como: um identificador único (ID do Bug) para facilitar a referência, o módulo do sistema afetado pelo bug, um título resumindo o problema causado, e uma classificação baseada na metodologia ODC (BUTCHER et al., 2002). Essa classificação, presente nas Colunas 4 a 6, categoriza os defeitos em diferentes aspectos. O termo activity se refere à fase ou estágio do ciclo de vida de desenvolvimento de software (SDLC) onde o defeito foi descoberto, como análise de requisitos, design, codificação, testes (unitário, funcional, de sistema, etc.) ou mesmo implantação e operação. O trigger representa a ação ou condição específica que fez com que o defeito se manifestasse ou se tornasse aparente, como entrada de dados incorreta, comportamento inesperado do usuário, limitações de hardware ou problemas de integração com outros sistemas. Já o impacto descreve o efeito ou consequência do defeito na funcionalidade, desempenho ou experiência do usuário do software, podendo variar de problemas estéticos menores a falhas críticas do sistema, perda de dados ou vulnerabilidades de segurança. A Coluna 7 representa a severidade do bug relatado.

As Tabelas 4 a 7 fornecem um registro detalhado das mutações injetadas, com o objetivo de garantir a rastreabilidade e a capacidade de reproduzir as falhas. Para cada mutação injetada, as tabelas documentam o número do bug no sistema de rastreamento (permitindo vinculá-lo ao problema original), um link para o relatório associado (fornecendo informações adicionais), o caminho do arquivo que sofreu a alteração, os trechos de código originais e modificados (evidenciando as mudanças) e o número da linha específica onde a mutação ocorreu. Quando as colunas 'Conteúdo Original' e 'Conteúdo Alterado' aparentam ser idênticas, a mutação realizada foi a adição de tabs (indentação). Devido à sensibilidade do Python à formatação do código, essa alteração de indentação pode introduzir erros, pois altera a estrutura e o comportamento do programa.

Tabela 4 – Bugs classificados como Crash

Id	Localização do arquivo	Linha Alterada	Conteúdo original	Conteúdo alterado	Link	Identificador
1906941	/opt/stack/nova/nova/compute/mana	4190	self.volume_api.begin_detachi	“ ”	https://bugzilla.redhat.com/s	Failover volume

	ger.py		ng(context, volume_id)		how_bug.cgi?id=1908405	
1908405	/opt/stack/nova/nova/virt/ironic/driver.py	759, 760	self.hash_ring = hash_ring.HashRing(services, partitions=_HASH_RING_PARTITIONS	self.hash_ring = hash_ring.HashRing({next(iter(services))}, partitions=_HASH_RING_PARTITIONS) if services else hash_ring.HashRing(services, partitions=_HASH_RING_PARTITIONS)	https://bugzilla.redhat.com/show_bug.cgi?id=1908405	Post-Reboot Desync
1894829	/opt/stack/nova/nova/compute/manager.py	1635	self.driver.init	“ “	https://bugzilla.redhat.com/show_bug.cgi?id=1894829	Libvirtd Hang (RHOSP16)
1774332	/opt/stack/nova/nova/objects/block_device.py	90	'volume_size': self.volume_size,	volume_size': none,	https://bugzilla.redhat.com/show_bug.cgi?id=1774332	Instance Resize (No-Swap)
1851417	/opt/stack/nova/nova/compute/manager.py	8889	with timeutils.StopWatch() as timer:	for service, info in statuses.items():	https://bugzilla.redhat.com/show_bug.cgi?id=1851417	Live Migration with SR-IOV
1852110	/opt/stack/nova/nova/compute/manager.py	10647	nodenames = set(self.driver.get_available_nodes())	nodenames = '2’	https://bugzilla.redhat.com/show_bug.cgi?id=1852110	PCI Mismatch (Host Evacuation)
1899468	/opt/stack/nova/nova/compute/manager.py	3418	self.driver.power_on(context, instance,	“ ”	https://bugzilla.redhat.com/show_bug.cgi?id=1899468	Nova-Manage Force Refresh
1700390	/opt/stack/nova/nova/virt/libvirt/driver.py	135	from nova.volume import cinder	from nova import cinder	https://bugzilla.redhat.com/show_bug.cgi?id=1700390	Reboot Hang (KVM-RT)
1860808	/opt/stack/nova/nova/compute/manager.py	6629	network_info = self.network_api.add_fixed_ip_to_instance(context,	network_info = self.network_apiadd_fixed_ip_to_inst	https://bugzilla.redhat.com/show_bug.cgi?id=1860808	Cold Evacuation (Down Hosts)

				ance(context,		
1860904	/opt/stack/nova/nova/compute/manager.py	2500	self.compute_task_api.build_instances(context, [instance],	self.compute_task_api.build_instances context, [instance],	https://bugzilla.redhat.com/show_bug.cgi?id=1860904	Disk Address Persistence (OSP17)
1899581	/opt/stack/nova/nova/compute/api.py	84	from nova.scheduler import utils as scheduler_utils	from nova.scheduler import utils	https://bugzilla.redhat.com/show_bug.cgi?id=1899581	Messaging Timeout (BDM Creation)
1851490	/opt/stack/nova/nova/compute/manager.py	246	class InstanceEvents(object):	“ ”	https://bugzilla.redhat.com/show_bug.cgi?id=1851490	Revert Migration Failure (Faulty Host)

Fonte: próprio autor

Tabela 5 – Bugs classificados como Hang

Id	Localização do arquivo	Linha Alterada	Conteúdo original	Conteúdo alterado	Link	Identificador
1679420	/opt/stack/nova/nova/config.py	38	def set_lib_defaults():	“ ”	https://bugzilla.redhat.com/show_bug.cgi?id=1679420	SSL Provisioning Failure
1774243	/opt/stack/nova/nova/compute/resource_tracker.py	142	def instance_claim(self, context, instance, nodename, allocations,	(self, context, instance, nodename, allocations	https://bugzilla.redhat.com/show_bug.cgi?id=1774243	Migration Error (Failed Compute)
1668318	/opt/stack/nova/nova/api/openstack/compute/keypairs.py	70	def create(self, req, body): # noqa	0	https://bugzilla.redhat.com/show_bug.cgi?id=1668318	ed25519 Key Support (Nova)

Fonte: próprio autor

Tabela 6 – Bugs classificados como Degradação de Desempenho

Id	Localização do arquivo	Linha Alterada	Conteúdo original	Conteúdo alterado	Link	Identificador
1590472	/opt/stack/nova/nova/virt/libvirt/driver.py	771	def init_host(self, host):	“ ”	https://bugzilla.redhat.com/show_bug.cgi?id=1590472	RFE: QEMU VFIO (NVMe)

1834451	/opt/stack/nova/nova/compute/manager.py	1059	def _init_instance(self, context, instance):	def _init_instance(self, context instance:	https://bugzilla.redhat.com/show_bug.cgi?id=1834451	nova-compute RP Update Failure
1856426	/opt/stack/nova/nova/compute/manager.py	733	def _get_instances_on_driver(self, context, filters=None):	_get_instances_on_driver(self, context, filters=None):	https://bugzilla.redhat.com/show_bug.cgi?id=1856426	Tempest Failures (16z2 to 16.1)

Fonte: próprio autor

Tabela 7 – Bugs classificados como Funcionalidade Incorreta

Id	Localização do arquivo	Linha Alterada	Conteúdo original	Conteúdo alterado	Link	Identificador
1569039	/opt/stack/nova/nova/virt/libvirt/driver.py	1265	def _check_cpu_set_configuration(self):	def _check_cpu_set_configuration:	https://bugzilla.redhat.com/show_bug.cgi?id=1569039	Live Migration fails
1907392	/opt/stack/nova/nova/api/openstack/compute/volumes.py	36	from nova.volume import cinder	from volume import cinder	https://bugzilla.redhat.com/show_bug.cgi?id=1907392	n-api BDM Validation
1904525	/opt/stack/nova/nova/compute/manager.py	1059	def _init_instance(self, context, instance):	def _init_instance self, context, instance:	https://bugzilla.redhat.com/show_bug.cgi?id=1904525	NOSTATE after FFU & Stop
1878749	/opt/stack/nova/nova/api/openstack/compute/servers.py	93	class ServersController(wsgi.Controller):	let ServersController(wsgi.Controller):	https://bugzilla.redhat.com/show_bug.cgi?id=1878749	API Compute Server DELETE Issue
1767797	/opt/stack/nova/nova/compute/manager.py	396	def exit_wait_early(self, events):	def exit_wait_early(self, events):	https://bugzilla.redhat.com/show_bug.cgi?id=1767797	Unshelve Scheduling Issue (SR-IOV)
1820202	/opt/stack/nova/nova/virt/libvirt/driver.py	379	with self._lock:	with self._lock:	https://bugzilla.redhat.com/show_bug.cgi?id=1820202	Live Migration CPU Mapping Error
1614280	/opt/stack/nova/no	24	class	let	https://bugzil	Overcommit

	va/scheduler/filters/compute_filter.py		ComputeFilter(filters.BaseHostFilter):	ComputeFilter(filters.BaseHostFilter):	la.redhat.com/show_bug.cgi?id=1614280	Launch Conflict (isolcpus)
1643487	/opt/stack/nova/nova/virt/libvirt/driver.py	771	def init_host(self, host):	def init_host, self, host:	https://bugzilla.redhat.com/show_bug.cgi?id=1643487	RFE: Raw Ephemeral Disk
1728298	/opt/stack/nova/nova/virt/libvirt/driver.py	195	def patch_tpool_proxy():	def patch_tpool_proxy():	https://bugzilla.redhat.com/show_bug.cgi?id=1728298	virtio-scsi to virtio-blk (Rescue)
1872314	/opt/stack/nova/nova/virt/libvirt/driver.py	176	VOLUME_DRIVERS = {	DRIVERS = {	https://bugzilla.redhat.com/show_bug.cgi?id=1872314	Trailing Number Launch Block
1877289	/opt/stack/nova/nova/virt/libvirt/driver.py	346	def delete_waiter(self, token: 'AsyncDeviceEventsHandler.Waiter'):	def delete_waiter(self, 'AsyncDeviceEventsHandler.Waiter'):	https://bugzilla.redhat.com/show_bug.cgi?id=1877289	RFE: Unique RBD Users (OSP17)
1808416	/opt/stack/nova/nova/virt/libvirt/driver.py	411	class LibvirtDriver(driver.ComputeDriver)::	class LibvirtDriver(driver.ComputeDriver):	https://bugzilla.redhat.com/show_bug.cgi?id=1808416	Unshelve Restriction (Migration/Resize)
1852299	/opt/stack/nova/nova/virt/libvirt/driver.py	2036	self, connection_info: ty.Dict[str, ty.Any]	self, ty.Dict[str, ty.Any]	https://bugzilla.redhat.com/show_bug.cgi?id=1852299	Snapshot Deletion Error (Cinder NFS)

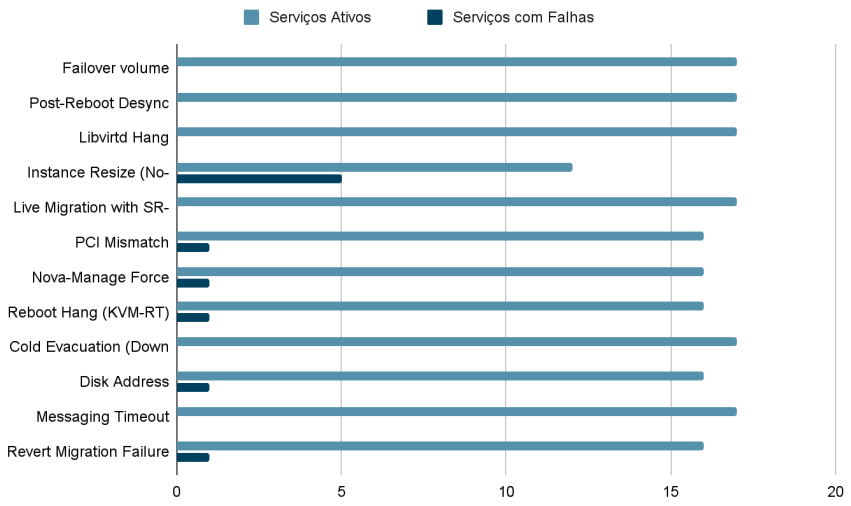
Fonte: próprio autor

A Figura 2 apresenta a relação entre bugs classificados como crash e o status dos serviços. Os seguintes bugs como Failover volume, Post-Reboot Desync, Libvirtd Hang (RHOSP16), Live Migration with SR-IOV, Cold Evacuation (Down Hosts) e Messaging Timeout (BDM Creation), não impactaram o status dos serviços obtendo um resultado de 17 serviços ativos. No entanto, esses bugs causaram problemas funcionais durante a criação de instâncias. Essa discrepância pode ser explicada pela natureza lógica dos erros, que comprometem a funcionalidade sem alterar o status do serviço.

Em contrapartida, outros bugs causaram mudanças no status dos serviços. No caso do Instance Resize (No-Swap), os serviços devstack@n-cpu.service, devstack@n-sch.service, devstack@n-cond-cell1.service, devstack@n-novnc-cell1.service e devstack@n-super-cond.service ficaram inativos, enquanto apenas devstack@n-api.service e devstack@n-api-meta.service permaneceram ativos. Isso representa que 71,47% dos serviços do módulo Nova tiveram degradação, resultando em status de falha. Em relação a todos os módulos, 12 serviços permaneceram ativos enquanto 5 tiveram status de falha.

Os demais bugs alteraram o status apenas do serviço devstack@n-cpu.service, resultando em 16 serviços ativos e 1 com falha. Essa observação sugere que o impacto nos serviços pode variar de acordo com a natureza específica do bug.

Figura 2 – Resultados do experimento com bugs do tipo crash



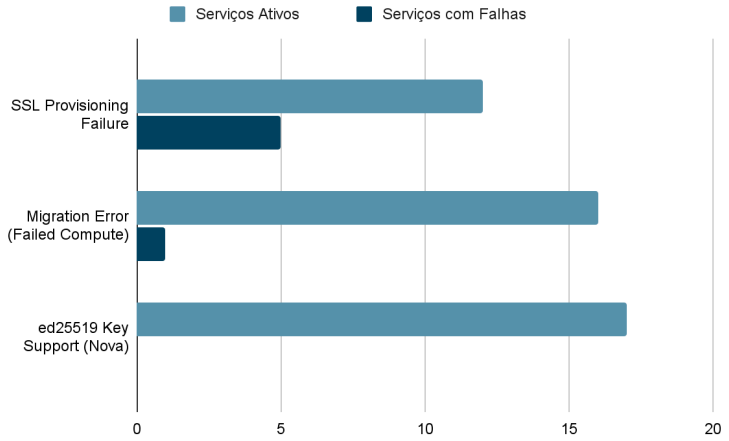
Fonte: próprio autor

A Figura 3 analisa a relação entre bugs classificados como hang e o status dos serviços. o bug ed25519 Key Support (Nova) não impactou o status dos serviços. O sistema manteve 17 serviços ativos. No entanto, esse bug causou falhas na criação de instâncias.

Em contraste, outros bugs causaram mudanças no status dos serviços. No caso do SSL Provisioning Failure (No-Swap), teve como resultado 12 serviços ativos e 5 com falhas. Uma parcela significativa dos serviços do módulo Nova foi afetada, como esses: devstack@n-cpu.service, devstack@n-sch.service, devstack@n-cond-cell1.service, devstack@n-novnc-cell1.service e devstack@n-super-cond.service ficaram inativos, enquanto apenas os serviços devstack@n-api.service e devstack@n-api-meta.service permaneceram ativos. Isso representa 71,47% dos serviços do módulo Nova.

O bug Migration Error (Failed Compute) provocou a ocorrência de falha no serviço devstack@n-cpu.service. Apesar disso, o sistema manteve 16 serviços ativos.

Figura 3 – Resultados do experimento com bugs do tipo hang



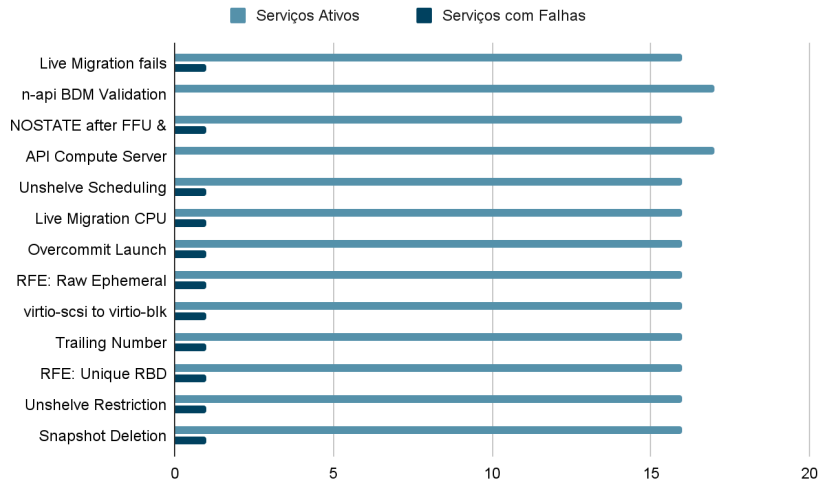
Fonte: próprio autor

A Figura 4 analisa a relação entre bugs classificados como funcionalidade incorreta e o status dos serviços no OpenStack. Uma observação importante é a propagação de erros causada por dois bugs específicos: Live Migration fails e Trailing Number Launch Block. Embora injetados no módulo Nova, esses bugs impactaram o módulo Cinder, impedindo a criação de volumes. Ambos alteraram o status do serviço devstack@c-vol.service para falha. O bug Trailing Number Launch Block apresentou um comportamento mais crítico: a aba de volumes desapareceu do ambiente de nuvem e a reversão do código para a versão original não restaurou o módulo Cinder. Para continuar os testes, foi necessário provisionar um novo ambiente.

A análise da Figura 4 também revela diferentes padrões de impacto nos serviços. O bug Overcommit Launch Conflict (isolcpus) causou um erro de status falha no serviço devstack@n-sch.service. Um grupo de bugs, incluindo NOSTATE after FFU & Stop, Unshelve Scheduling Issue (SR-IOV), Live Migration CPU Mapping Error, RFE: Raw Ephemeral Disk, virtio-scsi to virtio-blk (Rescue), RFE: Unique RBD Users (OSP17), Unshelve Restriction (Migration/Resize) e Snapshot Deletion Error (Cinder NFS), todos deixaram o serviço devstack@n-cpu.service com o status de falha.

Por fim, os bugs n-api BDM Validation e API Compute Server DELETE Issue se destacaram por não alterarem o status dos serviços. No entanto, esses bugs causaram erros na criação de instâncias, indicando problemas funcionais no OpenStack.

Figura 4 – Resultados do experimento com bugs do tipo Funcionalidade Incorreta

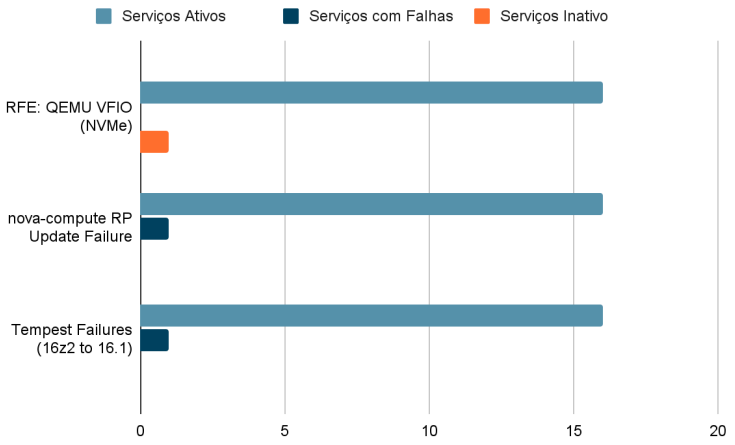


Fonte: próprio autor

A Figura 5 analisa a distribuição de serviços ativos, com falhas e inativos para bugs classificados como Degradação de Performance. O bug RFE: QEMU VFIO (NVMe) se destaca por apresentar um status inativo no serviço devstack@n-cpu.service. Esse status indica que o serviço não foi iniciado devido ao bug, representando uma interrupção completa da funcionalidade.

Em contraste, os bugs nova-compute RP Update Failure e Tempest Failures (16z2 to 16.1) apresentaram um status falha no serviço devstack@n-cpu.service. O status falha sugere que o serviço foi iniciado, mas encontrou erros durante sua execução, resultando em uma degradação do desempenho.

Figura 5 – Resultados do experimento com bugs do tipo Degradação de Desempenho

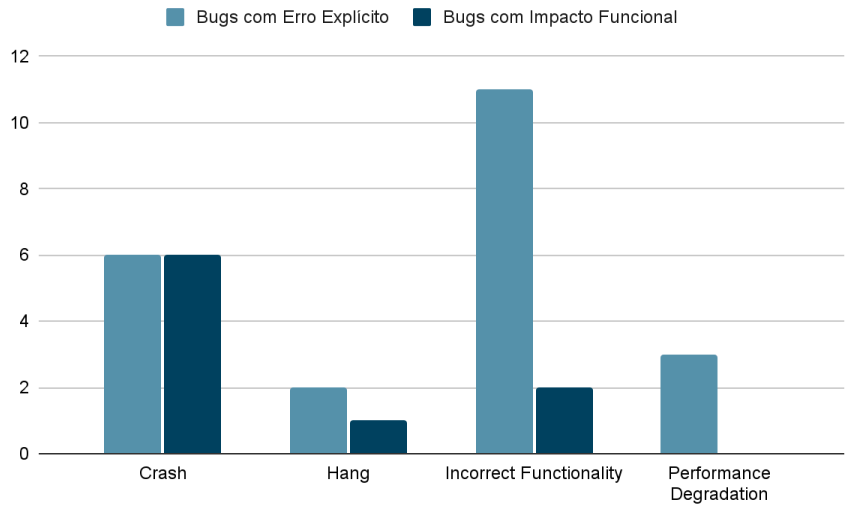


Fonte: próprio autor

A Figura 6 apresenta um agrupamento de todos os bugs analisados. Dentre eles, apenas um bug não resultou em falhas após a injeção da mutação no código. Os resultados obtidos podem ser classificados em duas categorias: bugs com erros explícitos, que causaram erros evidentes refletidos

no status dos serviços do módulo Nova, e bugs com impacto funcional, que, apesar de não apresentarem falhas explícitas no status dos serviços, impediram o uso de funcionalidades específicas da nuvem, como a criação de máquinas virtuais.

Figura 6 – Resultados dos experimentos classificação por bugs



Fonte: próprio autor

Dentre os tipos de falha, Crash apresentou uma distribuição equilibrada (50%) entre bugs com impacto funcional e erros explícitos. Hang exibiu maior incidência de erros explícitos (66%) em relação aos bugs de impacto funcional (34%). Funcionalidade Incorreta apresentou predominância de erros explícitos (84,62%) em comparação aos bugs de impacto funcional (15,38%). Degradação de Performance teve 100% das falhas classificadas como erros explícitos.

A análise estratificada por módulo e status (ativo, falha, inativo) visa compreender o comportamento do sistema em cenários de falha.

7 Conclusão/Considerações finais

Este trabalho apresentou o CIMut uma ferramenta desenvolvida para injeção de falhas em ambientes de nuvem, utilizando técnicas de mutação. Um estudo de caso baseado no módulo Nova da plataforma Openstack foi apresentado para avaliação da ferramenta. Através da ferramenta proposta, foram injetados 31 bugs, distribuídos da seguinte forma: 12 crash, 3 hang, 3 de degradação de desempenho e 13 de funcionalidade incorreta.

A análise dos bugs revelou impactos variados nos serviços do OpenStack. Enquanto alguns não alteraram o status dos serviços, outros resultaram em erros com status de falha ou inativo, indicando interrupções parciais ou completas. O impacto variou de acordo com o bug específico, com alguns afetando apenas um serviço e outros impactando vários simultaneamente.

É importante destacar que, mesmo nos casos em que o status dos serviços permaneceu inalterado, alguns bugs causaram problemas funcionais durante operações como a criação de instâncias. Essa discrepância entre o status do serviço e o impacto funcional, já apontada em trabalhos anteriores como NATELLA et al. (2013), demonstra a necessidade de novos métodos de monitoramento.

Os resultados obtidos a partir desses experimentos demonstram a eficácia do CIMut para injeção de falhas e coleta de dados em ambientes OpenStack. Para aprimorar a ferramenta, os trabalhos futuros se concentrarão no desenvolvimento de mecanismos de decisão. Esses mecanismos permitirão a injeção de falhas em locais mais estratégicos e relevantes para a análise do ambiente de

nuvem. Paralelamente, serão conduzidos testes para avaliar a resiliência da nuvem em infraestruturas distribuídas, ampliando o escopo da análise de falhas.

Financiamento

Esta pesquisa não recebeu financiamento

Conflito de interesses

Os autores declaram não haver conflito de interesses.

Referências

ABDERRAHIM, W.; CHOUKAIR, Z. PaaS dependability integration architecture based on cloud brokering. In: PROCEEDINGS OF THE 31ST ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, Pisa, Italy, 2016. New York: ACM, 2016.

ARLAT, J. et al. Fault injection for dependability validation: a methodology and some applications. IEEE Transactions on Software Engineering, v. 16, n. 2, p. 166–182, 1 fev. 1990.

ABED, Amira Hassan. The Techniques of authentication in the Context of Cloud Computing. Int. J. Advanced Networking and Applications, v. 16, n. 04, p. 6515-6522, 2025.

ARAUJO, J. et al. Availability and reliability modeling of mobile cloud architectures. IEEE Transactions on Industrial Informatics, 2023.

AMYAN, A. et al. Automating Fault Test Cases Generation and Execution for Automotive Safety Validation via NLP and HIL Simulation. Sensors, v. 24, n. 10, p. 3145, 2024.

ASADOVA, F.; KERTESZ, G.; LOVAS, R. Simulation-based Network Fault Injection in the CloudSim Plus Cloud Simulation Environment. Azerbaijan Journal of High Performance Computing, v. 6, n. 1, 2023.

BUGZILLA. Red Hat Bugzilla. Disponível em: https://bugzilla.redhat.com/. Acesso em: 12 fev. 2024.

BARAZA, J. -C.; GRACIA, J.; BLANC, S.; GIL, D.; GIL, P. -J. Enhancement of Fault Injection Techniques Based on the Modification of VHDL Code. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v. 16, n. 6, p. 693-706, June 2008. DOI: 10.1109/TVLSI.2008.2000254.

BUTCHER, M.; MUNRO, H.; KRATSCHMER, T. Improving software testing via ODC: Three case studies. IBM Systems Journal, v. 41, n. 1, p. 31-44, 2002. DOI: 10.1147/sj.411.0031.

BANABIC, R.; CANDEA, G. Fast black-box testing of system recovery code. Proceedings of the 7th ACM european conference on Computer Systems - EuroSys ’12, 2012.

CHOI, G. S.; IYER, R. K. FOCUS: an experimental environment for fault sensitivity analysis. IEEE Transactions on Computers, v. 41, n. 12, p. 1515-1526, Dec. 1992. DOI: 10.1109/12.214660.

CROUZET, Y.; KANOUN, K. System Dependability: Characterization and Benchmarking. Amsterdam: Elsevier, 2012. Disponível em: https://hal.science/hal-00761042/document. Acesso em: 05 jan. 2025.

CORREIA, F. Definição de computação em nuvem segundo o NIST. Plataforma Nuvem, 2011. Disponível em: https://plataformanuvem.wordpress.com/2011/11/21/definicao-de-computacao-em-nuvem-segundo-o-nist/. Acesso em: 05 jun. 2024.

COTRONEO, D.; DE SIMONE, L.; LIGUORI, P.; NATELLA, R. ProFIPy: programmable software fault injection as-a-service. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Valencia, Spain, 2020. Valencia: IEEE, 2020. p. 364-372. DOI: 10.1109/DSN48063.2020.00052.

COTRONEO, D.; LIGUORI, P. Neural Fault Injection: Generating Software Faults from Natural Language. arXiv preprint arXiv:2404.07491, 2024.

CARREIRA, J.; MADEIRA, H.; SILVA, J. Xception: software fault injection and monitoring in processor functional units. In: Proceedings of the 5th IFIP Working Conference on Dependable Computing for Critical Applications, 2001.

DURAES, J. A.; MADEIRA, H. S. Emulation of software faults: a field data study and a practical approach. IEEE Transactions on Software Engineering, v. 32, n. 11, p. 849-867, Nov. 2006. DOI: 10.1109/TSE.2006.113.

DIVYA, C. D.; HARSHITHA, K. Tools and Open Source Platforms for Cloud Computing. In: Advanced Computing Techniques for Optimization in Cloud. Chapman and Hall/CRC, 2025. p. 226-244.

DAI, Y.; LIU, S.; LIU, H. Mutation-Based Approach to Supporting Human–Machine Pair Inspection. Electronics, v. 14, n. 2, p. 382, 2025.

DUTERTRE, Jean-Max; CLÉDIÈRE, Jessy. An Introduction to Fault Injection Attacks. Embedded Cryptography 1, p. 215, 2025.

CHOWDHURY, D. D. Cloud Networking. In: Future of Networks: Modern Communication Infrastructure. Cham: Springer Nature Switzerland, 2025. p. 79-123.

GIL, D.; BARAZA-CALVO, J. C.; GRACIA, J.; GIL-VICENTE, P. VHDL simulation-based fault injection techniques. In: GIL, D. (Ed.). Dependable Computing Systems: Paradigms, Performance Issues, & Applications. Hoboken, NJ: John Wiley & Sons, Inc., 2004. p. 159-176. DOI: 10.1007/0-306-48711-X_10.

J. Arlat, J. . -C. Fabre and M. Rodriguez, "Dependability of COTS microkernel-based systems," in

IEEE Transactions on Computers, vol. 51, no. 2, pp. 138-163, Feb. 2002, doi: 10.1109/12.980005.

JENN, E. et al. Fault injection into VHDL models: the MEFISTO tool. In: Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing, Austin, TX, USA, 1994. p. 66-75. DOI: 10.1109/FTCS.1994.315656.

JOSHI, P.; GUNAWI, H. S.; SEN, K. PREFAIL. SIGPLAN Notices, v. 46, n. 10, p. 171–188, 18 out. 2011.

KAO, W.; IYER, R. K. DEFINE: a distributed fault injection and monitoring environment. In: Proceedings of IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, College Station, TX, USA, 1994. p. 252-259. DOI: 10.1109/FTPDS.1994.494497.

KANAWATI, G. A.; KANAWATI, N. A.; ABRAHAM, J. A. FERRARI: a tool for validating system dependability properties. In: [1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing, Boston, MA, USA, 1992. Proceedings [...]. Boston: IEEE, 1992. p. 336-344. DOI: 10.1109/FTCS.1992.243567.

LI, Y. et al. Predicting Node Failures in an Ultra-Large-Scale Cloud Computing Platform: An AIOps Solution. ACM Transactions on Software Engineering and Methodology, v. 29, n. 2, 2020.

MELL, P. The NIST Definition of Cloud Computing. 2011. Thesis (Doctorate in Information Technology) – Department of Information Technology, University of Maryland, College Park, 2011. Disponível em: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf. Acesso em: 25 Oct. 2023.

MADEIRA, H.; RELA, M.; MOREIRA, F.; SILVA, J. G. RIFLE: a general purpose pin-level fault injector. In: ECHTLE, K.; HAMMER, D.; POWELL, D. (Ed.). Dependable Computing — EDCC-1. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994. p. 197-216. (Lecture Notes in Computer Science, v. 852). DOI: 10.1007/3-540-58426-9_132.

MACIEL, P. Performance, Reliability, and Availability Evaluation of Computational Systems. Volume I: Performance and Background. Boca Raton: CRC Press, 2022. 10.1201/9781003306016.

MEIKLEJOHN, C. Resilient Microservice Applications, by Design, and without the Chaos. 2024. Tese de Doutorado. Carnegie Mellon University.

KHAN, H. U.; ALI, F.; NAZIR, S. Systematic analysis of software development in cloud computing perceptions. Journal of Software: Evolution and Process, v. 36, n. 2, p. e2485, 2024.

OPENSTACK. Open Source Cloud Computing Platform Software - OpenStack. Disponível em: https://www.openstack.org/software/project-navigator/openstack-components#openstack-services. Acesso em: 12 fev. 2025.

HAT, Red. OpenShift. Disponível em:: https://www. openshift. com/learn/what-isopenshift. Acesso

em: 19 fev. 2025.

RUANO, O. et al. Fault injection emulation for systems in FPGAs: Tools, techniques and methodology, a tutorial. Sensors, v. 21, n. 4, p. 1392, 2021.

R. NATELLA, D. CONTRONEO, J. A. DURAES, and H. S. MADEIRA, "On Fault Representativeness of Software Fault Injection," in IEEE Transactions on Software Engineering, vol. 39, no. 1, pp. 80-96, Jan. 2013, doi: 10.1109/TSE.2011.124.

SOUZA, D.; MATOS, R.; ARAUJO, J.; ALVES, V.; MACIEL, P. EucaBomber: experimental evaluation of availability in Eucalyptus private clouds. In: 2013 IEEE International Conference on Systems, Man, and Cybernetics, Manchester, UK, 2013. Manchester: IEEE, 2013. p. 4080-4085. DOI: 10.1109/SMC.2013.696.

SEGALL, Z.; VRSALOVIC, D.; SIEWIOREK, D.; YASKIN, D.; KOWNACKI, J.; BARTON, J. FIAT - Fault injection based automated testing environment. In: Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, 'Highlights from Twenty-Five Years'., Pasadena, CA, USA, 1995. Pasadena: IEEE, 1995. p. 394-. DOI: 10.1109/FTCSH.1995.532663.

SOLÍS, M. et al. Enhancing Plantation Forest Sustainability: A Review of Eucalyptus Defence Mechanisms to Foliar Fungal Pathogens. Current Forestry Reports, v. 11, n. 1, p. 13, 2025.

SALIH, N. K.; GOPAL, R. Optimization Software Fault Injection by Using Two-Dimensional Bin Packing Algorithms. J. Electrical Systems, v. 20, n. 10s, p. 6638-6645, 2024.

SALIH, N. K. et al. A survey on software/hardware fault injection tools and techniques. In: 2022 IEEE Symposium on Industrial Electronics & Applications (ISIEA). IEEE, 2022. p. 1-7.

TSAI, T.; IYER, R. K. Measuring fault tolerance with the FTAPE fault injection tool. In: DOULIERI, J. et al. (Ed.). Dependable Computing – EDCC-3. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. p. 26-43. (Lecture Notes in Computer Science, v. 1667). DOI: 10.1007/BFb0024305.

YALCIN, G.; UNSAL, O. S.; CRISTAL, A.; VALERO, M. FIMSIM: a fault injection infrastructure for microarchitectural simulators. In: 2011 IEEE 29th International Conference on Computer Design (ICCD), Amherst, MA, USA, 2011. Amherst: IEEE, 2011. p. 431-432. DOI: 10.1109/ICCD.2011.6081435.

VANKAYALAPATI, R. K. The future of hybrid cloud: Trends, technologies, and predictions. The Synergy Between Public and Private Clouds in Hybrid Infrastructure Models: Real-World Case Studies and Best Practices, p. 148, 2025.