Decisões de design de observabilidade informadas e avaliáveis em nuvem nativa Aplicações de Microsserviços

Pré-impressão, compilada em 16 de julho de 2024

Maria C. Borges1 , Joshua Bauer1 , Sebastião Werner1 , Michael Gebauer1 , e Stefan Tai1

[1]Engenharia de Sistemas de Informação, Technische Universitat Berlin, Alemanha

Resumo

A observabilidade é importante para garantir a confiabilidade dos aplicativos de microsserviços. Esses aplicativos são frequentemente propensos a falhas, pois possuem muitos serviços independentes implantados em ambientes heterogêneos. Quando Empregada "corretamente", a observabilidade pode ajudar os desenvolvedores a identificar e solucionar falhas rapidamente. No entanto, instrumentar e configurar a observabilidade de um aplicativo de microsserviço não é trivial, mas sim dependente de ferramentas e vinculado aos custos. Os arquitetos precisam entender as compensações relacionadas à observabilidade para ponderar entre diferentes alternativas de design de observabilidade. Ainda assim, essas decisões de design arquitetônico não são apoiadas por métodos e normalmente confiam apenas na "intuição profissional".

Neste artigo, defendemos um método sistemático para chegar a uma observabilidade informada e continuamente avaliável. decisões de design. Especificamente, focamos na observabilidade de falhas de aplicativos de microsserviços nativos da nuvem e transformar isso em uma propriedade testável e quantificável. Para atingir nosso objetivo, primeiro modelamos a escala e o escopo de decisões de design de observabilidade em toda a pilha nativa da nuvem. Em seguida, propomos métricas de observabilidade que pode ser determinado para qualquer aplicação de microsserviço por meio dos chamados experimentos de observabilidade. Apresentamos uma implementação de prova de conceito de nossa ferramenta experimental Oxn. Oxn é capaz de injetar falhas arbitrárias em um aplicação, semelhante à Engenharia do Caos, mas também possui a capacidade única de modificar a observabilidade configuração, permitindo a avaliação de decisões de design que antes eram inexploradas. Demonstramos nossa abordagem usando um aplicativo de microsserviço de código aberto popular e mostramos as compensações envolvidas diferentes decisões de design de observabilidade.

Palavras-chave: Observabilidade, Microsserviços, Arquitetura de Software, Compensações de Design de Software

1 Introdução

As práticas de observabilidade de monitoramento, rastreamento e registro desempenham um papel importante para garantir a confiabilidade dos serviços nativos da nuvem arquiteturas de microsserviços. Os microsserviços oferecem muitas vantagens sobre seus predecessores monolíticos [8], no entanto, os aplicativos resultantes podem ser difíceis de solucionar quando um ocorre uma falha [20, 29]. Isso ocorre porque cada solicitação geralmente viaja por meio de vários serviços desenvolvidos de forma independente, implantados em uma pilha intrincada e em evolução de plataformas de software. Para Para enfrentar este desafio, os sistemas de observabilidade evoluíram em conformidade, consistindo agora em serviços distribuídos que recolhem dados de observabilidade de todos os níveis de plataformas de tempo de execução também desde o nível da aplicação até a instrumentação integrada e definida pelo desenvolvedor. Portanto, o uso desses sistemas de observabilidade exige uma série de decisões difíceis em termos de pontos de instrumentação, serviços de observabilidade e configuração.
Quando empregada corretamente, a observabilidade pode ajudar os desenvolvedores identificar falhas rapidamente, mas configurações inadequadas também podem ofuscar falhas [5, 29] e aumentam latências e custos.

Ainda assim, projetar a observabilidade de aplicações de microsserviços não é trivial. Implica tarefas frequentemente esquecidas como a configuração de parâmetros dependente de ferramenta, configuração de alertas e adicionando código de instrumentação personalizado. Essas decisões precisam ser ponderado em relação às compensações relacionadas com a observabilidade, por exemplo, sobrecarga de desempenho no aplicativo, além da sobrecarga de custo associada à operação da infraestrutura de observabilidade.
No entanto, estas decisões de design arquitectónico não são apoiadas por métodos sistemáticos. Em vez disso, as decisões são tomadas com base em experiência anterior e "intuição profissional" dos praticantes [28, 29], ou os desenvolvedores podem acabar instrumentando e alterando ansiosamente os parâmetros de configuração de maneira reativa após a ocorrência de um problema [20]. Para ponderar entre diferentes projetos alternativas e chegar a uma configuração apropriada que justifique o esforço, os profissionais devem ter um método para avaliar a eficácia de sua observabilidade.

Consequentemente, a necessidade de avaliar o grau de observabilidade de um sistema foi reconhecida pela pesquisa e pela indústria. semelhantes [1, 9, 27, 28]. Com métricas adequadas, seria possível obter uma compreensão concreta da qualidade da observabilidade, que poderia então ser rastreada e continuamente aprimorada ao longo do tempo. No entanto, a observabilidade tem sido desafiadora quantificar até agora [27]. A maioria das pesquisas se concentrou em avaliações qualitativas de ferramentas de observabilidade [12], ou apenas analisa o custo e a sobrecarga [6, 7, 22], deixando de abordar a questão da eficácia da observabilidade. Atualmente, não há mecanismos para que os profissionais quantifiquem ou comparem a observabilidade de suas aplicações.

Neste artigo, defendemos uma avaliação sistemática, reprodutível e comparativa das decisões de projeto de observabilidade. Nosso foco está especificamente na observabilidade de falhas. Semelhante a medidas de qualidade de software, como cobertura de teste, nosso objetivo é fazer observabilidade de falhas, uma propriedade do sistema testável e quantificável. Isso, por sua vez, ajudaria os desenvolvedores a identificarem elementos não observáveis falhas na sua aplicação, orientando assim a configuração e processo de instrumentação.

Para atingir esse objetivo, apresentamos as seguintes contribuições:

1. Um modelo para compreender a escala e o escopo do ob-
decisões de design de servibilidade.
2. O conceito de métricas de observabilidade de falhas testáveis e
quantificáveis.
3. Projeto, implementação e demonstração do Oxn, um
ferramenta para automatizar o processo de avaliações de observabilidade.

O artigo está estruturado da seguinte forma: a Seção 2 discute questões relacionadas
trabalho. A Seção 3 fornece o contexto e modela o espaço de design de
observabilidade de arquiteturas nativas da nuvem. Seção 4
Apresentamos nossas métricas de observabilidade e método experimental.
A Seção 5.1 apresenta nossa ferramenta de suporte para experimentos,
Oxn. A Seção 6.3 avalia nosso método e ferramenta por meio de experimentos
exemplares. A Seção 7 discute adequação e limitações. A Seção 8 conclui
com trabalhos futuros.

2 Trabalhos Relacionados

Quando ocorrem falhas em composições complexas de microsserviços, elas
pode ser mais complicado de identificar do que em monólitos tradicionais, portanto,
as práticas de observabilidade evoluíram de acordo (por exemplo,
[13, 17, 24]) e a observabilidade continua sendo um assunto crescente na literatura
sobre confiabilidade. Múltiplas decisões de projeto devem ser tomadas
ao implementar e usar ferramentas de observabilidade. Diferentes
abordagens baseadas em modelos foram propostas para automatizar a
implementação de projetos de observabilidade [14, 21], mas nenhum deles
essas abordagens ajudam os profissionais a chegar a decisões apropriadas
e avaliáveis.

As decisões de design de observabilidade têm consequências significativas
porque, como Niedermeier et al. [20] apontam, “a implantação e configuração
descuidadas de agentes de monitoramento foram mencionadas como problemas
potenciais que podem causar instabilidade e
uma carga de rede crescente”. Até agora, diferentes abordagens têm
foi proposto para lidar com esse desafio de design.

Haselbock e Weinreich [10] propõem modelos de orientação de decisão para decisões
de observabilidade de microsserviços. Esses modelos ajudam
desenvolvedores com decisões de alto nível mais abstratas, por exemplo,
qual ferramenta adotar quando o objetivo é inspecionar interações de serviço. O
modelo serve bem como um recurso introdutório, mas
não pode fornecer assistência com configurações mais complexas ou
decisões de instrumentação. Por exemplo, não oferece orientação sobre a seleção
de métricas apropriadas ou a determinação do ideal
taxa de amostragem. Na mesma linha, [23] fornecem as melhores práticas
para apoiar decisões de projeto de rastreamento. Além desses guias,
os desenvolvedores também podem consultar pesquisas como [12] para tomar
decisões, onde as ferramentas de observabilidade são comparadas com base em
vários critérios qualitativos diferentes.

Algumas pesquisas também abordam as compensações do design de observabilidade,
em particular os custos ou despesas gerais de desempenho incorridos. Ernst
e Tai [7] propõem uma abordagem offline para rastrear a avaliação de sobrecarga.
A abordagem baseada em modelo gera rastreamento realista
dados, que podem então ser usados como uma carga de trabalho em diferentes
rastreando backends. Um benchmark concreto de instrumentação de observabilidade
também foi conduzido por [22]. Aqui, os pesquisadores propõem uma ferramenta
para medição contínua da
sobrecarga de bibliotecas de instrumentação populares como OpenTelemetry e
Kieker. Mais recentemente, a eficiência energética de ferramentas de observabilidade
também foi investigada [6]. Os pesquisadores
cobriu uma associação estreita entre o consumo de energia relacionado à
observabilidade e a sobrecarga de desempenho, um resultado que
era esperado. Todos os três artigos abordam as desvantagens da observabilidade,
mas ignoram as vantagens que ela oferece.
não é viável avaliar compensações sem compreender a
valor fornecido por algo.

Nosso trabalho é o primeiro a orientar escolhas de design de observabilidade
através da medição comparativa da sua eficácia. Ahmed
et al. [1] fizeram um trabalho semelhante, pois mediram a eficácia de quatro
diferentes ferramentas de monitoramento estão identificando regressões de
desempenho. No entanto, seu trabalho se concentra principalmente na comparação
as ferramentas sem nos aprofundarmos nos detalhes da instrumentação e
decisões de configuração necessárias para implementar cada ferramenta,
como apenas a configuração padrão e automática pronta para uso
instrumentação é usada para a avaliação. Outro método promissor
abordagem para melhorar a resiliência e testar a configuração
dos sistemas de observabilidade é a Engenharia do Caos. Ela fornece
um método [2] e ferramentas [11, 18, 26, 30, 30] para realizar experimentos de
resiliência. Aqui, falhas são injetadas em um
sistema de microsserviços para testar uma hipótese sobre a capacidade dos sistemas
para resistir a essas falhas. No entanto, esta abordagem pressupõe
um nível suficiente de observabilidade antes de conduzir os experimentos e não
oferece nenhuma orientação sobre como atingir isso.

Por último, Nedelkoski et al. [19] destacam a falta de conjuntos de dados
incorporando dados de telemetria de diversas fontes (métricas,
rastros, logs). Eles propõem uma solução através do desenvolvimento de um sistema
de microsserviços que simula falhas injetadas, possibilitando a
criação de novos conjuntos de dados, semelhante à nossa abordagem. Infelizmente,
a incapacidade do sistema de alterar ou configurar a observabilidade e pontos
específicos de instrumentação telemétrica limita significativamente
sua capacidade de avaliar a eficácia da observabilidade.

3 Decisões de Design de Observabilidade de Modelagem

Garantir a operação confiável de aplicações baseadas em microsserviços é uma
tarefa complexa que requer observabilidade. Observabilidade, conforme definida na
literatura, é a capacidade de mensurar a
estado interno de um sistema por suas saídas [20]. O processo de
definir essas saídas é o que definimos como “decisões de projeto de observabilidade”.
Nesta seção, nos aprofundamos nas complexidades
e nuances dessas decisões modelando um aplicativo de microsserviço nativo da
nuvem típico implantado de acordo com
práticas modernas. Com este modelo, pretendemos mostrar a escala
e escopo das decisões de design de observabilidade, além de servir como
uma linguagem comum para os profissionais entenderem e discutirem
alternativas de design de observabilidade. A Figura 1 mostra nosso modelo,
que detalhamos nos parágrafos seguintes.

Uma **aplicação nativa em nuvem** é implantada em um ambiente de nuvem
gerenciado por um provedor terceirizado. O modelo representa a topologia da
aplicação como uma ou mais instâncias de implantação.
clusters (Cluster), implantados em uma região específica e gerenciados
por um orquestrador. O orquestrador é uma tecnologia de implantação, por
exemplo, Kubernetes1 ou Docker Compose2 , que automatiza
a implantação e o gerenciamento de contêineres em máquinas virtuais (VMs). Cada
contêiner executa uma instância de um microsserviço. Modelamos um microsserviço
como uma composição de

1https://kubernetes.io
2https://docs.docker.com/compose/

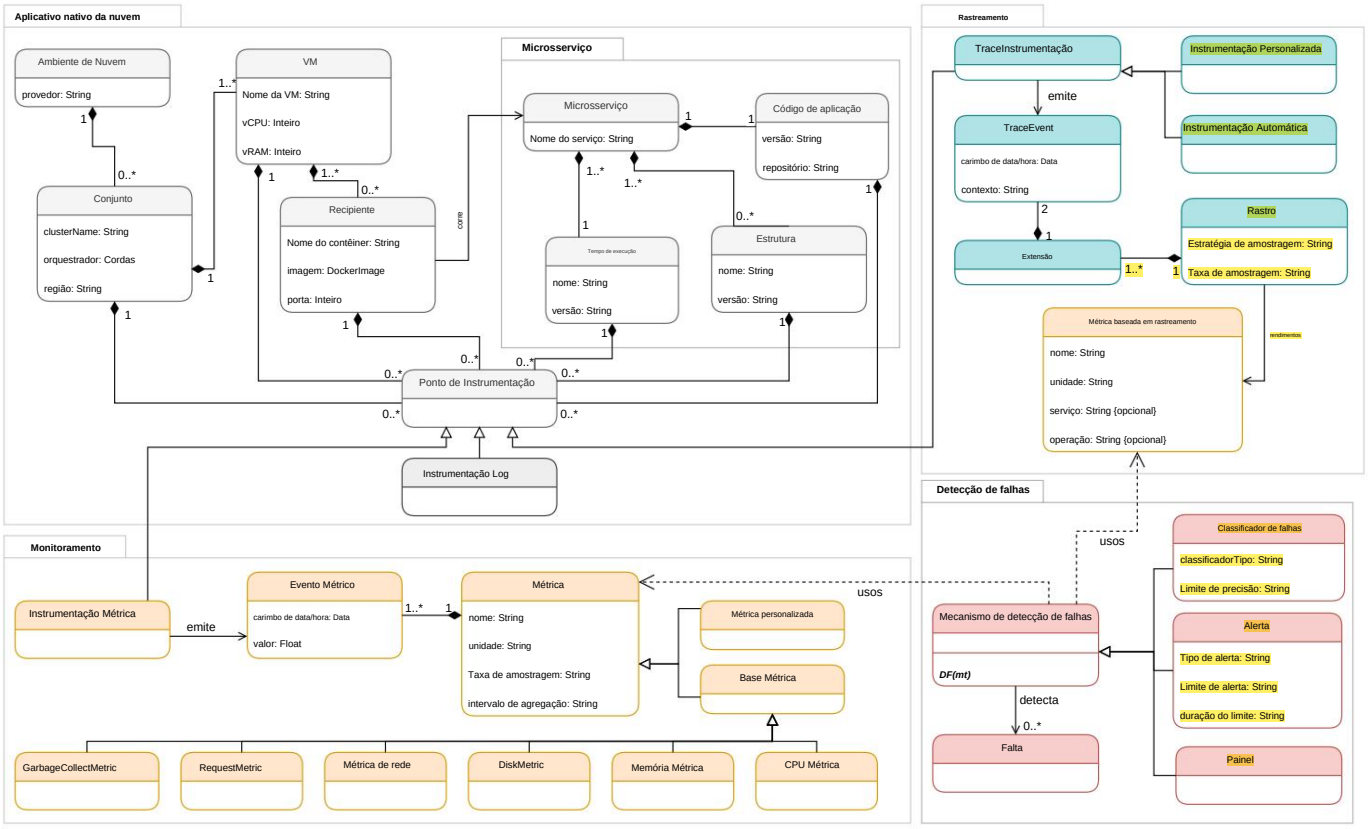

Figura 1: Modelo de decisões de design de observabilidade em aplicativos nativos da nuvem

Um Runtime, que fornece o ambiente específico da linguagem do microsserviço, por exemplo, o Java Runtime. Os microsserviços também podem ser compostos por diversos Frameworks, que oferecem componentes reutilizáveis que agilizam o processo de desenvolvimento, como Django ou gRPC. Por fim, os microsserviços também são compostos pelo Código da Aplicação escrito pela equipe de desenvolvimento.

Para capturar dados de observabilidade, um aplicativo nativo da nuvem precisa ser instrumentado por meio dos chamados InstrumentationPoints. Instrumentação refere-se ao processo de incorporar código em um componente de software que permite coletar e emitir métricas, rastreamentos e logs. Alguns componentes da pilha de implantação nativa da nuvem são equipados com a chamada instrumentação automática, o que significa que os pontos de instrumentação são fornecidos prontos para uso. Dependendo dos dados de observabilidade que coletam, os InstrumentationPoints podem ser do tipo MetricInstrumentation, LogInstrumentation e TraceInstrumentation. Cada tipo de instrumentação suporta uma prática de observabilidade diferente. Embora o registro seja igualmente importante para a construção de sistemas de software confiáveis, focamos principalmente na modelagem de monitoramento e rastreamento neste caso, já que o registro é menos padronizado para criar um modelo genérico além de mensagens de texto legíveis por humanos e com registro de data e hora.

Monitoramento é a medição contínua das propriedades quantitativas de um sistema ao longo de um período de tempo. Pontos de Instrumentação Métrica emitem MetricEvents, que são medições de valores com registro de tempo. Uma Métrica, por sua vez, representa a coleção de séries temporais de vários MetricEvents. Eles podem ser simples BaseMetrics, coletados diretamente por meio de in-

Pontos de instrumentação fornecidos em Clusters, VMs, Contêineres, Tempos de Execução e Frameworks. Na Figura 1, mostramos vários desses tipos de Métricas Base, mas a lista não pretende ser exaustiva. Métricas também podem ser especificadas e instrumentadas por desenvolvedores no caso de Métricas Personalizadas. Ao projetar o monitoramento de uma aplicação, os profissionais enfrentam diversas decisões de projeto, incluindo (1) quais Métricas Base coletar dos Pontos de Instrumentação automáticos disponíveis, (2) se devem ou não adicionar Métricas Personalizadas e, em caso afirmativo, onde adicionar os Pontos de Instrumentação, (3) como configurar os atributos samplingRate e aggregationInterval para cada instância de Métrica.

O rastreamento permite a observação ponta a ponta do comportamento do aplicativo, recuperando e agregando dados de eventos nos chamados rastreamentos. Nesse contexto, um TracingInstrumentation-Point, que pode ser do tipo AutomaticInstrumentation se suportado nativamente pelo microsserviço Runtime ou Framework, ou do tipo CustomInstrumentation se adicionado ao ApplicationCode pela equipe de desenvolvimento, emite um TraceEvent. Um par de TraceEvents demarcando a entrada e a saída cria um Span, e múltiplos spans formam um Trace. As informações dentro dos traces podem ser posteriormente processadas em TraceBasedMetrics, por exemplo, a duração dos spans ou de traces inteiros. Ao projetar o rastreamento de um aplicativo, os profissionais novamente se deparam com diversas decisões de design, a saber: (1) onde adicionar CustomInstrumentation e (2) como configurar os atributos samplingStrategy e samplingRate.

Para **detecção de falhas,** os sistemas de observabilidade empregam os chamados Mecanismos de detecção de falhas para poder detectar falhas.
Esses mecanismos podem empregar uma variedade de métodos de detecção, incluindo Classificadores pré-treinados, Alertas ou tendo um engenheiros treinados em confiabilidade de site analisam os painéis. Para falhas detecção, os profissionais primeiro (1) precisam selecionar um mecanismo de detecção de falhas viável e, em seguida, (2) definir atributos apropriados para não sobrecarregar as equipes de confiabilidade e ainda fornecer resposta rápida a falhas potenciais. Naturalmente, as decisões de projeto anteriores sobre registro, monitoramento e rastreamento influenciam fortemente a qualidade de esses métodos de detecção, pois poucas entradas podem levar a uma baixa sensibilidade e muitas observações levam a muito ruído e, portanto, pode levar à hipersensibilidade.

4 Abordagem para quantificar a observabilidade eficácia

Para chegar a decisões de projeto de observabilidade informadas e avaliáveis, precisamos ser capazes de quantificar sua eficácia. A observabilidade serve como um precursor necessário para medir muitas outras qualidades do sistema, como desempenho [1] ou custo [15], mas a observabilidade em si é muito desafiadora de quantificar [27]. Sua eficácia está intrinsecamente ligada à própria aspectos que pretende capturar. Neste trabalho, examinamos a observabilidade para o propósito específico de garantia de confiabilidade, ou seja, a processo de garantir que o sistema esteja funcionando apesar das falhas ocorrendo.

A garantia de confiabilidade gira em torno de duas métricas principais: (1) tempo médio até a falha / tempo médio entre falhas (MT TF/MT BF) e (2) tempo médio de restauração (MT TR). O MT TR pode ser dividido em (2.1) tempo médio de detecção (MT TD) e (2.2) tempo médio de reparo (MT TRepair). O objetivo de manter um alto MT TF geralmente fica além do domínio da observabilidade e, em vez disso, depende de testes completos. O MT TR, por outro lado, é influenciado pela presença de medidas de observabilidade.

Ainda assim, as métricas do tipo MT TD e MT TR não são específicas para mecanismos de observabilidade, mas se aplicam a todo o processo operacional, incluindo quaisquer outros mecanismos de tolerância a falhas implementados. Eles servem para rastrear o efeito de longo prazo das medidas de confiabilidade e de equipes de confiabilidade, mas são muito grosseiras e podem ser influenciadas por muitos fatores sobrepostos diferentes. Por exemplo, eles não indicar se uma falha poderia ter sido detectada antes ou em menos custo com uma configuração de observabilidade diferente. Para chegar em um método sistemático para decisões de design de observabilidade, nós Precisamos medir o impacto das decisões de configuração e instrumentação. Portanto, precisamos ampliar nossa perspectiva para além das métricas de processo baseadas em tempo. A seguir, propomos um primeiro conjunto de métricas refinadas como base para tal sistematização método.

4.1 Conceito: Métricas de Observabilidade de Falhas

Para chegar a métricas testáveis e explícitas para decisões de design de observabilidade, restringimos nosso escopo à visibilidade de falhas, ou o que chamamos de **visibilidade de falhas** [5]. A visibilidade de falhas pode ser definida como o grau em que os dados produzidos durante a ocorrência de uma falha específica é suficientemente distinta da operação normal para acionar um mecanismo de detecção de falhas. Intuitivamente, falhas que produzem dados muito distintos são mais fáceis e rápidas de solucionar.

visibilidade de falhas, consideramos um conjunto de falhas F = {f1, f2, ..., fl}, a conjunto de métricas de observabilidade ou traços M = {m1, m2, ..., mn} e um mecanismo de detecção d. Uma falha f pode ser considerada visível em métrica m se os dados registrados durante a ocorrência da falha, representado pelo intervalo de tempo [t0 ÿ t1], é significativamente diferente dos dados registrados em condições normais, em termos de tempo intervalo [t-1 ÿ t0[, e assim detectado pelo mecanismo de detecção d. Consulte a Figura 2(A) para uma representação visual desses tempos intervalos. Conforme modelado na seção 3, o mecanismo de detecção podem ser de diferentes tipos, por exemplo, um classificador pré-treinado. Consideramos a falha f visível na métrica m se o método de detecção d for capaz de atingir seu limite de detecção e detectar a falha. Se a detecção mecanismo não é capaz de detectar a falha, isso sugere que (i) a métrica m não é adequada para detectar f (ii) os parâmetros da métrica m não estão definidos adequadamente ou (iii) os parâmetros da função do mecanismo de detecção d não estão ajustados corretamente. Visibilidade da falha para a falha f na métrica m através do mecanismo de detecção d pode ser definido como

$$vf,m,d = \begin{cases} 1 & DF(mt) > ÿ \\ 0 & \text{de outra forma} \end{cases} \quad (1)$$

onde DF representa a função de detecção de d, ÿ o limite configurado e mt um subconjunto de observações de m para o tempo t.

A pontuação de visibilidade pode ser determinada para cada falha e par métrico, permitindo-nos avaliar o impacto de falhas individuais em métricas específicas, dada a sua configuração atual. No entanto, essas pontuações individuais não são capazes de fornecer uma visão abrangente e compreensão generalizada da observabilidade do sistema como um todo. Para isso, podemos construir agregados ou pontuações compostas. Propomos duas pontuações compostas: falha cobertura e observabilidade geral de falhas. **A cobertura de falhas** mostra o grau em que uma falha f é visível no conjunto de diferentes métricas coletadas M. Nós o definimos como a razão entre o número de métricas coletadas e um número de métricas onde a falha f é visível. Se uma falha não for visível em nenhuma métrica do sistema, isso implica que essa falha não é coberta pelo sistema observabilidade e, portanto, a cobertura de falhas para a falha f é 0.

$$FCf,d = \frac{1}{n} \sum_{i=1}^{n} vf,mi ,d \quad (2)$$

Por último, a **observabilidade geral da falha** mostra a relação entre as falhas teoricamente observáveis versus as falhas que foram realmente observado ou detectado pelo mecanismo de detecção d. Isto a proporção pode melhorar ao longo do tempo, à medida que os desenvolvedores adicionam mais métricas a instrumentação ou ajustar a configuração. Pode então ser definido como

$$OFOd = \frac{1}{eu} \sum_{i=1}^{eu} 1\{FCi>0\} \quad (3)$$

A Figura 2(B) ilustra nossas métricas de observabilidade de falhas sugeridas usando um exemplo. O principal objetivo dos desenvolvedores deve ser aumentar o OFO. Para isso, os desenvolvedores têm três opções: aumentar a visibilidade de falhas invisíveis alterando a configuração das métricas existentes ou adicionar métricas adicionais que são mais sensíveis às falhas invisíveis, ou fazem mudanças ao mecanismo de detecção. Para ponderar entre diferentes opções, os desenvolvedores precisam considerar a compensação entre a sobrecarga de observabilidade e o custo. Assim, como última métrica para a eficácia da observabilidade, propomos uma métrica de custo. Aqui,

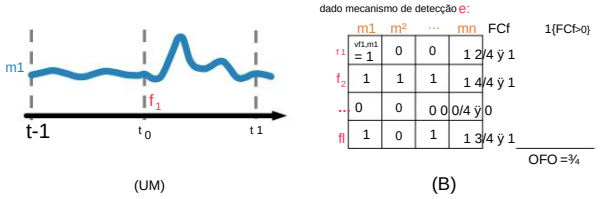

Figura 2: (A) Visualização do modelo de falha, utilizado para definir as métricas em (B)

o trabalho [6, 7, 22] já forneceu diferentes métodos e abordagens para quantificar a sobrecarga dos sistemas de observabilidade. Para para os propósitos deste artigo, usaremos a utilização da CPU como um medida de custo, mas outras métricas como memória e armazenamento sobrecarga também são adequadas.

4.2 Abordagem: Experimentos de Observabilidade

Propomos uma abordagem empírica e sistemática para navegar o espaço de design de observabilidade: experimentos de observabilidade. Isto é um processo formalizado de experimentação inspirado na Engenharia do Caos. Na Engenharia do Caos, falhas são injetadas em um sistema de microsserviços em execução para testar a capacidade do sistema de resistir a falhas. Em nossa abordagem, em vez de focar em se o sistema sobrevive a uma falha, estamos particularmente interessados em analisar a qualidade dos dados de observabilidade gerados durante tal experimentos de falhas. Essa qualidade pode ser quantificada com as métricas propostas na seção anterior e pode ser medida em várias configurações de observabilidade e instrumentação alternativas. Os profissionais podem usar experimentos de observabilidade como um ciclo de feedback para avaliar as decisões de projeto de observabilidade, avaliando especificamente quais mudanças na maquinaria e configuração da instrumentação melhoram ou degradam as métricas, todas as ao mesmo tempo em que se mantém atento ao custo dessas mudanças.

A seguir, descrevemos brevemente a abordagem. Um experimento de observabilidade consiste formalmente em uma descrição do sistema em experimento (SUE), uma carga de trabalho, um conjunto de tratamentos e um conjunto de métodos de avaliação. e um conjunto não vazio de variáveis de resposta.

Sistema em Experimento (SUE) No contexto de experimentos de observabilidade, o SUE é a totalidade de um sistema ou um subconjunto dele, ou seja, apenas um número seleto de microsserviços. É muitas vezes é desnecessário implantar toda a arquitetura de microsserviços quando os desenvolvedores estão interessados apenas em estudar a observabilidade e instrumentação de um único serviço e seus dependentes serviços. Portanto, é importante permitir a experimentação em subconjuntos, especialmente considerando que as arquiteturas modernas de microsserviços podem crescer rapidamente para centenas de serviços [16].

Carga de trabalho Exigimos geração de carga para simular usuários que criar dados de observabilidade realistas. Porque a carga pode afetar os pontos de dados dos dados de observabilidade, incluímo-los como um componente do experimento. Além disso, como muitas das métricas elaboradas na seção anterior dependem de análise comparativa, é importante também ter uma maneira de gerar uma carga comparável repetidamente.

Tratamentos são mudanças controladas no sistema em experimento. Nossa conceituação é mais ampla do que na Engenharia do Caos, onde o tratamento é caótico ou destrutivo. Isso não precisa ser o caso com tratamentos em experimentos de observabilidade. Nós distinguir entre tratamentos de falhas e tratamentos de instrumentação. Os tratamentos de instrumentação permitem que os usuários alterem facilmente configuração de observabilidade sem ter que mexer no Código SUE e são executados durante a compilação. Tratamentos de falhas, por sua vez, são aplicados em tempo de execução e alteram o código SUE. por meio de injeção dinâmica de falhas. Os tratamentos nos permitem investigar mudanças no sistema normalmente além do escopo da Engenharia do Caos. Por exemplo, o operador do experimento pode estar interessado em investigar se um aumento no intervalo de amostragem em algum ponto de instrumentação métrica resulta em um aumento na precisão de um modelo de detecção de falhas.

Variáveis de resposta em experimentos de observabilidade são métricas ou traços distribuídos e representam os diferentes tipos de dados de observabilidade que um sistema pode emitir. Eles são usados para calcular a visibilidade da falha, a cobertura da falha e a observabilidade geral da falha. As respostas são dissociadas dos tratamentos. Isso significa que podemos definir uma variável de resposta que não esteja diretamente relacionada a um tratamento, ou seja, poderíamos observar uma resposta no serviço A ao tratar o serviço B. O desacoplamento nos permite levar em conta consideração dos efeitos de ordem superior dos tratamentos. Por exemplo, podemos querer investigar se injetar um atraso de rede em serviço A consequentemente também afeta o rendimento em outros serviços que dependem de A.

Tais experimentos são especialmente viáveis em ambientes de preparação para sistemas onde tanto o SUE quanto a observabilidade O sistema é descrito usando infraestrutura como código. A seguir, apresentamos um sistema para automatizar o processo de avaliações de observabilidade para aplicações baseadas em microsserviços.

5 OXN: Motor de Experimentos de Observabilidade

Nesta seção, apresentamos Oxn, uma estrutura de software extensível para executar experimentos de observabilidade3 . Oxn segue o design princípios para benchmarking de nuvem [3, 4, 25] e, portanto, se esforça particularmente para experimentos portáteis, repetíveis e relevantes. Construímos o Oxn em torno de um arquivo de configuração baseado em YAML que permite que experimentos sejam compartilhados, versionados e repetidos. a configuração do experimento descreve o SUE, ou seja, a implantação, tratamentos, carga de trabalho e variáveis de resposta a serem coletadas (ver seção 4.2). Usando este único arquivo de experimento, Oxn orquestra o experimento completo (ver Figura 3), primeiro construindo e implantando o SUE e também o gerador de carga. Usamos um componente de corredor extensível para executar diferentes tratamentos, como matar um contêiner de serviço, em combinação com a coleta automática de variáveis de resposta do experimento por meio de um componente observador. Para isso, contamos parcialmente com o SUE para implementar o padrão OpenTelemetry, com ferramentas do Pilha CNCF 4 , ou seja, Jaeger e Prometeu.

5.1 Arquitetura

O design do Oxn é modular, desacoplado e extensível. Em sua núcleo, Oxn combina um orquestrador para gerenciar o SUE, um executor para executar tratamentos, um gerador de carga para estressar o SUE e observadores para capturar resultados. Todos esses componentes usam software existente e bem testado, que acoplamos fracamente para

3https://github.com/nymphbox/oxn
4https://landscape.cncf.io

Tabela 1: Diferentes tratamentos implementados em oxn

Nome	Propósito	Ferramenta
Tratamento de falhas:		
Pausa	Simula um serviço sem resposta por suspendendo todos os processos em um serviço	Docker
Matar	Simula uma falha de serviço	Docker
Atraso de rede	Injeta atraso de rede em uma interface tc5	
Perda de pacotes	Injeta perda de pacotes em uma interface	tc
Corrupção de pacotes	Injeta corrupção de pacotes em uma interface	tc
Estresse	Simula o esgotamento de recursos injetando estressores	estresse-ng6
Tratamento de Instrumentação:		
Taxa de amostragem métrica	Altera o intervalo de amostragem para métricas	Coletor
TracingSamplingStrategy	Amostras de rastreamentos com base em uma determinada estratégia	Coletor
Taxa de amostragem de rastreamento	Amostras de rastreamento com base em uma determinada taxa de amostragem	Coletor

permitir experimentos de observabilidade. Assim, cada um pode ser substituído ou estendido para se adaptar a diferentes SUEs ou tratamentos. Por exemplo, usamos o Docker Compose para o orquestrador. No entanto, também poderíamos ter usado um orquestrador baseado em Kubernetes.
ou um baseado em CloudFormation. Para o corredor, contamos com uma interface Python que pode ser usada para manipular o SUE.
Para o gerador de carga, usamos o framework Locust, e para os observadores, implementamos interfaces para consultar Jaeger e Prometeu por enquanto.

Os tratamentos de falhas e instrumentação são definidos de forma flexível modelo de tratamento. A Tabela 1 mostra a falha e a instrumentação tratamentos implementados em nosso protótipo. Além destes tratamentos, tratamentos personalizados podem ser adicionados facilmente implementando uma nova classe de tratamento em nossa interface de tratamento. Isso permite que Oxn seja estendido com cenários de falhas arbitrários ou com novas abordagens de instrumentação. Assim, permitindo que os profissionais construam uma grande biblioteca de falhas e tratamentos relevantes para testar com Oxn.

Variáveis de resposta são capturadas por meio de múltiplos componentes. Um componente observador captura informações do experimento, por exemplo, carimbos de data/hora de início e término da execução. O componente de respostas recebe dados métricos e de rastreamento, rastreando variáveis definidas e também implementa diferentes estratégias de rotulagem, dependendo da tipo de dados. Um componente de armazenamento grava todas as informações capturadas como um formato de dados binários legível pela maioria das soluções de análise de dados. Um componente repórter pode gerar um formato legível por máquina relatório de experimento para dar aos engenheiros uma visão geral rápida da eficácia do projeto de observabilidade examinado.

Para calcular as métricas de visibilidade de falhas, o Oxn oferece uma ferramenta extensível interface, permitindo que os desenvolvedores integrem perfeitamente seus próprios mecanismos de detecção de falhas. Isso pode variar de limiar-

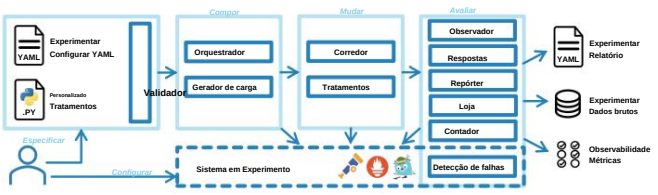

Figura 3: Arquitetura do sistema Oxn

5Controle de tráfego Linux

6Ferramenta de teste de carga e estresse do kernel Linux

técnicas de alerta baseadas em detecção de anomalias mais avançadas modelos. Por padrão, incluímos um classificador de regressão logística com Oxn, que pode ser treinado automaticamente para detectar falhas para um determinado limite de precisão, tornando-o prontamente disponível para desenvolvedores para usar imediatamente. No entanto, desenvolvedores e profissionais de observabilidade também podem integrar seu próprio mecanismo de detecção de falhas, permitindo-lhes testar tanto sua observabilidade configuração e seus mecanismos de detecção de falhas separadamente. Desta forma, o Oxn também pode ser usado para avaliar a detecção de falhas mecanismos que utilizam dados de observabilidade reais em vez de sintéticos vestígios comumente usados até agora pela pesquisa.

Por último, o componente do contador contém funcionalidade para estimar os custos da configuração de observabilidade fornecida com base em Uso da CPU. Este componente poderia ser estendido novamente para incluir outras métricas de custo, como sobrecarga de armazenamento ou memória.

6 Aplicabilidade e Observabilidade Exemplar Avaliação de Design

Nossa abordagem e ferramentas permitem que os profissionais avaliem a observabilidade de falhas em sua aplicação de microsserviços. Além de estabelecer a eficácia da configuração atual,
Eles também podem usá-lo para raciocinar entre alternativas de design de observabilidade, ponderando as compensações entre observabilidade e custo. seção, demonstramos a aplicabilidade de nossa abordagem conduzindo uma avaliação exemplar da observabilidade de um aplicativo de microsserviço de código aberto popular.

6.1 Configuração do SUE

Como nosso sistema em experimento (SUE), usamos o aplicativo de microsserviço Open-Telemetry Astronomy Shop Demo7,
que é um projeto comunitário que visa ilustrar o uso de diferentes métodos e ferramentas de observabilidade em um ambiente quase real. O aplicativo consiste em 20 serviços principais, além de quatro dedicados a serviços de observabilidade.
Escolhemos esta aplicação porque ela abrange uma ampla gama de linguagens e estruturas em todo o aplicativo nativo da nuvem pilha. Ele é instrumentado para coletar rastros em toda a aplicação, aproveitando instrumentação automática e personalizada.
Além dos rastros, ele também é instrumentado para coletar diversas métricas, incluindo métricas de integridade do contêiner base e métricas personalizadas adicionadas manualmente pelos desenvolvedores. É um SUE muito adequado. para apresentar o OXN, já que ele oferece um amplo espectro de botões de ajuste de observabilidade.

Para os experimentos, implantamos um fork do aplicativo8 para garantir compatibilidade com a funcionalidade de injeção de falhas de Oxn e para aprimorar o monitoramento do tempo de execução do contêiner. Executamos o Oxn contra o SUE em uma máquina virtual baseada em nuvem (8vCPUs, 32 GB de memória). A Figura 4 aplica nosso modelo de observabilidade decisões de design (ver 3) e mostra o instantâneo do nosso SUE no momento da medição de base. Direcionamos nossos experimentos para o serviço de recomendação, que reflete uma experiência da vida real cenário para desenvolvedores que, após criarem um novo serviço para sua aplicação, se deparam com decisões de instrumentação e configuração para poder observar esse serviço de forma eficaz.

7https://github.com/open-telemetry/ demonstração de telemetria aberta

8Consulte o repositório Oxn para obter detalhes.

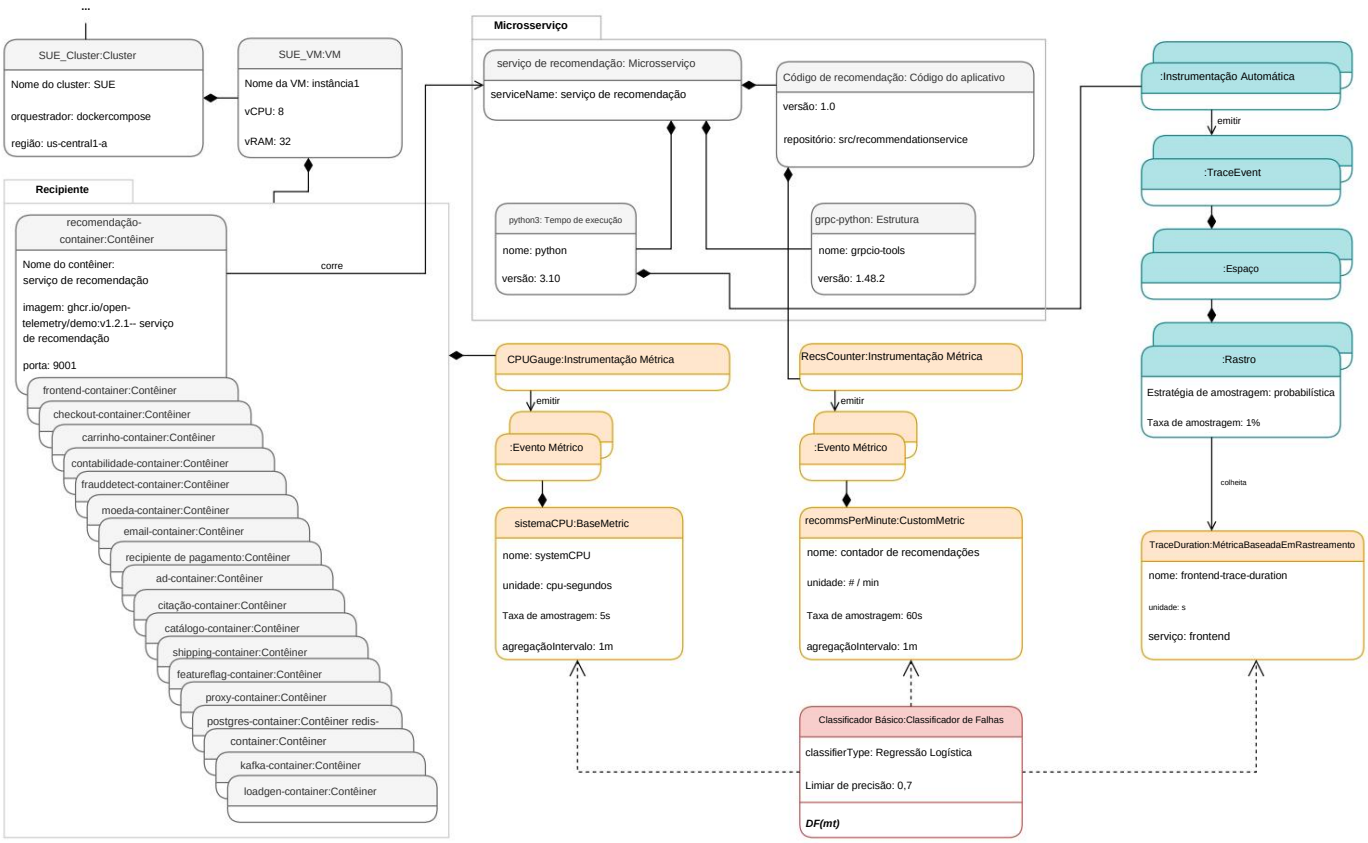

Figura 4: SUE com configuração de observabilidade de base

Nossa configuração de linha de base monitora a utilização geral da CPU do sistema (systemCPU) coletando e agregando medições dos CPUGauges de cada contêiner. Por padrão, as métricas de base são configuradas com uma taxa de amostragem de 5 segundos. Além disso, a linha de base também coleta a métrica personalizada recommsPerMinute, por meio de um ponto de instrumentação que os desenvolvedores adicionaram ao código do aplicativo de recomendação. Este ponto é configurado com uma taxa de amostragem de 60 segundos por padrão. Para o rastreamento, o aplicativo é instrumentado com instrumentação automática e personalizada; no entanto, o serviço de recomendação em questão utiliza apenas a instrumentação automática. A estratégia de amostragem de rastreamento é definida como um amostrador probabilístico9 com uma amostragem de 1% .
avaliar.

Como mecanismo de detecção de falhas, usamos o classificador de regressão logística que vem com o Oxn pronto para uso. A regressão logística é conceitualmente simples, interpretável, rápida de treinar em grandes conjuntos de dados e requer apenas o ajuste de um hiperparâmetro.
Para o treinamento, dividimos os dados do experimento em conjuntos de treinamento e teste e garantimos um equilíbrio de classes entre rótulos com e sem falhas por meio de uma técnica de sobreamostragem. Pré-processamos ainda os dados de telemetria por normalização de escore z. Após o treinamento, avaliamos a função de detecção do classificador (DF(mt)) nos dados de teste e calculamos a precisão da classificação com um limite de 0,7, como nossa implementação de ÿ para visibilidade de falhas.

9https://opentelemetry.io/docs/specs/otel/trace/ tracestate-probability-sampling/

6.2 Avaliação da linha de base - Resultados

Investigamos a observabilidade do nosso SUE com três tipos diferentes de falhas: Pausa, Perda de Pacotes e Atraso de Rede. Para cada falha, executamos um experimento de 10 minutos sob carga constante (50 usuários simultâneos) e repetimos cada experimento 10 vezes.

A Figura 5 plota as métricas que coletamos nos diferentes experimentos com nossa configuração de observabilidade de base. A Tabela 2 mostra a precisão do classificador, calculada em média ao longo das dez execuções do experimento, e as pontuações de visibilidade de falhas resultantes. Como definimos o limite ÿ do nosso classificador como 0,7, tudo o que estiver abaixo desse limite não será identificado como falha pelo nosso mecanismo de detecção de falhas e, portanto, será invisível. Como podemos ver, falhas como o tratamento de pausa, que simula um serviço sem resposta, manifestam-se de forma bastante proeminente nas métricas. A perda de pacotes também está presente na configuração de observabilidade de base, embora com uma pontuação de cobertura de falhas (FC) menor, uma vez que se manifesta apenas na CPU do sistema. Por fim, falhas como atraso, que poderiam, por exemplo, simular um cache defeituoso, são pouco visíveis nos gráficos e não são detectadas pelo nosso mecanismo de detecção de falhas, uma vez que a função de detecção não atinge o limite para nenhuma das métricas coletadas.

6.3 Avaliação de alternativas de design - Resultados

Para melhorar a observabilidade de falhas da nossa aplicação, consideramos três alternativas de design de observabilidade (veja Figura 6).
A primeira (Fig. 6A) é aumentar a taxa de amostragem do

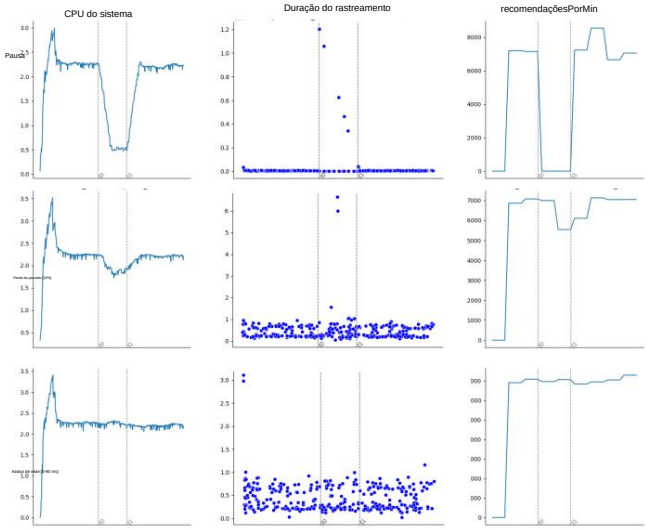

Figura 5: Experimentos executados contra o SUE usando Oxn, mostrando como diferentes falhas aparecem visualmente, semelhante a como um desenvolvedor os veria em um painel. Observe como a falha de pausa é visível em todas as métricas. A perda de pacotes é perceptível na CPU do sistema, mas menos pronunciado em outras métricas, com NetworkDelay não sendo visível de forma alguma. Na Figura 7, vemos como as alterações na configuração de observabilidade propostas na Figura 6 afetam essas métricas.

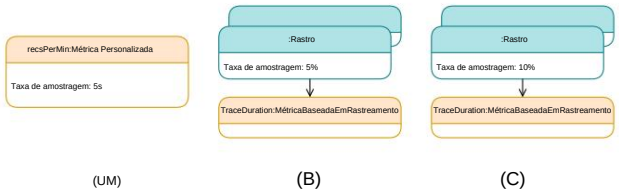

Figura 6: Alternativas de projeto de observabilidade em consideração

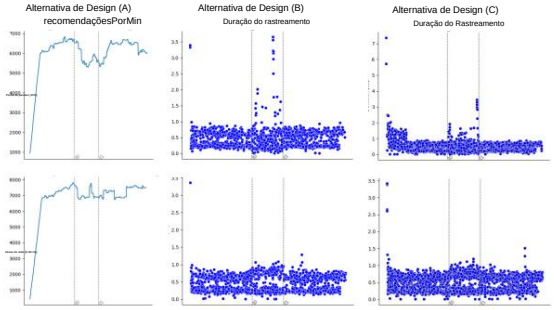

Figura 7: Mudanças visuais na visibilidade de falhas em todo o projeto alternativas. Reconhecer o efeito agora mais pronunciado de PacketLoss em recommsPerMin, com NetworkDelay ainda não afeta significativamente esta métrica. O NetworkDe-lay causa um ligeiro aumento perceptível na TraceDuration para os Traços mais longos para ambas as alternativas de design B e C.

Tabela 2: Métricas de visibilidade de falhas para configuração de observabilidade padrão

	Utilitário de CPU.	Rastrear Dur.	#Recs/min	FC
Pausa	1 (DF = 0,83)	1 (1,00)	1 (0,89)	3/3
Perda de pacotes [15%]	1 (0,86)		0 (0,61) 0 (0,42)	1/3
Atraso de rede [0-90 ms]	0 (0,50)		0 (0,61) 0 (0,43)	0/3
				OFO = 2/3

Tabela 3: Efeitos das alternativas de projeto nas métricas de observabilidade de falhas

	Perda de pacotes [15%]	Atraso de rede [0-90 ms]	ýFC	ýOFO
(A)	1 (DF=0,72)	0 (0,59)	+1	0
(B)	0 (0,58)	1 (0,77)	+1	+1
(C)	0 (0,60)	1 (0,70)	+1	+1

Tabela 4: Custo das alternativas de projeto em tempo de CPU [s]

	SUE de linha de base (A)	(B)	(C)
recomendo serviço	143,30 150,74 143,86 144,63		
hotel-col	37,38 37,49 39,99	39,05	
prometheus	9,01 9,07 9,07 9,48		
jaeger	2,14 2,17 5,70 7,96		
total	191,83 199,47 197,68 202,06		
overhead	-	+3,98% +3,05%	+5,33%

contador de recomendações em nível de aplicativo, que por padrão é definido para relatar apenas uma vez a cada minuto. As outras opções em a consideração é aumentar a taxa de amostragem de rastreamento para 5% (Fig. 6B) ou 10% (Fig. 6C).

Para cada alternativa de design de observabilidade, repetimos o Experimentos PacketLoss e NetworkDelay, mantendo todos os outros aspectos do SUE e da configuração do experimento iguais. Figura 7 plota as métricas coletadas após as alterações descritas acima. A Tabela 3 mostra os resultados relativos às melhorias na observabilidade de falhas, enquanto a Tabela 4 revela os custos associados à realização cada uma dessas decisões de design de observabilidade (como tempo de CPU [s] somados aos serviços afetados).

A partir destes resultados, podemos concluir que o aumento do intervalo para o contador de recomendações fornece melhor cobertura de falhas para falhas de PacketLoss, um aumento de 1/3 para 2/3, mas não afeta o OFO, uma vez que a falha do NetworkDelay ainda não está visível. A segunda opção oferece melhor cobertura de falhas para Falhas de NetworkDelay, um aumento de 0/3 para 1/3, bem como um aumento no OFO = 3/3, agora que a falha de atraso finalmente foi visível. A terceira opção tem um desempenho semelhante à segunda em as métricas de observabilidade de falhas. Ao analisar a observabilidade-compensações relacionadas na forma de custo (ver também tabela 4), vemos que enquanto B são C opções viáveis para melhorar o OFO, B está vinculado a um custo menor que C, com 3,05% de overhead em vez de 5,33%.

Assim, ao aplicar o modelo, as métricas e as ferramentas apresentadas em No artigo, podemos encontrar uma decisão de projeto com melhor observabilidade de falhas e podemos obter imediatamente uma primeira avaliação de impacto no desempenho. Um tomador de decisão pode agora julgar se o maior o custo é aceitável e implementar a mudança. Caso contrário, mais experimentos de observabilidade podem ser realizados para encontrar uma solução melhor. Além disso, o modelo também fornece aos desenvolvedores e profissionais com uma linguagem comum para entender, discutir e documentar suas decisões de design de observabilidade.

7 Limitações e Trabalho Futuro

Embora este artigo represente um passo inicial no desenvolvimento de métricas de observabilidade e a exploração de otimização estratégias, é importante reconhecer certas limitações. Em primeiro lugar, a abordagem baseia-se na simulação e em experiências isoladas, que podem não captar totalmente as complexidades e falhas enfrentados em aplicações do mundo real. Para isso, projetamos deliberadamente o Oxn com a extensibilidade em mente, permitindo que os desenvolvedores integrem suas próprias curvas de carga e falhas personalizadas. no futuro, pretendemos validar ainda mais a abordagem em condições reais cargas e cenários de falhas mais extensos.

Em segundo lugar, as métricas individuais propostas para a observabilidade das falhas foram deliberadamente mantidas claras e flexíveis, para permitir uma avaliação independente de tecnologia. Uma generalização para mais sistemas complexos de detecção de falhas podem não ser possíveis sem alto acoplamento ao mecanismo de relatórios. Ainda assim, consideramos nossa abordagem foi validada, pois fomos capazes de mostrar o impacto de diferentes opções de configuração na observabilidade gerada dados, evidentes tanto nos gráficos quanto no classificador pontuação. Como parte do nosso esforço contínuo para expandir a observabilidade avaliações, pretendemos aprimorar nossa ferramenta incorporando suporte para sistemas de alerta e Tempo Médio de Detecção (MTTD). Além disso, também pretendemos melhorar a análise de trade-off incorporando outras métricas de custo além da utilização da CPU.

Terceiro, a implementação atual do Oxn tem algumas limitações, notavelmente a ausência de certos recursos como suporte para logs[10] , Integração com Kubernetes, entre outros. Nosso objetivo é abordar esses à medida que continuamos a utilizar nossa ferramenta em avaliações de observabilidade adicionais. Em trabalhos futuros, planejamos explorar como as decisões de observabilidade podem ser otimizadas e automatizadas. Utilizando nossas métricas como uma fundação, nosso próximo objetivo é mergulhar em inteligência e abordagens de aprendizagem para otimizar e ajustar parâmetros de configuração de observabilidade.

8 Conclusão

À medida que a observabilidade se torna uma propriedade cada vez mais indispensável das aplicações de microsserviços e a configuração se torna cada vez mais complexa, há também uma preocupação crescente em relação como configurar e instrumentar melhor uma aplicação. Para habilitar desenvolvedores e profissionais para avaliar a adequação de diferentes projetos, propusemo-nos a tornar a observabilidade uma propriedade do sistema testável e quantificável, semelhante às medidas de qualidade do software como cobertura de teste.

Neste artigo, apresentamos uma abordagem para avaliar e aprimorar a observabilidade de falhas em aplicações de microsserviços baseadas em nuvem. Nossa abordagem consiste em um modelo para compreensão e documentar o espaço de projeto de observabilidade, um conjunto de métricas para avaliar a observabilidade de falhas e uma ferramenta que fornece evidências quantitativas para decisões de projeto de observabilidade que vão além da intuição visceral ou profissional. Mostramos como essa abordagem pode ser usado na prática, com uma aplicação do modelo para documentar o espaço de design de observabilidade de um aplicativo de microsserviço baseado em nuvem de última geração. Além disso, avaliamos múltiplos designs, revelando a observabilidade até então inexplorada.

[10] No momento da implementação, o OpenTelemetry não havia finalizado registros.

compensações de custos. Nosso objetivo é melhorar o Oxn adicionando suporte para mais plataformas como o Kubernetes, integrando a ferramenta em Pipelines de CI e aumentando a automação e a inteligência do processo de avaliação. Com a ajuda de outros profissionais, o modelo e as ferramentas também podem ser estendidos para cobrir mais falhas cenários e ambientes de microsserviços mais diversos. Com neste trabalho, lançamos as bases para um método sistemático para apoiar decisões de design de observabilidade em nuvem nativa aplicações de microsserviços.

Agradecimentos

Financiado pela União Europeia (TEADAL, 101070186). Visualizações e as opiniões expressas são, no entanto, apenas do(s) autor(es) e não refletem necessariamente os da União Europeia. Nem a União Europeia Nem o sindicato nem a autoridade concedente podem ser responsabilizados por eles.

Referências

[1] Tarek M. Ahmed, Cor-Paul Bezemer, Tse-Hsun Chen, Ahmed E. Hassan e Weiyi Shang. Estudo da eficácia de ferramentas de gerenciamento de desempenho de aplicações (APM) para detectar regressões de desempenho em aplicações web: um relato de experiência. Na Proc. da 13ª Conferência Internacional sobre Repositórios de Software de Mineração, página 1–12, 2016.

[2] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds e Casey Rosenthal. Caos engenharia. IEEE Software, 33(3):35–41, 2016.

[3] David Bermbach, Jorn Kuhlenkamp, Akon Dey, Arunmoezhi Ramachandran, Alan Fekete e Stefan Tai. Fundição de bancada: Uma estrutura de benchmarking para serviços de armazenamento em nuvem. Em inglês. Conf. sobre Computação Orientada a Serviços, páginas 314–330, 2017.

[4] David Bermbach, Erik Wittern e Stefan Tai. Serviço em nuvem Benchmarking: Medindo a qualidade dos serviços em nuvem a partir de uma Perspectiva do Cliente. Springer, Cham, 2017.

[5] Maria C. Borges, Sebastian Werner e Ahmet Kilic. Mais rápido solução de problemas - avaliação de abordagens de rastreamento distribuído para Aplicações sem servidor. Na Conferência Internacional IEEE de 2021 sobre Nuvem Engenharia, páginas 83–90, 2021.

[6] Madalina Dinga, Ivano Malavolta, Luca Giamattei, Antonio Guerriero e Roberto Pietrantuono. Uma avaliação empírica da sobrecarga de energia e desempenho das ferramentas de monitoramento em sistemas baseados em Docker. Na Conferência Internacional sobre Orientação a Serviços Computação, páginas 181–196, 2023.

[7] Dominik Ernst e Stefan Tai. Geração de rastreamento offline para observabilidade de microsserviços. No IEEE 25th Intl. Enterprise Workshop de Computação de Objetos Distribuídos, páginas 308–317, 2021.

[8] Martin Fowler e James Lewis. Microsserviços, 2014.

[9] Google Architecture Center. Medição de Devops: Monitoramento e observabilidade, 2023.

[10] Stefan Haselbock e Rainer Weinreich. Modelos de orientação de decisão para monitoramento de microsserviços. Em IEEE Intl. Conf. sobre Workshops de Arquitetura de Software, páginas 54–61, 2017.

[11] Victor Heorhiadi, Shriram Rajagopalan, Hani Jamjoom, Michael K. Reiter e Vyas Sekar. Gremlin: Teste sistemático de resiliência de microsserviços. Em 2016, na 36ª Conferência Internacional do IEEE. sobre Sistemas de Computação Distribuída, páginas 57–66, 2016.

[12] Andrea Janes, Xiaozhou Li e Valentina Lenarduzzi. Abrir ferramentas de rastreamento: Visão geral e comparação crítica, 2022.

[13] Jonathan Kaldor, Jonathan Mace, Michaÿ Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, Vinod Venkataraman, Kaushik Veeraraghavan e Yee Jiun Song. Canopy: Um sistema de rastreamento e análise de desempenho de ponta a ponta. Em Proc. do 26º Simpósio sobre Princípios de Sistemas Operacionais, páginas 34–50, 2017.

[14] John Klein, Ian Gorton, Laila Alhmoud, Joel Gao, Caglayan Gemici, Rajat Kapoor, Prasanth Nair e Varun Saravagi. Observabilidade orientada a modelos para armazenamento de big data. Em 2016, 13ª Conferência IEEE/IFIP sobre Arquitetura de Software (WICSA), páginas 134–139, 2016.

[15] Jorn Kuhlenkamp e Markus Klems. Costradamus: Um sistema de rastreamento de custos para serviços de software baseados em nuvem. Em Conferência Internacional sobre Computação Orientada a Serviços, páginas 657–672, 2017.

[16] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun e Xuanzhe Liu. Aproveite sua observabilidade: uma pesquisa industrial sobre rastreamento e análise de microsserviços. Engenharia de Software Empírica, 27(1):1–28, 2022.

[17] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao e Yanbo Han. Malha de serviços: desafios, estado da arte e oportunidades futuras de pesquisa. Conf. Internacional IEEE de 2019 sobre Engenharia de Sistemas Orientada a Serviços, páginas 122–1225, 2019.

[18] Christopher S. Meiklejohn, Andrea Estrada, Yiwen Song, Heather Miller e Rohan Padhye. Teste de injeção de falhas em nível de serviço. Em ACM Symp. on Cloud Computing, páginas 388–402, 2021.

[19] Sasho Nedelkoski, Jasmin Bogatinovski, Ajay Kumar Mandap-ati, Soeren Becker, Jorge Cardoso e Odej Kao. Dados de sistema distribuído de várias fontes para análises baseadas em IA. Em euros. Conf. sobre computação orientada a serviços e em nuvem, páginas 161–176, 2020.

[20] S. Niedermaier, F. Koetter, A. Freymann e S. Wagner. Sobre observabilidade e monitoramento de sistemas distribuídos – um estudo de entrevistas na indústria. Em Conferência Internacional sobre Computação Orientada a Serviços, páginas 36–52, 2019.

[21] Chadarat Phipathananunth e Panuchart Bunyakiati. Monitoramento sintético em tempo de execução de arquiteturas de software de microsserviços. 42ª Conferência Anual de Software e Aplicações de Computador (COMPSAC) do IEEE de 2018, volume 02, páginas 448–453, 2018.

[22] David Georg Reichelt, Stefan Kuhne e Wilhelm Hasselbring. Comparação de overhead entre opentelemetry, inspectit e kieker. Em Simpósio sobre Desempenho de Software, 2021.

[23] Raja R. Sambasivan, Ilari Shafer, Jonathan Mace, Benjamin H. Sigelman, Rodrigo Fonseca e Gregory R. Ganger. Rastreamento centrado em fluxo de trabalho de sistemas distribuídos. Em ACM Symp. on Cloud Computing, SoCC '16, páginas 401–414, 2016.

[24] Benjamin H. Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan e Chandan Shanbhag. Dapper, uma infraestrutura de rastreamento de sistemas distribuídos em larga escala. Relatório técnico, Google Research, 2010.

[25] Marcio Silva, Michael R. Hines, Diego Gallo, Qi Liu, Kyung Dong Ryu e Dilma da Silva. Cloudbench: Automação de experimentos para ambientes de nuvem. Em Conferência Internacional sobre Engenharia de Nuvem, páginas 302–311, 2013.

[26] Jesper Simonsson, Long Zhang, Brice Morin, Benoit Baudry e Martin Monperrus. Observabilidade e engenharia do caos em chamadas de sistema para aplicações em contêineres no Docker. Future Generation Computer Systems, 122:117–129, 2021.

[27] Damian A. Tamburri, Marcello M. Bersani, Raffaela Mirandola e Giorgio Pea. Observabilidade de serviços Devops por design: Experimentando com modelo-visão-controlador. Em Eur. Conf. sobre Computação em Nuvem e Orientada a Serviços, páginas 49–64, 2018.

[28] Guilherme Vale, Filipe Figueiredo Correia, Eduardo Martins Guerra, Thatiane de Oliveira Rosa, Jonas Fritzsch e Justus Bogner. Projetando sistemas de microsserviços usando padrões: um estudo empírico sobre compensações de qualidade. Em 2022, IEEE 19ª Conferência Internacional sobre Arquitetura de Software (ICSA), páginas 69–79, 2022.

[29] He Zhang, Shanshan Li, Zijia Jia, Chenxing Zhong e Cheng Zhang. Arquitetura de microsserviços na realidade: uma investigação industrial. Em 2019, Conferência Internacional IEEE sobre Arquitetura de Software (ICSA), páginas 51–60, 2019.

[30] Long Zhang, Brice Morin, Benoit Baudry e Martin Monper-rus. Maximizando o realismo da injeção de erros para engenharia de caos com chamadas de sistema. IEEE Transactions on Dependable and Secure Computing, 19(4):2695–2708, 2022.