Chaos-K8s: Avaliação Sistemática de Disponibilidade em Clusters Kubernetes

Jonas Nunes[1], Iure Fé[1], Lucas Silva Lopes[1], José Miqueias[1], Elias P. Duarte Jr.[2] e Francisco Airton Silva[1]

[1]Universidade Federal do Piauí – PI – Brasil

[2]Universidade Federal do Paraná – PR – Brasil

{jonas.nunes,iure.fe,lucaslopes092020,jmiqueias,faps}@ufpi.edu.br

elias@inf.ufpr.br

Resumo. *A dependência crescente de serviços digitais críticos tornou a disponibilidade um requisito estratégico, porém a variedade de configurações possíveis em clusters Kubernetes dificulta antecipar o comportamento do sistema frente a falhas. Apesar dos avanços em técnicas de experimentação controlada, ainda é pouco explorado como relacionar configurações do cluster a métricas de disponibilidade percebidas pelos usuários. Este trabalho propõe o Chaos-K8s, uma solução que orquestra campanhas de testes de falhas em componentes (plano de controle, nós e pods), realizando descoberta automática da infraestrutura, calibração de MTTR por componente e medição de tempo de indisponibilidade, com o objetivo de permitir que administradores quantifiquem o impacto de diferentes configurações sobre a disponibilidade percebida pelos usuários finais. O estudo de caso demonstrou a eficácia do Chaos-K8s na avaliação de configurações, resultando em estimativas de disponibilidade validadas por modelagem em Rede de Petri Estocástica, com 95% de confiança.*

1. Introdução

Nos últimos anos, a dependência de serviços digitais em setores críticos como comércio eletrônico, serviços financeiros, governo eletrônico e aplicações de saúde tornou a disponibilidade um requisito tão importante quanto desempenho e segurança. Quedas recentes de grandes plataformas de nuvem e de serviços amplamente utilizados como a Amazon Web Services (AWS) afetaram cerca de 500 empresas por conta da interrupção, algumas delas no Brasil foram iFood, Mercado Livre, Itaú e entre outros [G1 2025]. Mesmo que de curta duração, as quedas geraram indisponibilidade global e impactos milionários, evidenciando como poucos minutos de interrupção podem comprometer cadeias de negócio inteiras e a confiança de usuários [Thakare 2025]. Nesse cenário, métricas de disponibilidade e de tempo de recuperação deixaram de ser apenas indicadores operacionais para se tornarem insumos estratégicos no planejamento de capacidade e na definição de acordos de nível de serviço (SLAs) [Nguyen et al. 2021].

No contexto de computação em nuvem, o Kubernetes é amplamente utilizado para orquestração de contêineres [Barletta et al. 2024]. Sua ampla adoção decorre da capacidade de escalar aplicações baseadas em microsserviços, automatizar implantação e facilitar a observabilidade de sistemas complexos. Por outro lado, a própria flexibilidade da plataforma com múltiplos componentes de *control plane*, serviços de rede, camadas de armazenamento e diferentes políticas de agendamento e reinício de *pods* introduz

uma grande variedade de configurações possíveis. Neste contexto, o *control plane* reúne os serviços responsáveis por manter o estado global do cluster e decidir onde as cargas serão executadas (por exemplo, `etcd`, `kube-apiserver`, `kube-scheduler` e `kube-controller-manager`). Os nós *workers* correspondem às máquinas que de fato executam os contêineres de usuário e os *pods* são a menor unidade de implantação do Kubernetes, agrupando um ou mais contêineres que compartilham rede e armazenamento. Isso torna desafiadora a tarefa de antecipar como o sistema irá se comportar em face de falhas em cada camada.

Embora técnicas de teste e de experimentação controlada de falhas para avaliar a robustez de aplicações distribuídas tenham avançado significativamente, ainda persiste uma lacuna na compreensão de como diferentes configurações de um cluster Kubernetes impactam a disponibilidade percebida pelos usuários finais [Vayghan et al. 2019]. Equipes de desenvolvimento e operação frequentemente dispõem de ferramentas focadas em monitoramento ou em campanhas de falhas pontuais [Sebastio et al. 2021]. Contudo, têm dificuldade em relacionar, de forma sistemática, parâmetros de configuração do cluster com métricas quantitativas de disponibilidade [Hecht and Agena 2024]. Essas métricas incluem tempo de indisponibilidade de *pods*, perda de requisições e tempo de recuperação após eventos adversos [Nguyen et al. 2020].

Este trabalho propõe o Chaos-K8s[1], uma solução integrada para avaliação sistemática de disponibilidade em clusters Kubernetes, com o objetivo de permitir que administradores quantifiquem o impacto de diferentes configurações sobre a disponibilidade percebida pelos usuários finais. O Chaos-K8s contribui com: (i) descoberta automática de infraestrutura, que mapeia componentes do *control plane* incluindo etcd, kube-apiserver, kube-controller-manager, kube-scheduler, nós *workers* e implantações de aplicação para gerar configurações iniciais sem esforço manual; (ii) caracterização automática de tempos de recuperação (MTTR), permitindo que administradores entendam quanto tempo leva para cada serviço se recuperar de falhas, complementando ou substituindo valores padrão; (iii) orquestração unificada de testes de falha em componentes, suportando falhas no plano de controle, nós e pods, com mecanismos especializados que operam em baixo nível na infraestrutura do Kubernetes; e (iv) medição automática de disponibilidade, que calcula a disponibilidade do sistema por meio de execuções controladas e monitoramento, capturando métricas como tempo de indisponibilidade de *pods*, janelas de recuperação e impacto sobre as cargas de trabalho.

A solução oferece configuração flexível, tanto automática (com valores preenchidos automaticamente para os componentes identificados na infraestrutura extraída, a partir de valores padrão) quanto manual (permitindo ajustes granulares de *Mean Time To Failure* (MTTF), *Mean Time To Recovery* (MTTR) e parâmetros do teste de falha), facilitando a experimentação iterativa. Além disso, o Chaos-K8s foi validado por meio de modelagem baseada em Rede de Petri Estocástica (SPN), um formalismo amplamente utilizado para análise quantitativa de sistemas com comportamento estocástico [Silva et al. 2015], permitindo comparar os resultados empíricos com valores analíticos derivados do modelo matemático [Carvalho et al. 2020]. Como resultado, a ferramenta apoia a comparação de estratégias de configuração e a avaliação de arquiteturas resilientes com base em evidências quantitativas de disponibilidade.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta e dis-

[1]https://gitlab.com/pasid3/chaos-k8s

cute os principais trabalhos relacionados, destacando suas contribuições e limitações em comparação com a abordagem proposta. A Seção 3 detalha a arquitetura do Chaos-K8s e descreve seus componentes e fluxo de operação. A Seção 4 apresenta a validação por modelagem SPN. A Seção 5 apresenta o estudo de caso, incluindo a configuração do ambiente e os resultados obtidos. Por fim, a Seção 6 discute as considerações finais e as perspectivas futuras derivadas deste trabalho.

2. Trabalhos Relacionados

Esta seção apresenta uma revisão dos principais trabalhos relacionados à avaliação de resiliência e disponibilidade em clusters Kubernetes, organizados em três categorias: plataformas de teste e avaliação de disponibilidade, tolerância a falhas em Kubernetes e microsserviços, e avaliação de resiliência em ambientes híbridos (ou seja, ambientes que funcionam tanto localmente quanto na nuvem). A Tabela 1 sintetiza a comparação entre os oito trabalhos analisados, considerando critérios como medição de *downtime*, suporte a infraestrutura, mecanismos de *self-healing*, parâmetros MTTR/MTTF e foco em serviços.

Tabela 1. Comparação entre trabalhos relacionados

Trabalho	Medição de Downtime	Infraestrutura (CP/Nós)	Suporte Self-Healing	Parâmetros MTTR / MTTF	Foco Serviços
[Gortázar et al. 2017]	–	–	–	–	–
[Vayghan et al. 2019]	✓	✓	✓	–	–
[Ikeuchi et al. 2020]	✓	–	✓	–	–
[Flora et al. 2022]	–	–	✓	–	–
[Baptista et al. 2023]	–	–	✓	–	✓
[Chen et al. 2025]	✓	✓	✓	✓	✓
[Aderaldo and Mendonça 2025]	✓	–	✓	–	–
Este Trabalho	✓	✓	✓	✓	✓

Plataformas de teste e avaliação de falhas. O ElasTest [Gortázar et al. 2017] oferece uma plataforma unificada e de código aberto para testes de ponta a ponta em aplicações distribuídas em ambientes de nuvem. A plataforma gerencia o ciclo completo de testes (implantação, execução, monitoramento) e integra um *Instrumentation Manager* capaz de executar testes de falha como perda de pacotes, sobrecarga de CPU e falhas de nó, facilitando a avaliação de elasticidade e tolerância a falhas em arquiteturas de microsserviços. O Defektor [Baptista et al. 2023] atua como meta-ferramenta extensível que orquestra campanhas de teste de falha via sistema de *plugins* para diferentes injetores e orquestradores, automatizando o ciclo completo de planejamento da campanha, geração de carga, execução do teste e coleta de dados.

Tolerância a falhas em Kubernetes e microsserviços. Em [Vayghan et al. 2019], os autores investigam o Kubernetes como gerenciador de disponibilidade para aplicações baseadas em microsserviços, avaliando se seus mecanismos nativos de recuperação (*self-healing*) são suficientes para garantir níveis de confiabilidade exigidos por provedores *carrier-grade*. Os resultados demonstram que a configuração padrão apresenta tempo de recuperação proibitivo para falhas de nó ($\approx$5 min), mas ajustes nos parâmetros de monitoramento e adição de redundância permitem alcançar tempos comparáveis ao *middleware* de alta disponibilidade OpenSAF. O estudo apresentado em [Flora et al. 2022] investiga envelhecimento de *software* e tolerância a falhas de microsserviços em Kubernetes, evidenciando que as sondas nativas (*liveness* e *readiness probes*) não detectam eficazmente problemas de envelhecimento

nem identificam falhas funcionais injetadas em arquiteturas como TeaStore e Sockshop. Em [Ikeuchi et al. 2020], é proposto um *framework* que combina *Deep Reinforcement Learning* com técnicas de experimentação controlada de falhas para aprendizado autônomo de políticas de recuperação de falhas, onde o agente aprende a política ideal ao experimentar ações de recuperação e observar métricas em ambientes baseados em contêineres, dispensando histórico de recuperação ou modelagem prévia do sistema.

Avaliação de resiliência em ambientes híbridos. Em [Chen et al. 2025], é apresentado um *framework* de avaliação de resiliência para ambientes híbridos *cloud-edge*, integrando múltiplos injetores (Chaos Mesh, Gremlin, ChaosBlade) e gerador de carga (Locust) para automatizar falhas em nível de nó, *pod* e rede. As principais contribuições incluem um *dataset* público de mais de 30 GB e análise quantitativa demonstrando maior estabilidade na borda sob atrasos de rede, enquanto a nuvem se mostra superior sob limitação de banda. O ResilienceBench-Operator [Aderaldo and Mendonça 2025] apresenta um operador nativo de Kubernetes que define espaços de teste via CRDs, aplicando falhas com Envoy e executando carga com K6 [Grafana Labs 2026] para avaliar padrões de resiliência como *Retry* e *Circuit Breaker* em topologias reais de microsserviços.

Conforme observado na Tabela 1, a maioria dos trabalhos foca em falhas no nível de aplicação (`pods` e serviços) ou utiliza injetores de terceiros para conduzir as campanhas de testes. Este trabalho diferencia-se por: (i) atuar diretamente sobre componentes internos do Kubernetes (`etcd`, `kube-apiserver`, `containerd`, `kubelet`), e não apenas sobre aplicações de usuário; (ii) oferecer a extração da infraestrutura do cluster e calibração empírica de MTTR por componente; (iii) suportar implantação simplificada tanto em ambientes de nuvem (AWS) quanto em clusters locais (Kind, Minikube); e (iv) integrar validação por modelagem SPN, ampliando o foco tradicional de experimentação controlada de falhas para avaliação quantitativa de disponibilidade e permitindo comparação com resultados analíticos. Assim, a contribuição não é substituir plataformas generalistas de caos (por exemplo, Chaos Mesh e Gremlin), mas oferecer um fluxo integrado de calibração e avaliação quantitativa de disponibilidade para Kubernetes.

3. Arquitetura

Esta seção descreve a arquitetura e o fluxo de operação da solução proposta (Chaos-K8s), destacando como o administrador extrai a composição da infraestrutura, configura parâmetros (por exemplo, MTTF/MTTR) e executa testes de disponibilidade sob cenários de falha.

O Chaos-K8s é uma solução de teste de falhas projetada para avaliar sistematicamente a disponibilidade de clusters Kubernetes. A solução atua como um orquestrador externo que interage com o cluster Kubernetes, realizando testes controlados em diferentes componentes da infraestrutura (*control plane*, nós *workers* e *pods*) e monitorando o comportamento do sistema durante a recuperação. O objetivo principal é permitir que administradores quantifiquem o impacto de diferentes configurações sobre a disponibilidade observada pelos usuários finais.

3.1. Fluxo de Preparação da Campanha

Nessa subseção, detalha-se o fluxo de preparação e execução de uma campanha de avaliação de disponibilidade utilizando o Chaos-K8s. A Figura 1 ilustra o fluxo completo de preparação de uma campanha de avaliação no Chaos-K8s, projetado para minimizar o

esforço de configuração inicial e permitir ajustes progressivos conforme as necessidades do administrador. A partir de um cluster Kubernetes em execução, o Chaos-K8s oferece mecanismos para extrair a topologia da infraestrutura, gerar configurações padrão e, opcionalmente, calibrar parâmetros de recuperação com base em medições empíricas. Esse processo estruturado permite que os usuários iniciem rapidamente campanhas de testes de disponibilidade, obtendo informações detalhadas sobre os tempos de falha e de recuperação de cada componente do sistema.

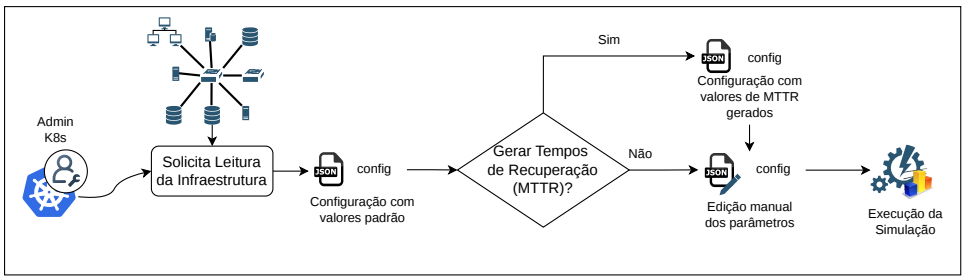


Figura 1. Fluxo de uso do Chaos-K8s.

Conforme apresentado na Figura 1, o fluxo se inicia com a solicitação da extração e leitura da infraestrutura do cluster. Essa etapa é fundamental porque permite identificar todos os componentes presentes (como nós, plano de controle e aplicações implantadas), capturando a topologia real do ambiente a ser avaliado. Dessa forma, o sistema pode gerar um arquivo de configuração inicial que reflete fielmente a infraestrutura existente, evitando erros decorrentes de configurações genéricas ou desatualizadas e reduzindo o esforço manual do administrador.

Após essa etapa, há uma decisão no fluxo: caso o administrador opte por não gerar automaticamente os tempos de recuperação (MTTR), ele pode editar manualmente os parâmetros de configuração no arquivo JSON, ajustando valores conforme as necessidades do cenário avaliado. Esse arquivo, gerado inicialmente com valores padrão para MTTF e MTTR de cada tipo de componente (baseados na literatura [Sebastio et al. 2021] ou valores hipotéticos), além de parâmetros da campanha como número de iterações, duração e critérios de disponibilidade, serve como ponto de partida para a configuração da campanha e pode ser utilizado imediatamente ou passar por etapas de refinamento manual.

Por outro lado, caso o administrador escolha gerar os tempos de recuperação (MTTR), o Chaos-K8s executa uma campanha de testes de falha controlados e mede empiricamente quanto tempo cada serviço leva para se recuperar. Dessa forma, os valores de MTTR são obtidos de forma automática e incorporados ao arquivo de configuração, substituindo os valores padrão. Em ambos os casos, os parâmetros de configuração (incluindo MTTF, MTTR, seleção de componentes-alvo e critérios de disponibilidade) podem ser revisados antes da execução dos testes de disponibilidade, oferecendo flexibilidade para ajustar a campanha conforme necessidades específicas do cenário avaliado. Por fim, a campanha é executada e, ao término, são obtidos os valores de disponibilidade da configuração avaliada, permitindo a análise quantitativa do impacto das decisões arquiteturais sobre a resiliência do sistema.

3.2. Detalhamento dos métodos

O Chaos-K8s possui uma arquitetura composta por três atividades principais, conforme apresentado na Figura 1: (i) descoberta da infraestrutura com geração de configuração inicial, (ii) caracterização empírica de tempos de recuperação (MTTR) por componente e (iii) execução de testes de disponibilidade. As subseções a seguir detalham cada um desses métodos, ilustradas nas Figuras 2, 3 e 4, respectivamente. O foco está na organização conceitual dos módulos e fluxos do sistema, e não em detalhes de implementação ou linguagem utilizada.

3.2.1. Descoberta de Infraestrutura (Get_Config)

A Figura 2 apresenta o fluxo detalhado do método de descoberta de infraestrutura, que corresponde à primeira atividade do fluxo geral (Figura 1). Esse método é responsável por mapear automaticamente a topologia do cluster e gerar o arquivo de configuração inicial.

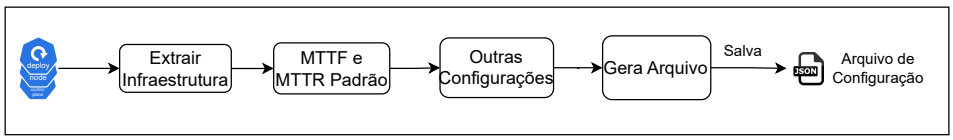


Figura 2. Fluxo do método *Get_Config*: descoberta automática da infraestrutura e geração do arquivo de configuração.

O módulo de descoberta interage com o cluster para obter, de forma automatizada, uma visão consolidada da infraestrutura. Em alto nível, ele identifica: (i) os principais componentes do *control plane*, responsáveis por manter o estado global e coordenar o agendamento de cargas; (ii) os nós *workers* disponíveis, que fornecem recursos de processamento para as aplicações; e (iii) as implantações (*deployments*) de aplicação em execução, incluindo a distribuição de réplicas entre os nós. Essas informações são organizadas em um arquivo de configuração que resume a topologia atual do ambiente e serve de base para as etapas seguintes da avaliação de disponibilidade.

Esse arquivo contém valores padrão de MTTF e MTTR para cada categoria de componente, além de parâmetros da campanha como número de iterações, duração total e critérios de disponibilidade por aplicação. O arquivo resultante serve como ponto de entrada para as etapas seguintes do fluxo (Figura 1) e pode ser editado manualmente pelo administrador para ajustar parâmetros específicos do cenário de teste antes da execução.

3.2.2. Caracterização de MTTR (Get_Config_All)

A Figura 3 ilustra o fluxo do método de caracterização empírica de MTTR, que estende a descoberta básica adicionando medições reais de tempos de recuperação. Esse método corresponde à atividade opcional de calibração indicada no fluxo geral (Figura 1).

Após realizar a descoberta de infraestrutura (mesmas etapas da método anterior), o sistema inicia uma campanha automatizada para medir o MTTR de cada componente identificado. O processo itera sobre os componentes (de 1 até n), aplicando técnicas específicas para cada camada: (i) Pods: encerramento do processo principal

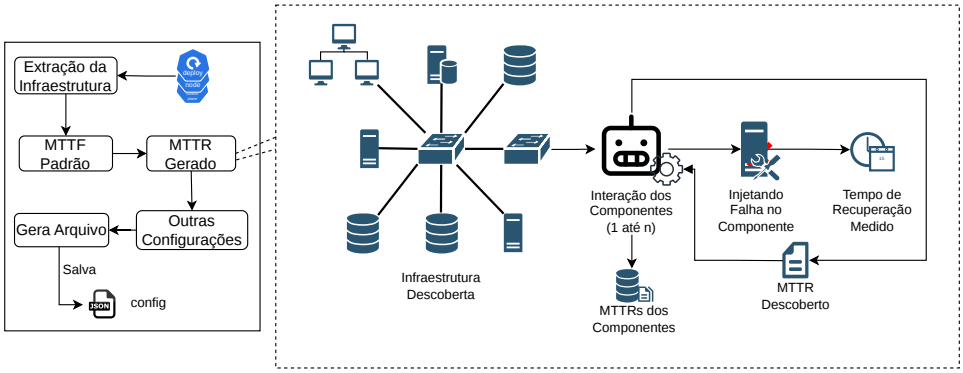


Figura 3. Fluxo do Método *Get_Config_All*: descoberta de infraestrutura combinada com caracterização empírica de MTTR por componente.

do contêiner, forçando o Kubernetes a reiniciar o *pod* conforme a política de reinício configurada; (ii) nós *workers*: execução de comandos de reinicialização de serviços críticos como `containerd` (runtime de contêineres) e `kubelet` (agente do nó) via conexão SSH; e (iii) Plano de controle: interrupção de processos críticos como `etcd` e `kube-apiserver`, simulando falhas nos componentes centrais do cluster.

Para cada componente-alvo, o módulo de monitoramento registra o instante de início da falha (T_{inicio}) e aguarda ativamente a recuperação, verificando periodicamente o estado de saúde por meio de consultas de *liveness* e requisições HTTP às aplicações. O instante de recuperação (T_{fim}) é registrado quando o componente retorna ao estado operacional, permitindo calcular o tempo de indisponibilidade observado como $MTTR = T_{fim} - T_{inicio}$. Ao final da campanha, o arquivo JSON é atualizado automaticamente com os valores de MTTR medidos empiricamente, substituindo os valores padrão por dados reais do ambiente.

3.2.3. Teste de Disponibilidade

A Figura 4 apresenta o fluxo do método de teste de disponibilidade, que corresponde à terceira atividade do fluxo geral (Figura 1). Esse método utiliza o arquivo de configuração (inicial ou calibrado) para executar campanhas de teste de falha e medir a disponibilidade do sistema.

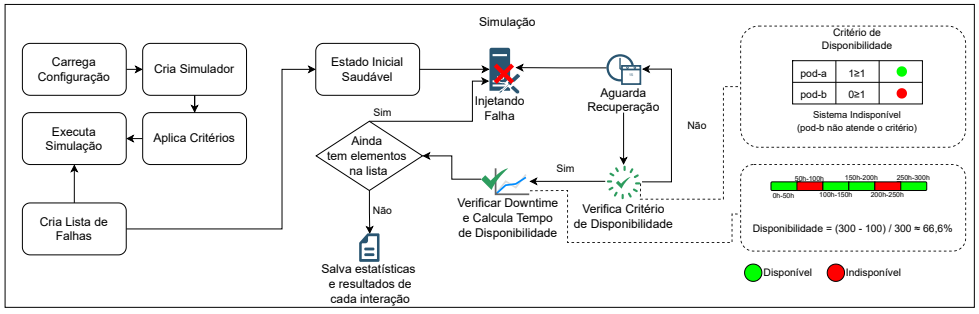


Figura 4. Fluxo do método de teste de disponibilidade: simulação de falhas baseada em critérios configurados e lista de componentes-alvo.

O processo inicia-se com o carregamento do arquivo de configuração JSON, seguido pela instanciação do módulo de teste de disponibilidade. O administrador define critérios de disponibilidade, como o número mínimo de réplicas saudáveis por *pod* ou serviço para que o sistema seja considerado disponível. Em seguida, é gerada a agenda de eventos de falha a partir dos parâmetros de MTTF (horizonte de tempo lógico, que pode ser acelerado), e a campanha é executada com injeções controladas e monitoramento contínuo, registrando janelas de indisponibilidade e recuperação de acordo com os MTTR configurados.

Durante a execução, o estado do sistema varia entre condições de disponível e indisponível conforme os eventos de falha ocorrem e os componentes se recuperam. A cada evento de falha, o Chaos-K8s aguarda a recuperação conforme os valores de MTTR configurados e verifica se os critérios de disponibilidade são atendidos. Ao longo das iterações (campo `iterations`), o fluxo acumula estatísticas detalhadas, incluindo: (i) downtime total, tempo acumulado em que o sistema permaneceu indisponível durante a campanha; (ii) número de eventos de falha aplicados, quantidade de eventos de falha aplicados a cada tipo de componente; (iii) tempo disponível, período em que o sistema atendeu aos critérios de disponibilidade definidos; e (iv) disponibilidade percentual, razão entre o tempo disponível e o tempo total da campanha, calculada para cada configuração de *pods* avaliada.

3.3. Ciclo de Execução do Teste e Monitoramento

O ciclo de execução de um teste de disponibilidade no Chaos-K8s é orquestrado a partir de uma lista de eventos de falha que é gerada no início da campanha com base nas configurações de MTTF definidas no arquivo de configuração, coordenando a interação entre os módulos da solução. O fluxo completo de uma execução segue as etapas descritas a seguir e ilustradas na Figura 5, esclarecendo o funcionamento interno do sistema durante a campanha de testes de falha e monitoramento contínuo.

1. **Verificação inicial:** são realizadas requisições às aplicações para estabelecer a *baseline* do sistema.
2. **Seleção do alvo:** escolhe-se o próximo componente e método de falha conforme a lista de eventos e tipo (pod, nó ou plano de controle).
3. **Execução do teste de falha:** o procedimento de falha é aplicado ao componente selecionado e o instante de início (T_{inicio}) é registrado.
4. **Monitoramento:** são realizadas verificações periódicas para detectar a recuperação do componente, registrando o instante de fim (T_{fim}).
5. **Cálculo e registro:** o MTTR é calculado como $T_{fim} - T_{inicio}$ e os resultados são registrados para análise. O ciclo retorna à Etapa 2 até processar todos os alvos configurados.

3.3.1. Forma de Calcular a Disponibilidade

Ao final de cada campanha de testes, o Chaos-K8s calcula a disponibilidade do sistema com base nos dados de *downtime* acumulados durante a campanha. A disponibilidade é definida como a fração do tempo total em que o sistema atendeu aos critérios de disponibilidade configurados (por exemplo, número mínimo de réplicas saudáveis por aplicação). A fórmula utilizada é:

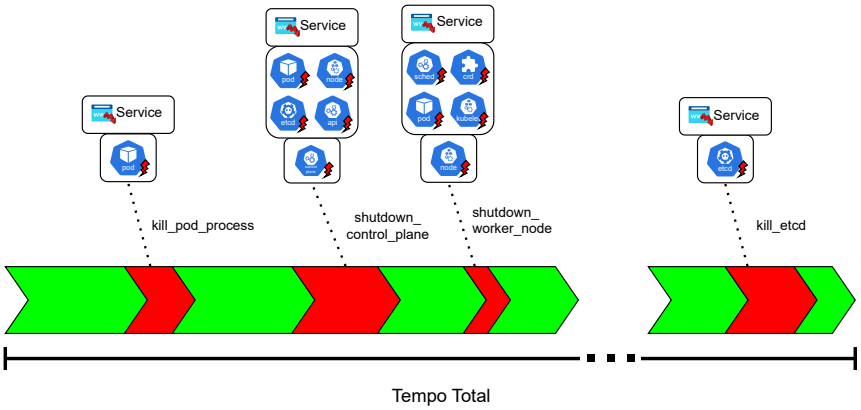


Figura 5. Fluxo de execução de um teste de disponibilidade.

$$A = \frac{T_{total} - T_{downtime}}{T_{total}} = \frac{T_{uptime}}{T_{total}} \tag{1}$$

onde A representa a disponibilidade (valor entre 0 e 1), T_{total} é a duração total da campanha (campo `duration` do arquivo JSON), $T_{downtime}$ é o tempo acumulado em que o sistema permaneceu indisponível (soma dos períodos de indisponibilidade detectados pelo módulo de monitoramento), e $T_{uptime} = T_{total} - T_{downtime}$ é o tempo em que o sistema permaneceu disponível.

Durante a execução do teste, o módulo de monitoramento verifica continuamente se os critérios de disponibilidade são atendidos. Quando o número de réplicas saudáveis de uma aplicação cai abaixo do limiar configurado no campo `availability_criteria`, o sistema é considerado indisponível e o tempo de indisponibilidade começa a ser contabilizado. A contagem cessa quando todas as aplicações voltam a atender seus critérios mínimos. O *downtime* total é calculado como a soma de todos os intervalos de indisponibilidade ao longo da campanha:

$$T_{downtime} = \sum_{i=1}^{n} (T_{fim,i} - T_{inicio,i}) \tag{2}$$

onde n é o número de eventos de indisponibilidade detectados e $(T_{fim,i} - T_{inicio,i})$ representa a duração de cada evento individual.

4. Validação por Modelagem SPN

Para aumentar a confiança nos resultados obtidos com o Chaos-K8s, foi realizada uma etapa de validação baseada em modelagem por SPN, seguindo a abordagem apresentada em trabalho anterior dos autores [Fé et al. 2023]. A validação tem como objetivo verificar se os resultados empíricos obtidos pela solução são estatisticamente consistentes com os valores analíticos derivados do modelo matemático.

O ambiente de validação foi configurado com um cluster Kubernetes composto por 1 nó de *control plane* e 2 nós *workers*. Foram implantadas 3 aplicações de teste, cada uma com 1 réplica (`bar-app`, `foo-app` e `test-app`). A escolha por uma única

réplica teve como objetivo estabelecer uma configuração mínima, suficiente para validar o modelo e a solução sem introduzir redundâncias adicionais que mascarassem os efeitos das falhas sobre a disponibilidade observada. As aplicações eram leves, projetadas apenas para fins de validação, retornando uma pequena resposta textual a cada requisição HTTP recebida. As execuções foram realizadas com 30 iterações, duração fictícia de 1000h (simulada) e intervalo de 10 segundos entre eventos de falha. Os valores de MTTF variaram de 100h (pods) a 800h (etcd), enquanto os valores de MTTR foram calibrados empiricamente, variando de aproximadamente 9,3s para o `kube-proxy` até cerca de 3 minutos para o `kube-apiserver` e `etcd`.

O modelo SPN representa os estados de funcionamento e falha dos componentes do cluster (*pods*, nós *workers* e serviços do *control plane*), utilizando como parâmetros de entrada os valores de MTTF e MTTR extraídos do arquivo JSON calibrado empiricamente pela ferramenta. As transições temporizadas do modelo seguem distribuições exponenciais com taxas $\lambda = 1/\text{MTTF}$ para falhas e $\mu = 1/\text{MTTR}$ para recuperações. A análise de estado estacionário foi realizada utilizando a ferramenta Mercury [Pinheiro et al. 2021], que permite calcular a disponibilidade de estado estacionário a partir da estrutura da rede e dos parâmetros configurados.

Para verificar a equivalência estatística entre os resultados empíricos e analíticos, foi aplicado o teste t de Welch, adequado para comparação de médias com variâncias potencialmente diferentes. A Tabela 2 apresenta os resultados da validação estatística.

Tabela 2. Resultados da validação estatística entre Chaos-K8s e modelo SPN.

Métrica	**Valor**
Disponibilidade média (Chaos-K8s)	0.9878 (98.78%)
Disponibilidade média (Modelo SPN)	0.9885 (98.85%)
Estatística t	-1.272
Graus de liberdade (Welch)	30.78
Valor-p	0.213
Validado (95% de confiança)?	**Sim**

A Figura 6 apresenta graficamente a comparação entre os resultados empíricos e analíticos, incluindo os intervalos de confiança de 95%. A proximidade entre as curvas demonstra a consistência entre a ferramenta proposta e o modelo matemático.

Os resultados indicam que não há diferença estatisticamente significativa entre as médias de disponibilidade obtidas pelo Chaos-K8s ($\bar{A} = 0.9878$) e pelo modelo SPN ($\bar{A} = 0.9885$), com valor-p de 0.213 (superior ao nível de significância $\alpha = 0.05$). A diferença absoluta entre as médias é inferior a 0.08%, demonstrando excelente concordância entre os resultados empíricos e analíticos. Essa validação confirma tanto a correção do processo de calibração automática de MTTR implementado na solução quanto a lógica de simulação de falhas e monitoramento de disponibilidade, fortalecendo a confiabilidade do Chaos-K8s como solução para avaliação quantitativa de disponibilidade de ambientes Kubernetes de produção.

5. Avaliação da Solução: Estudo de Caso

Esta seção apresenta um estudo de caso conduzido com o Chaos-K8s em um cluster Kubernetes de referência, com o objetivo de ilustrar o fluxo completo de uso da solução: descoberta da infraestrutura, geração e ajuste do arquivo de configuração em JSON, execução

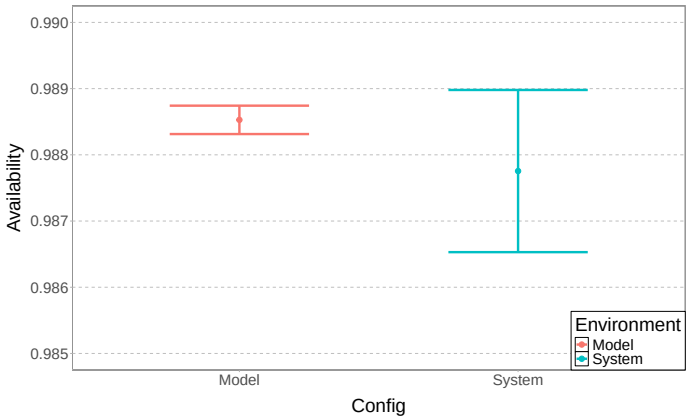


Figura 6. Comparação entre disponibilidade empírica (Chaos-K8s) e analítica (SPN) com intervalos de confiança de 95%.

dos testes de disponibilidade e análise dos resultados obtidos. O cenário considera três aplicações implantadas em um cluster com múltiplos nós *workers* e um nó de *control plane*. A partir desse ambiente, foram realizadas (i) a descoberta automática da topologia, (ii) a calibração dos tempos médios de recuperação (MTTR) por componente a partir de campanhas de escalonamento de falhas e (iii) a simulação de falhas para estimar disponibilidade e *downtime* por configuração de *pods*.

5.1. Configuração Inicial e Arquivo JSON Gerado

No primeiro método, foi utilizada a opção *Get_Config* do Chaos-K8s para realizar a extração da infraestrutura do cluster e gerar um arquivo de configuração inicial em formato JSON. Esse arquivo contém, entre outros campos, a seleção de aplicações e nós a serem considerados na campanha, além de valores padrão de MTTF e MTTR por tipo de componente. Um trecho simplificado dessa configuração inicial é mostrado a seguir:

Tabela 3. Parâmetros principais do arquivo de configuração inicial.

Parâmetro	Valor / Seleção
iterations	30
duration (s)	1000
delay (s)	10
applications	bar-app; foo-app; test-app
worker_node	ip-10-0-0-241; ip-10-0-0-98
control_plane	ip-10-0-0-28
availability_criteria	bar-app=1; foo-app=1; test-app=1
mttf_config	deployments; worker_node; control_plane
mttr_config	deployments; worker_node; control_plane

5.2. Calibração de MTTR e JSON Ajustado

No segundo método, foi utilizada a opção *Get_Config_All*, que combina a descoberta de infraestrutura com uma campanha automatizada de falhas de componentes para estimar o MTTR de cada componente. Para cada alvo (*pods*, nós *workers* e serviços do *control plane*), o módulo de testes de confiabilidade executou o ciclo descrito na Seção 3.3, registrando os tempos de início (T_{inicio}) e fim (T_{fim}) da indisponibilidade e calculando $MTTR = T_{fim} - T_{inicio}$.

Ao final dessa etapa, foi gerado e salvo um arquivo de configuração completo, no qual os campos de `mttr_config` passam a refletir os valores medidos empiricamente por componente. A Tabela 4 ilustra os parâmetros principais do arquivo JSON calibrado:

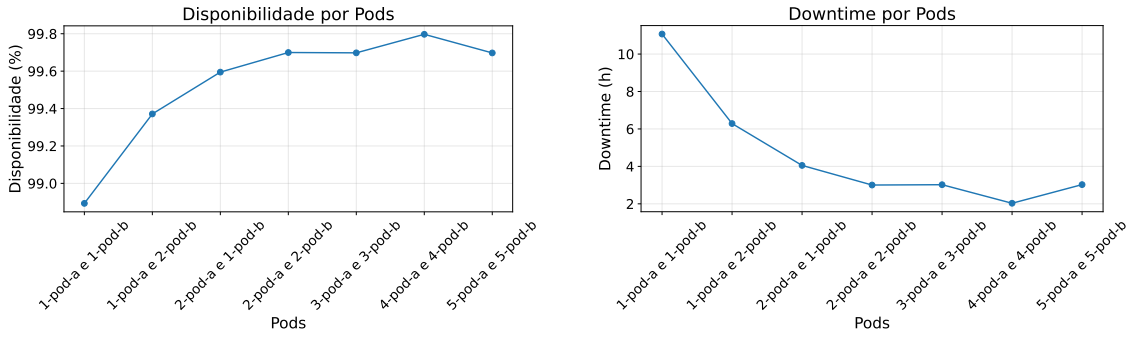
Tabela 4. Parâmetros de configuração MTTR/MTTF calibrados para o estudo de caso.

Componente	MTTF (h)	MTTR (h)
bar-app	100.0	0.05
foo-app	100.0	0.05
test-app	100.0	0.05
ip-10-0-0-141 (worker)	200.0	1.0
ip-10-0-0-98 (worker)	200.0	1.0
ip-10-0-0-85 (control plane)	500.0	1.0

Comparando-se os dois arquivos, observa-se que o JSON ajustado incorpora (i) a topologia efetiva descoberta no cluster (endereços IP de nós *workers* e do *control plane*) e (ii) valores de MTTR diferenciados por componente, obtidos a partir de medições reais de tempos de recuperação. Na implementação do estudo de caso, esse JSON calibrado é mantido no arquivo, que funciona como *template* completo para novas campanhas com o Chaos-K8s.

5.3. Execução do Teste de Disponibilidade e Resultados por Pods

Com o JSON calibrado, a opção *teste de disponibilidade* do Chaos-K8s foi empregada para executar campanhas ao longo de um tempo previamente definido, realizando a interrupção de processos essenciais de cada componente para avaliar a disponibilidade do sistema. Em cada iteração, o *framework* aplicou a lista de falhas configurada, alternando o sistema entre estados saudável e indisponível, aguardando a recuperação de acordo com os MTTR medidos e verificando se os critérios de disponibilidade (número mínimo de réplicas saudáveis por aplicação) eram atendidos. A Figura 7a apresenta o gráfico de disponibilidade estimada por configuração de *pods*, enquanto a Figura 7b mostra o *downtime* acumulado correspondente a cada configuração. Nesta versão, a campanha considera falhas injetadas de forma sequencial (um evento por vez), sem composições simultâneas entre componentes.



(a) Disponibilidade estimada. **(b) *Downtime* acumulado.**

Figura 7. Resultados por configuração de *pods* no estudo de caso: (a) disponibilidade estimada e (b) *downtime* acumulado.

Os resultados evidenciam que o aumento no número de réplicas de *pods* eleva a disponibilidade do sistema, pois a falha de uma réplica não causa indisponibilidade enquanto outras permanecem operacionais (**redundância ativa**). Consequentemente, o *downtime* acumulado diminui com mais réplicas, já que o sistema permanece disponível durante a recuperação via *self-healing*. Observa-se também retornos decrescentes: o ganho ao passar de 1 para 2 réplicas é maior do que ao passar de 4 para 5 [Limam et al. 2019], o que é relevante para balancear disponibilidade e custos operacionais.

6. Conclusão

Este trabalho apresentou o Chaos-K8s, uma abordagem para avaliação sistemática de disponibilidade de clusters Kubernetes, automatizando a descoberta da topologia, a calibração empírica de MTTR por componente, a execução de campanhas de escalonamento de falhas e a geração de relatórios com métricas de disponibilidade. O objetivo principal é permitir que administradores quantifiquem o impacto de diferentes configurações sobre a disponibilidade percebida pelos usuários finais. O estudo de caso demonstrou que configurações de *pods* (réplicas e distribuição entre nós) impactam significativamente a disponibilidade, e a validação por modelagem SPN confirmou a correção da solução, com boa concordância entre valores analíticos e empíricos. Como trabalhos futuros, pretende-se expandir o escopo para cenários de maior escala, falhas simultâneas/correlacionadas (rede e armazenamento), análise de overhead da instrumentação, comparação direta com ferramentas de caos e investigação de distribuições alternativas para tempos de recuperação.

Referências

Aderaldo, C. M. and Mendonça, N. C. (2025). Resiliencebench-operator: A kubernetes extension for orchestrating resilience experiments on microservice applications. In *Simpósio Brasileiro de Engenharia de Software (SBES)*, pages 983–989. SBC.

Baptista, G., Correia, J., Bento, A., Soares, J., Ferreira, A., Duraes, J., Barbosa, R., and Araujo, F. (2023). Defektor: An extensible tool for fault injection campaign management in microservice systems. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 184–187.

Barletta, M., Cinque, M., Martino, C., Kalbarczyk, Z., and Iyer, R. K. (2024). Mutiny! how does kubernetes fail, and what can we do about it? *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–14.

Carvalho, D., Rodrigues, L., Takako Endo, P., Kosta, S., and Airton Silva, F. (2020). Mobile edge computing performance evaluation using stochastic petri nets. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6.

Chen, Z., Goudarzi, M., and Toosi, A. N. (2025). Resilience evaluation of kubernetes in cloud-edge environments via failure injection. *arXiv preprint arXiv:2507.16109*.

Flora, J., Gonçalves, P., Teixeira, M., and Antunes, N. (2022). A study on the aging and fault tolerance of microservices in kubernetes. *IEEE Access*, 10:132786–132799.

Fé, I., Nguyen, T. A., Soares, A. B., Son, S., Choi, E., Min, D., Lee, J.-W., and Silva, F. A. (2023). Model-driven dependability and power consumption quantification of kubernetes-based cloud-fog continuum. *IEEE Access*, 11:140826–140852.

G1 (2025). Como pane em apenas um data center da amazon causou falhas para ifood, mercado livre e mais centenas de empresas. https://g1.globo.com/tecnologia/noticia/2025/10/21/mais-barato-e-mais-antigo-como-falha-em-um-data-center-da-amazon-afetou-ifood-mercado-livre-e-mais-centenas-de-empresas.ghtml. Acesso via conteúdo replicado e referências de busca.

Gortázar, F., Gallego, M., García, B., Carella, G. A., Pauls, M., and Gheorghe-Pop, I.-D. (2017). Elastest — an open source project for testing distributed applications with failure injection. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–2.

Grafana Labs (2026). k6: Open-source load testing tool. `https://k6.io/`.

Hecht, M. and Agena, S. (2024). A reliability and availability model of a kubernetes cluster using sysml. *2024 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–7.

Ikeuchi, H., Ge, J., Matsuo, Y., and Watanabe, K. (2020). A framework for automatic failure recovery in ict systems by deep reinforcement learning. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 1310–1315.

Limam, S., Mokadem, R., and Belalem, G. (2019). Data replication strategy with satisfaction of availability, performance and tenant budget requirements. *Cluster Computing*, 22:1199 – 1210.

Nguyen, T., Fé, I., Brito, C., Kaliappan, V. K., Choi, E., Min, D., Lee, J.-W., and Silva, F. A. (2021). Performability evaluation of load balancing and fail-over strategies for medical information systems with edge/fog computing using stochastic reward nets. *Sensors (Basel, Switzerland)*, 21.

Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., and Kim, S. (2020). Horizontal pod autoscaling in kubernetes for elastic container orchestration. *Sensors (Basel, Switzerland)*, 20.

Pinheiro, T., Oliveira, D., Matos, R., Silva, B., Pereira, P., Melo, C., Oliveira, F., Tavares, E., Dantas, J., and Maciel, P. (2021). The mercury environment: a modeling tool for performance and dependability evaluation. In *Intelligent Environments 2021*, pages 16–25. IOS Press.

Sebastio, S., Ghosh, R., and Mukherjee, T. (2021). An availability analysis approach for deployment configurations of containers. *IEEE Transactions on Services Computing*, 14(1):16–29.

Silva, F. A., Rodrigues, M., Maciel, P., Kosta, S., and Mei, A. (2015). Planning mobile cloud infrastructures using stochastic petri nets and graphic processing units. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 471–474.

Thakare, S. (2025). The societal imperative of resilient cloud infrastructure: Beyond business continuity. *Global Journal of Engineering and Technology Advances*.

Vayghan, L. A., Saied, M. A., Toeroe, M., and Khendek, F. (2019). Kubernetes as an availability manager for microservice applications. *arXiv preprint arXiv:1901.04946*.