UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA
PROGRAMA DE EDUCAÇÃO CONTINUADA
PÓS-GRADUAÇÃO LATO SENSU
MBA EM ENGENHARIA DE SOFTWARE

NÍCOLAS SILVA VIEIRA DE OLIVEIRA - N. USP 14634024

Observabilidade Em Sistemas Distribuídos: Salesforce e AWS

São Paulo
2024

NÍCOLAS SILVA VIEIRA DE OLIVEIRA

Observabilidade Em Sistemas Distribuídos: Salesforce e AWS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Especialista em Engenharia de Software.

Área de Concentração: Engenharia e Computação.

Orientador: Prof. Dr. Alexandre dos Santos Mignon.

São Paulo
2024

AGRADECIMENTOS

Ao Dr. Alexandre Mignon, o qual tive o prazer de ser aluno e me possibilitou tanto aprendizado, além de ter me orientado na construção desse trabalho, de forma clara e construtiva, os seus direcionamentos foram importantes para o resultado final.

À minha amada esposa Ellen Gouveia, beletrista, que além de me ajudar emocionalmente nos momentos em que estive à beira do precipício, me ajudou nas construções dos textos e revisões.

Aos meus pais Arivaldo e Adriana, por me ajudarem a concluir a minha graduação e me darem suporte financeiro e emocional para iniciar a minha vida adulta e profissional, nunca conseguirei retribuir à altura todo o carinho e amor que vocês tiveram por mim.

Aos meus amigos mais próximos, por me aturarem nos momentos em que estive mais cansado. Nossas risadas foram alívios momentâneos necessários para que eu conseguisse terminar este trabalho.

RESUMO

O monitoramento de sistemas baseados em arquiteturas distribuídas em provedores nuvem apresenta desafios aos desenvolvedores. Provedores como a Salesforce, embora entreguem vantagens aos times de desenvolvimento - tais como a abstração da gestão do banco de dados e suporte ao desenvolvimento ágil - não oferecem ferramentas nativas que possibilitem o monitoramento detalhado das operações construídas em sua plataforma. Mesmo provedores como AWS, que possuem formas robustas de monitoramento, exigem que sejam feitas intervenções nos códigos ou configuração de ferramentas terceiras, para que seja possível obter a visão macro do processamento das operações, isso quando o sistema é construído utilizando os conceitos de arquitetura distribuída. A fim de resolver estes problemas, este trabalho propõe uma estrutura para captura e análise de logs, métricas e traces, em sistemas construídos utilizando Salesforce e AWS. A abordagem apresentada permite monitorar os processos que ocorrem nesses ambientes, e integrar os dados das transações que ocorrem em sistemas distintos, criando assim uma visão macro do fluxo de processamento. Desta forma, os resultados do trabalho foram a obtenção dos dados detalhados das execuções das operações na Salesforce para realização de depuração e viabilização de monitoramento proativo, além da criação de uma visão unificada de transações que ocorrem de forma distribuída, facilitando a identificação e resolução de problemas, e contribuindo para a eficiência e estabilidade operacional dos sistemas monitorados.

Palavras-chave: Monitoramento de sistemas; Salesforce; AWS; Arquitetura distribuída.

ABSTRACT

Monitoring systems based on distributed architectures in cloud providers presents significant challenges to developers. Providers such as Salesforce, while offering advantages to development teams - such as database management abstraction and support for agile development - do not provide native tools that enable detailed monitoring of operations built on their platform. Even providers such as AWS, which offer robust monitoring solutions, require intervention in the program code or other tools configuration, in order to obtain a macro view of the operations process, when the system is built using distributed architecture concepts. To solve these problems, the present study proposes a structure to capture and analyze logs, metrics and traces, in systems built with Salesforce e AWS. The showing approach enables the monitoring of processes occurring in these environments and integration of transaction data from different systems, thereby creating a macro view of the process flow. Consequently, the results of this work include the capture of execution data from Salesforce operations to facilitate debugging and enable proactive monitoring, as well as the creation of a unified view of transactions occurring in a distributed system. This unified view aids in problem identification and resolution, contributing to the operational efficiency and stability of the monitored systems.

Keywords: System monitoring; Salesforce; AWS; Distributed architectures.

SUMÁRIO

1. INTRODUÇÃO

A Salesforce e a AWS são provedores de nuvem que são utilizados em conjunto ou separadamente para construir soluções diversas à inúmeras áreas de negócio e oferecem benefícios às empresas, como a abstração do gerenciamento de componentes tecnológicos da infraestrutura, agilidade para o desenvolvimento, e, no caso da Salesforce, abstração do gerenciamento do banco de dados e aproximação da empresa ao cliente por meio do fácil acesso ao dado do cliente. Por isso, muitas empresas escolhem construir as soluções para o seu negócio utilizando essas ferramentas.

Porém, o monitoramento dos processos construídos nessas ferramentas, especialmente a Salesforce, tem diversos desafios.

1.1 Problema

Diferentemente do on-premises, em que comumente todas as aplicações eram dispostas em um mesmo ambiente, na nuvem vemos uma transformação para uma arquitetura distribuída. Nesse cenário, para ser executada uma operação de negócio, normalmente diferentes serviços precisam se comunicar sem estar no mesmo ambiente.

Além das aplicações estarem distribuídas, alguns dos provedores de nuvem, como a Salesforce, ferramenta cloud para gestão de relacionamento com o cliente, não oferecem de forma nativa ferramentas para monitorar as operações, ou as ferramentas que estes oferecem não são satisfatórias para que seja possível monitorar as operações de forma eficaz, e, assim sendo, muitas das operações construídas em Salesforce ficam sem monitoramento, o que acarreta em dificuldade para manter as operações em produção e impossibilita a sustentação dos sistemas de forma proativa.

Devido a característica de arquitetura distribuída, torna-se difícil monitorar as operações mesmo quando as aplicações possuem logs e métricas.Quando há um cenário de erro, para encontrar o log exato da execução em que houve falha é necessário identificar todos os ambientes que podem ter executado a operação, até encontrar o dado certo, o que torna o processo de depuração demorado.

Além disso, quando alguma operação ocorre em uma função serverless como, por exemplo, uma lambda function, o processo de monitoramento e depuração torna-se ainda mais complicado. Segundo os autores (Baldini et al.,

2017) no artigo “Serverless Computing: Current Trends and Open Problems”, em tradução livre:

> “Monitorar e depurar aplicações serverless será muito mais desafiador, já que não há acesso direto ao servidor para ver o que aconteceu de errado. Ao invés disso, plataformas serverless precisam reunir todos os dados da execução do código e disponibilizar a posteriori. De forma semelhante, depurar é muito diferente se ao invés de ter um artefato (como um micro serviço ou uma tradicional aplicação monolítica) os desenvolvedores precisam lidar com uma série de peças menores de código.”

1.2 Objetivo

O objetivo geral deste trabalho é mostrar uma estrutura para que seja feito o monitoramento em sistemas que estão construídos em nuvem, mais especificamente, Salesforce e AWS.

Ao definir aqui o conceito de monitoramento enquanto captura de logs, métricas e traces, pretende-se alcançar os seguintes objetivos específicos: demonstração de estrutura para realizar o monitoramento nos ambientes Salesforce e AWS e criação de visão macro de monitoramento das operações em uma arquitetura distribuída.

Para isso, são apresentados dados dois exemplos. O primeiro visa apenas exemplificar como a Salesforce pode ser instrumentada para possibilitar o monitoramento das operações, com captura de logs e métricas. Nesse cenário, o sistema apresenta problemas de execução em uma das operações, a qual será o foco, e, devido à falta de monitoramento, não é possível identificar o erro e corrigi-lo na causa raiz.

Já o segundo exemplo ilustra como realizar a construção do monitoramento de um sistema distribuído, utilizando Salesforce e alguns serviços da AWS. Dessa forma, habilitando que seja feito um monitoramento proativo das operações.

1.3 Justificativa

Monitorar os sistemas é uma tarefa muito importante para garantir a estabilidade e desempenho, já que um monitoramento eficaz possibilita a resposta rápida a incidentes e problemas que foram relatados (Kleppmann, 2017).

Além disso, um bom monitoramento permite que as equipes de TI consigam identificar pontos de melhoria olhando as métricas de desempenho, por exemplo: tempo de resposta, número de requisições processadas e uso dos recursos computacionais. Assim, é possível identificar gargalos potenciais e corrigi-los antes de se tornarem um problema.

Em sistemas distribuídos, o monitoramento é ainda mais importante, por conta da característica da arquitetura, em que os serviços funcionam de forma independente. Com isso, um erro em um componente pode afetar diversos outros componentes, gerando assim um erro em cascata, e a falta de monitoramento torna a situação muito mais complexa de ser resolvida (Newman, 2019).

Por isso, a observabilidade dos sistemas - prática que envolve monitoramento detalhado, com coleta de métricas em tempo real, trace dos fluxos e logs das aplicações - tem sido recomendada como boa prática a se empregar por instituições como a AWS para fornecer aos desenvolvedores e equipes de operação insumo suficiente sobre o comportamento dos sistemas e suas operações, e, assim, facilitar a identificação de anomalias (AWS, 2024). Sem tal prática, a resolução dos problemas pode se tornar algo complexo e demorado.

2. CONCEITOS TEÓRICOS

Para entender de forma aprofundada a temática focalizada neste trabalho, é importante saber alguns conceitos que o contextualizam. A seguir, são apresentadas de forma breve algumas explicações de conceitos-chave para a compreensão e melhor fluidez da leitura.

2.1 Cloud

“Cloud Computing", ou computação em nuvem, em português, é o modelo em que o fornecimento dos recursos de computação acontece sob demanda, como armazenamento, processamento e redes, isto através da internet. Por meio disso, os usuários podem acessar e utilizar serviços de infraestrutura, plataformas e

software de forma flexível e escalável, excluindo a necessidade do gerenciamento das partes físicas envolvidas nesses serviços (Mell e Grance, 2011).

Há três categorias principais de computação em nuvem: IaaS, PaaS e SaaS.

São denominadas IaaS (Infraestrutura como Serviço), quando são fornecidos os recursos básicos de computação, como máquinas virtuais e armazenamento.

Já as PaaS (Plataforma como Serviço), são ambientes para o desenvolvimento dos aplicativos, e a fornecedora do ambiente faz a gestão dos recursos básicos de computação.

Por fim, nas SaaS (Software como Serviço), a fornecedora provê o software já para uso, permitindo apenas parametrizações do mesmo.

2.2 Salesforce

A Salesforce é um CRM - Gestão de Relacionamento com o Cliente - baseado em nuvem, inicialmente projetada para ajudar empresas no gerenciamento de vendas, marketing e atendimento ao cliente, mas que já se expandiu para outras aplicações, como análise de dados e processos de workflow.

Ainda que seja denominada como SaaS, a plataforma possui diversas tecnologias proprietárias de desenvolvimento, que permitem que os desenvolvedores criem aplicativos dentro de sua infraestrutura.

Dividida em módulos, cada um especificado para uma solução, a Salesforce possui clouds de Sales, Service, Marketing, Data, Industries, etc.

2.3 Apex

Apex é uma linguagem de programação proprietária da Salesforce e se assemelha muito à sintaxe do Java. Com essa linguagem de programação, é possível desenvolver a camada de backend para processamento das regras de negócio. Assim como a maioria das linguagens de backend, há suporte para exposição e API, possibilitando chamadas através de outros servidores ou aplicativos, mas também há suporte para que sejam feitas chamadas por meio de outras tecnologias da Salesforce de forma nativa, como: VisualForce, Aura e LWC.

Essa linguagem é fortemente tipada e orientada a objetos, e seu foco está principalmente em operações que envolvem o modelo de dados da Salesforce. Permite que os desenvolvedores criem classes de lógica que interagem diretamente

com os dados armazenados na Salesforce sem a necessidade de uma segunda camada de implementação para acesso aos dados (SALESFORCEb, 2024).

Dentre as suas características, nota-se o fato de sua execução ser controlada, havendo limites de governança para a execução do código. Tais limites têm o objetivo de evitar o consumo excessivo do sistema, o que prejudica as demais operações na Salesforce.

2.4 Serverless

Esse termo, que vem ganhando força dentro do universo de computação, refere-se ao modelo de computação em nuvem em que o provedor gerencia a alocação dos recursos computacionais para a execução da operação de forma automática. Dessa forma, os desenvolvedores não precisam se preocupar com a configuração dos servidores, apenas com a construção dos aplicativos (Baldini et al., 2017).

Apesar do nome serverless em tradução livre para o português significar “sem servidor”, os servidores ainda estão presentes para a execução dos códigos. Porém, como dito, os servidores são totalmente gerenciados pelo provedor e abstraídos para o desenvolvedor, possibilitando um desenvolvimento mais ágil, além de permitir que o desenvolvedor foque apenas na parte da lógica do negócio.

Além disso, normalmente o custo da utilização das funções serverless é comumente calculado pela quantidade de requisições realizadas e o tempo de execução, o que pode tornar-se mais barato do que manter uma máquina virtual sempre disponível consumindo recursos que muitas vezes não estão sendo requeridos a todo momento.

3. INTERVENÇÃO EM UM APLICAÇÃO QUE JÁ ESTÁ SENDO UTILIZADA

A falta de monitoramento nas execuções sistêmicas, como já comentado anteriormente, pode tornar difícil identificar problemas e resolver incidentes produtivos. Sendo assim, em situações que um provedor cloud não fornece de forma nativa ferramentas para realizar o monitoramento, deve-se procurar outras maneiras para a realização de tal tarefa. Este é o caso da Salesforce.

A fim de auxiliar na compreensão da situação focalizada neste trabalho, o problema será apresentado por meio de um exemplo prático.

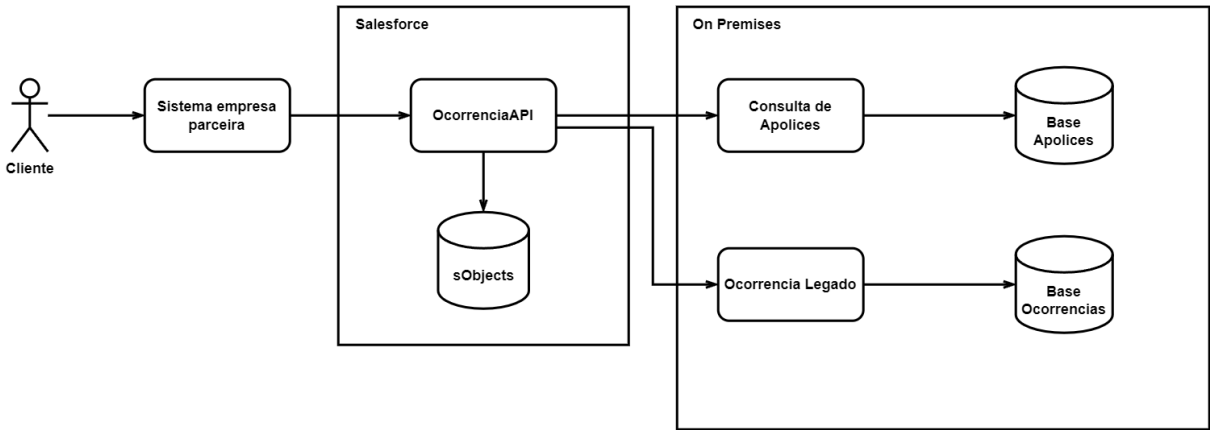
A empresa fictícia “Seguros Para A Minha A Vida” comercializa diversos tipos de seguros em diferentes vertentes.

Para algumas das coberturas comercializadas pela empresa foi tomada a decisão de que a solicitação para a utilização do seguro não será tratada pela seguradora, e, sim, por uma empresa parceira especializada. Nessas situações, o cliente - ao entrar no site da empresa e escolher reportar a necessidade de utilizar o seguro - é direcionado para o portal da empresa parceira e lá informará o ocorrido.

Neste tipo de cenário, em que a solicitação é inteiramente tratada pela empresa parceira, é necessário que o caso aberto também esteja nas bases da empresa “Seguros Para A Minha Vida”, para que seja possível realizar controles de utilização das apólices comercializadas e também a prestação de contas com a empresa parceira.

Para cumprir essa necessidade de negócio, foi desenvolvida uma API para o recebimento dos casos que são abertos na empresa especializada. Como mostra a figura 1, essa API foi construída na Salesforce, ambiente em que também está construído o sistema aviso de ocorrência para utilização do seguro. Os dados recebidos nesse fluxo são armazenados nos objetos da Salesforce, porém, há a necessidade de serem enviados para a base oficial da empresa “Seguros Para A Minha Vida”. O envio dos dados é realizado por meio de um serviço disposto fora da Salesforce.

Figura 1 - Fluxo de abertura de ocorrência integrado com empresa parceira



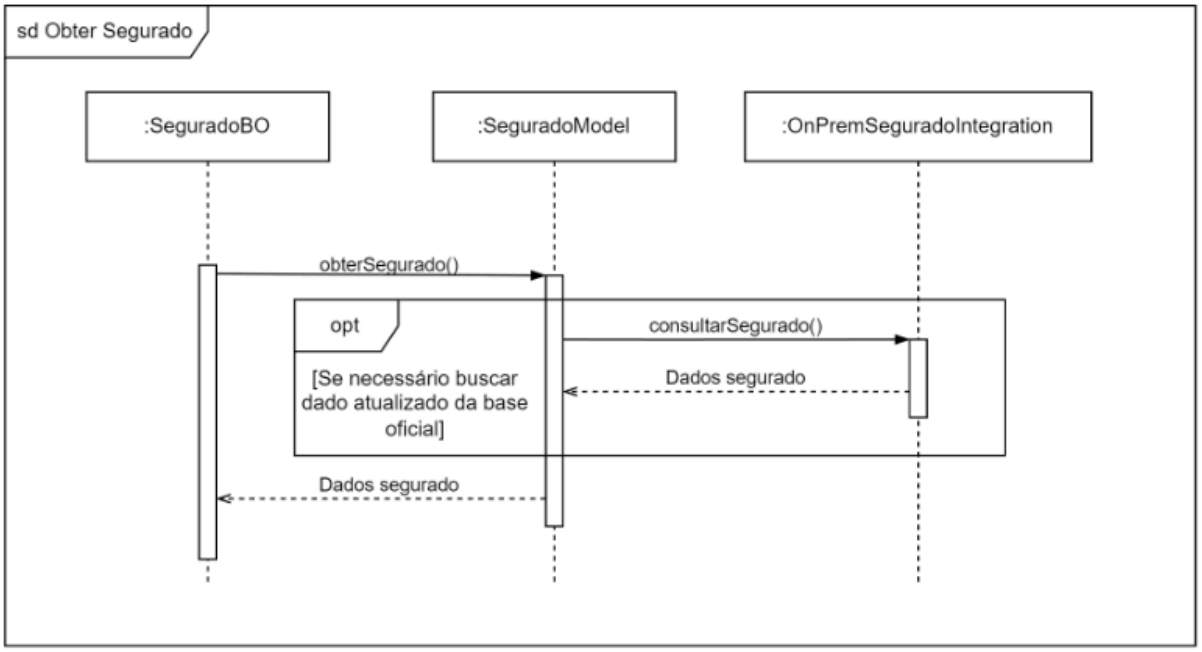
Fonte: Produção do próprio autor (2024).

Após a implantação desse fluxo, o time financeiro da “Seguros Para A Minha Vida” começou a reportar ao time responsável pela API o fato de que nem todas as solicitações cobradas pela empresa parceira estavam presentes na base de dados, e, após algumas investigações, foi descoberto que algumas das solicitações reportadas pela empresa parceira não são processadas corretamente pela API. Já que esta foi construída sem tratar o requisito não funcional de observabilidade, o time responsável não tem informações a respeito da execução da API, como por exemplo: quantas vezes a API foi chamada, quantas execuções foram processadas com sucesso e quantas execuções foram processadas com erro. Além disso, também não há logs de depuração para a identificação e solução dos erros que ocorrem nessa API. Sendo assim, é necessário fazer uma intervenção nesse fluxo para a obtenção desses dados, com o intuito de melhorar a observabilidade dessa API.

3.1 Análise da Situação

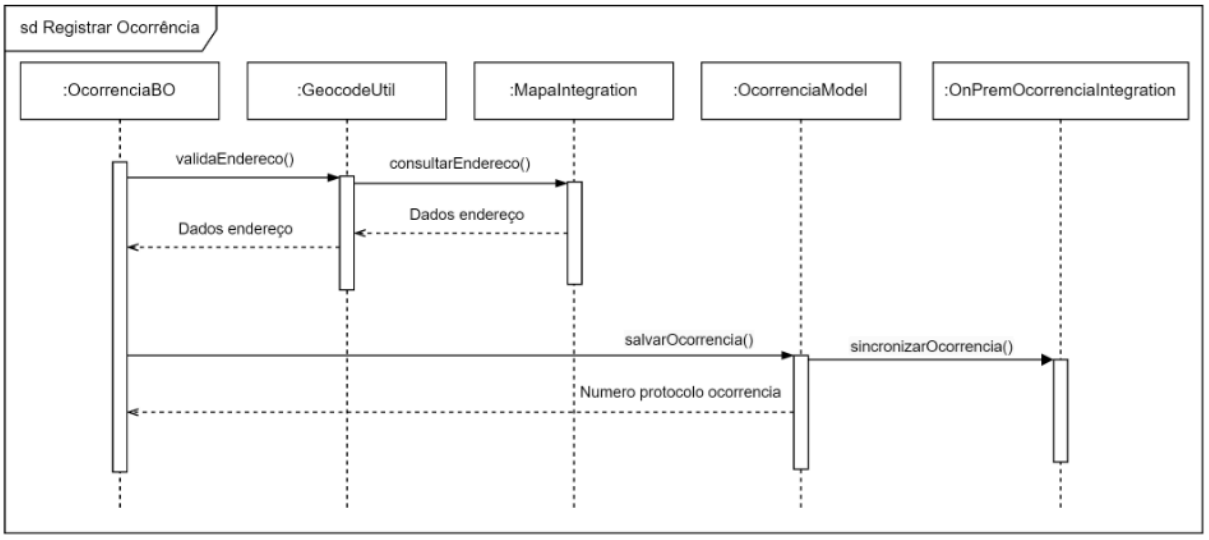
Partindo do pressuposto de que se sabe que a API de ocorrências não tem o funcionamento esperado, e também que não há dados das execuções para análise dos erros da operação, é necessário entender em quais pontos deve ser feita a intervenção, além de entender quais ações a operação está executando, para assim compreender como instrumentar corretamente a API, visando a captura dos logs e métricas.

Figura 2 - Diagrama de sequência de obtenção de segurado



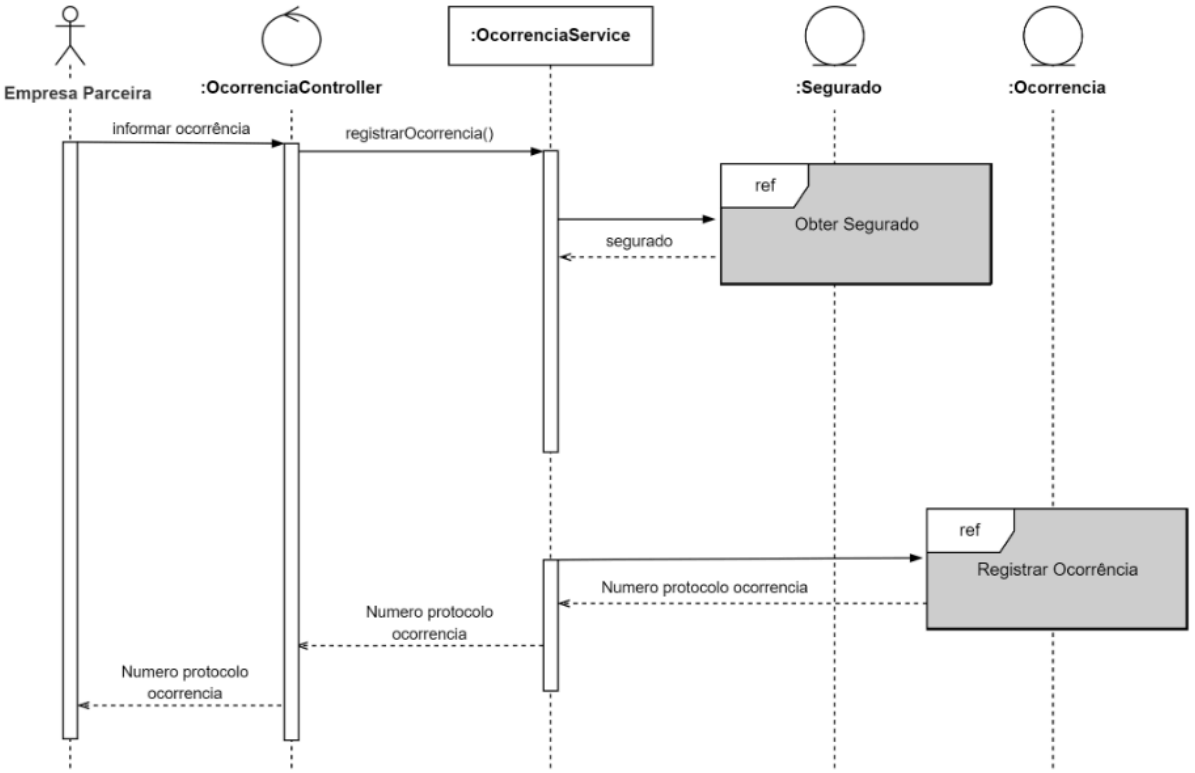
Fonte: Produção do próprio autor (2024).

Figura 3 - Diagrama de sequência de registro de ocorrência



Fonte: Produção do próprio autor (2024).

Figura 4 - Diagrama de sequência de abertura de ocorrência



Fonte: Produção do próprio autor (2024).

A figura 4, ilustra o diagrama de sequência do fluxo dessa API que está com mal funcionamento, assim como as suas demais partes demonstradas nas figuras 3 e 2. Estas demonstram o funcionamento da operação realizada pela API que recebe as ocorrências pela empresa parceira. Nestas imagens, é possível ver a comunicação dentro do sistema, e ter uma visão mais clara dos pontos que devem ser monitorados.

Para citar um exemplo, é possível identificar alguns pontos mais propícios a erros: o ponto de integração para consulta de um segurado na base on premises, a consulta do endereço e a sincronização entre Salesforce e on premises. Neste sentido, autores, como Martin Kleppmann, abordam a fragilidade de sistemas distribuídos, ressaltando que pelo fato de a comunicação ser realizada através da rede, como por exemplo por meio de um http, o resultado dessa comunicação é imprevisível para o requisitante, que está suscetível a uma série de erros (Kleppmann, 2017). Sendo assim, é um ponto válido para se realizar a devida tratativa de erros e o monitoramento.

Outro ponto que pode-se observar é o momento de salvar os dados na Salesforce. O banco de dados é abstraído para o desenvolvedor, representado como objetos, e há diversas possíveis validações que os próprios objetos podem fazer no momento de persistência dos dados. Por exemplo: um campo picklist, que se assemelha a um enum no Java, pode ter seus valores pré definidos e apenas aceitá-los, emitindo um erro caso a aplicação tente criar um registro com um valor que não está definido. Também pode ser um campo de valores pré definidos, mas que aceita outros valores livres. Nesse caso, aceitando a criação do registro. Devido a essa característica da Salesforce, outro ponto frágil da aplicação que pode gerar erros e que ganhamos ao monitorar é a camada de persistência.

Pontos de transformação de dados comumente podem gerar exceções, principalmente se não foram aplicadas técnicas de desenvolvimento que previnem erros, a exemplo do null pointer. Isto posto, apesar de os diagramas não representarem ao certo os pontos que possuem transformações de dados, estes, no geral, também são pontos em que vale a pena existir uma tratativa de erros para captura das execuções e salvamento dos logs, tornando mais fácil realizar a sustentação para identificar os pontos de melhoria.

Por fim, a Salesforce possui diversos limites de execução para códigos Apex, como limite de consumo de memória por execução, tempo limite de execução, quantidade de consultas ao banco, quantidade de callouts, tempo limite para retorno de um callout, entre outros. Exceder algum limite de execução gera a interrupção do processamento. Portanto este pode ser um dos motivos do mau funcionamento dessa API, e deve ser monitorado como os demais pontos.

Após elencar os pontos que devem conter monitoramento, pôde-se passar para uma análise focada em de qual maneira realizar esse monitoramento. Para isso, foi necessário entender como realizar a instrumentação da Salesforce para captura dos dados de execução.

3.2 Proposta de Solução, Instrumentação da Salesforce

A Salesforce provê de forma nativa a captura dos dados de execução das operações no Apex. Porém, analisando a documentação do Debug Log, nos deparamos com diversas limitações que dificultam o monitoramento das transações (SALESFORCEb, 2024).

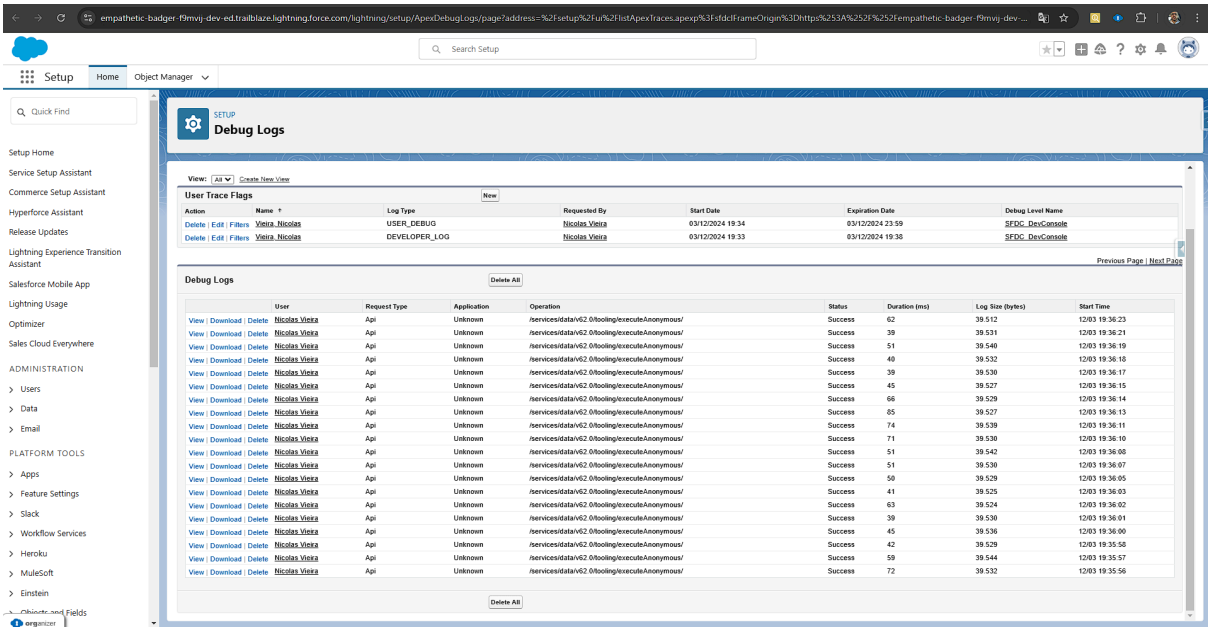
Para a utilização do Debug Log, é necessário fazer a configuração diária para captura de dados de execução de um determinado usuário, o que não limita a captura de dados da API, visto que esta roda em contexto do usuário que o cliente servidor se autenticou, mas demanda que diariamente uma pessoa do time ative a ferramenta para rastrear as execuções do usuário chamando a API (SALESFORCEb, 2024).

Além disso, há um limite de geração de 1 GB de dados em um intervalo de 15 minutos, que, caso atingido, faz com que toda a geração de logs de toda a organização seja desativada. Para o exemplo que estamos utilizando, considerando que a abertura de ocorrências que são tratadas pela empresa “Seguros Para a Minha Vida” é feita na mesma organização em que a API está sendo executada, caso os dois fluxos estejam sendo monitorados, devido a quantidade de usuários utilizando o sistema ao mesmo tempo e o volume de dados de execução gerados ao mesmo tempo, o limite de 1 GB em 15 minutos torna-se um impeditivo para a utilização da ferramenta nativa da Salesforce para a geração de logs.

Ainda que todas as operações sendo executadas na org não gerassem um volume maior de 1 GB no intervalo de 15 minutos, há ainda um limite de 1 GB de geração de logs no total, independentemente do intervalo de tempo para atingir esse volume de dados. Por sua vez, atingir esse segundo limite não desabilita os logs que já foram ligados, porém, impede a configuração para gerar logs em outros pontos do sistema. Diferentemente do que ocorre em outras aplicações e servidores, o rotacionamento dos logs na Salesforce acontece com um intervalo de 24 horas. Por fim, outro ponto que podemos identificar enquanto limitação da ferramenta provida pela Salesforce é a forma de captura e visualização dos dados que foram gerados.

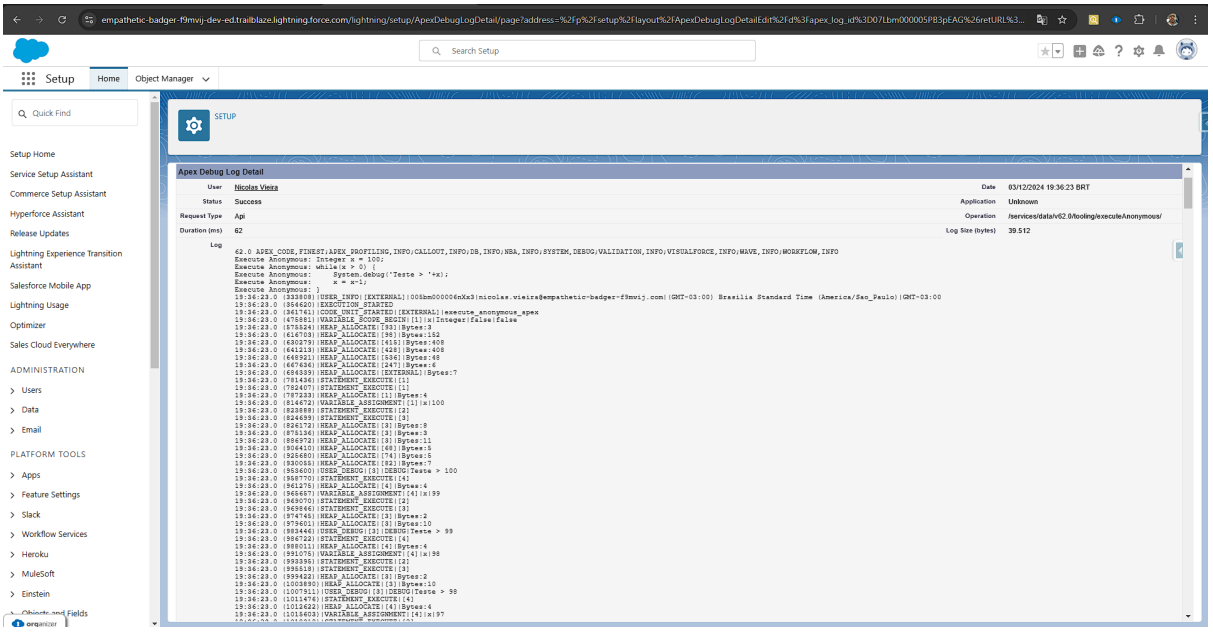
Como mostra as figuras 5 e 6, todos os logs ficam agrupados sem diferenciar a transação que o gerou e não existe uma forma de filtrar a informação por classe ou fluxo. Além do mais, para realizar a busca de um log específico na ferramenta disponibilizada pela Salesforce, é necessário abrir um por um e procurar o trecho desejado para entender o que ocorreu na execução. No cenário de geração de muitos dados, procurar uma informação específica nessa ferramenta se assemelha a encontrar uma agulha no palheiro.

Figura 5 - Tela Debug Logs da Salesforce



Fonte: Produção do próprio autor (2024).

Figura 6 - Tela de visualização de log da Salesforce



Fonte: Produção do próprio autor (2024).

Podemos concluir, então, que a ferramenta padrão da Salesforce não atende à situação que precisamos resolver, e por isso houve a necessidade de buscar outras ferramentas que fazem o trabalho de geração e organização dos dados de

execução. Dentre alguns frameworks desenvolvidos pela comunidade, um dos mais utilizados, e que também foi o escolhido para esse trabalho, é o Nebula Logger.

O Nebula Logger é um framework construído pela comunidade de desenvolvedores da Salesforce, e é uma solução de observabilidade que pode ser instalada na org sem necessidade de pacotes externos além da estrutura do próprio framework (Nebula Logger, 2024).

Quando realizada a instalação do Nebula Framework, são criadas classes, metadados, objetos e eventos de plataforma, para a realização das capturas dos dados, salvamento e personalização da ferramenta.

Esse framework permite salvar os dados de execução de diferentes formas: emitindo eventos, chamando serviços síncronos, adicionando um processo na fila de execução assíncrona e executando DML nos objetos de forma síncrona. Cada método disponível para realizar o salvamento dos dados tem suas próprias características e podem afetar de jeitos diferentes a execução do programa.

Ao analisar especificamente de qual maneira o método para salvar as informações pode impactar o tempo de execução da aplicação, podemos concluir o seguinte: uma operação que apenas recebe a requisição e retorna a string 1, opera com tempo médio de 225,5 milisegundos. Adicionando nessa operação o Nebula Framework e salvando os logs e métricas no modelo padrão da ferramenta, a operação passa a ter tempo médio de execução de 674,84 milisegundos, já alterando a forma de salvamento para queueable, este exemplo passou a executar com o tempo médio de 381,06 milisegundos. Logo, é correto afirmar que, ao olhar especificamente o tempo de execução, o melhor método a se usar é o queueable, em que há acréscimo no tempo de execução, mas ainda consideravelmente menor do que o método comparativo.[1]

---

[1] Utilizando um ambiente da Salesforce criado para fins educativos através da plataforma https://trailhead.salesforce.com/, foi criado uma API como descrito no corpo do texto, os códigos utilizados estão disponíveis nos apêndices A, B e C. O teste foi conduzido utilizando a ferramenta Jmeter. Foram utilizadas 5 threads com ramp-up period de 1 segundo. Cada thread realizou 20 requisições em cada teste, totalizando 100 requisições. Foram executados os testes sem a utilização do Nebula Logger, com a utilização em modo de salvamento por emissão de evento e por queueable. Os dados produzidos pelo teste estão disponíveis no link https://docs.google.com/spreadsheets/d/13hq2QoRZa13whC2x-YjdbpClxQZL5uDlMZpBSeQVhK0/edit?gid=182322353#gid=182322353.

Uma funcionalidade importante do framework é o processo de exclusão dos dados, que é totalmente personalizável para atender a quantidade de tempo que o dado precisa permanecer disponível. Para tal, é fornecido um batch agendável, que exclui todos os registros em que o campo indicador no objeto de log for menor ou igual a data de execução.

Ao definir que o Nebula será utilizado para a captura de logs, ainda há um problema a ser resolvido. Essa ferramenta, ao salvar os logs, consome armazenamento da Salesforce que é destinado aos dados transacionais de negócio.

Sabe-se que o armazenamento de dados na Salesforce é caro, principalmente se comparado a outras ferramentas cloud. Segundo o site da Salesforce (SALESFORCEa, 2024), toda organização dos tipos: Contact Manager, Group, Essentials, Professional, Enterprise, Performance e Unlimited, são criadas com 10 GB disponível para armazenamento de dados, e a cada licença de usuário contratada, são adicionados mais 200 MB para utilização na organização.

Para adição apenas do espaço de armazenamento é necessário negociar junto à Salesforce, sendo que o preço deste não é disponibilizado amplamente pela Salesforce na internet. Porém, sabendo que a forma aconselhada pela plataforma para alocar mais storage é por meio de contratação de novas licenças, é correto afirmar que, utilizando como exemplo uma conta Enterprise, a cada 200 MB adicionais, o custo será de $150 dólares estadunidenses, sendo esse o preço por licença para uma org Service Cloud.

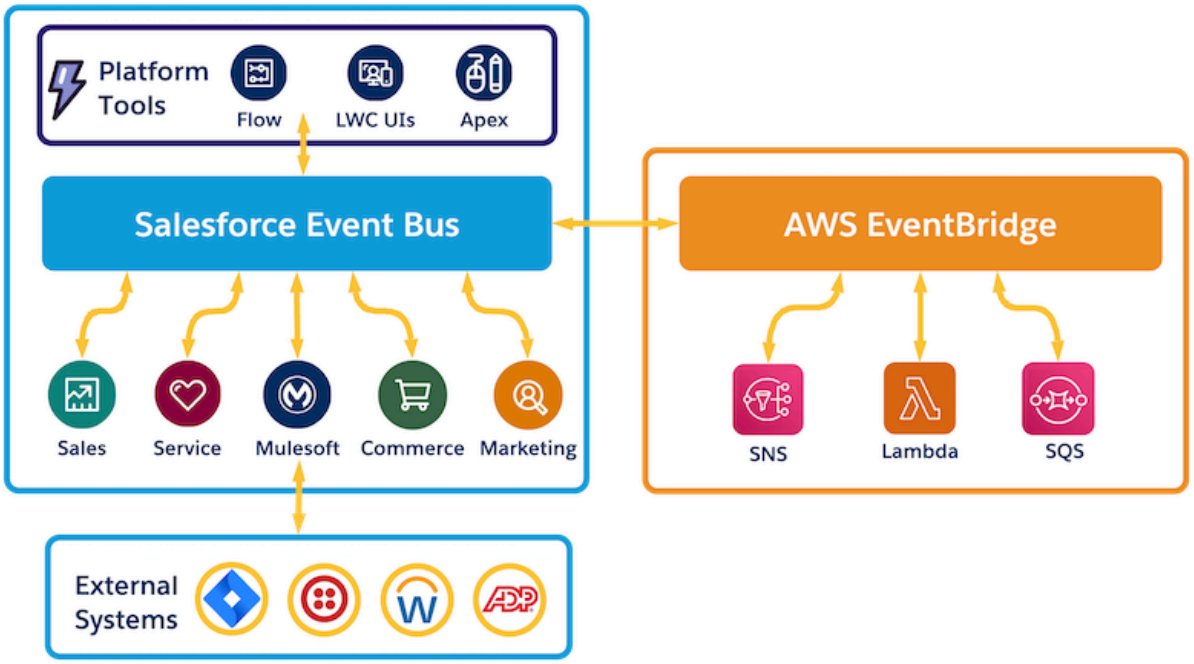
Utilizando como comparação outras ferramentas de logs fornecidas por outras clouds, como Amazon CloudWatch, uma solução em que o custo por GB de armazenamento é equivalente a $750 dólares estadunidenses é inviável. A nível comparativo, 1 GB de dados armazenados no CloudWatch na região ca-central-1 é o equivalente a $0,03 dólares estadunidenses.

Deste modo, por mais que a utilização do Nebula Framework seja útil para a captura dos dados de execução e geração de métricas, os dados produzidos não podem ficar armazenados por longos períodos dentro da Salesforce, sendo necessário transferi-los para outra ferramenta com custo mais barato.

O Nebula Logger, já citado anteriormente, possui uma forma de persistência assíncrona, nesta, além do salvamento dos dados ser feito nos objetos dentro da Salesforce, também é emitido um evento no Platform Events.

A ferramenta de eventos nativa da Salesforce permite que seja feita uma conexão com contas da AWS para a propagação de eventos emitidos do Platform Events para um eventbus - roteador de eventos entre um transmissor e um receptor - dentro da ferramenta Amazon Eventbridge. Esse caminho é configurado através de um Event Relay, uma funcionalidade da Salesforce que se subscreve ao bus da Salesforce e propaga o evento capturado para outro eventbus, conforme configurado, como mostra a figura 7.

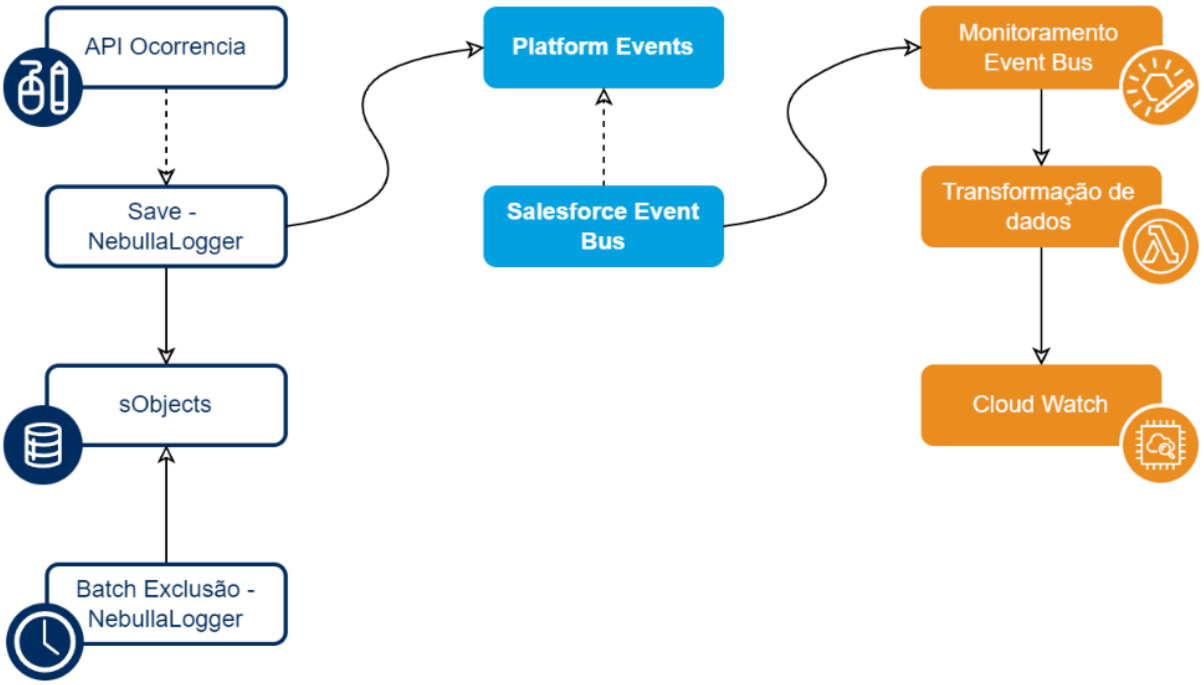
Figura 7 - Fluxo de comunicação entre Platform Events e AWS EventBridge



Fonte: SALESFORCEa (2024).

Por mais que seja possível a criação de uma regra dentro do eventbus para o envio do evento direto para o Amazon CloudWatch - destino dos dados que estão transitando no evento - neste cenário utilizaremos uma lambda para fazer a tratativa do dado que está transitando para fazer a transformação da estrutura salva na Salesforce para uma estrutura de dados mais aderente ao padrão da Amazon CloudWatch, visando ser mais fácil a geração de métricas e visualização dessas informações. Assim, o caminho dos logs de execução da API Ocorrências, até ser armazenado no CloudWatch, ficou como mostrado na figura 8.

Figura 8 - Fluxo transição Salesforce logs para AWS CloudWatch



Fonte: Produção do próprio autor (2024).

A partir da definição de como os dados serão capturados, para onde irão e como será feito o processo de transporte, passou-se para o processo de análise de quais dados deveriam ser capturados. A ferramenta de log que aqui foi utilizada, tem a capacidade de obter diversos dados de performance da execução de uma transação, mas aqui o foco está em apenas alguns dos dados que mais podem demonstrar mais significativamente como estão as execuções desta API, estes dados estão dispostos na tabela 1.

Tabela 1 - Informações da execução capturadas para monitoramento

Nome Lógico	Informação Contida
LimitsHeapSizeUsed__c	Utilização de memória
LoggedByUsername__c	Usuário executor
Message__c	Mensagem impressa através de um log info ou error
LimitsSoqlQueryRowsUsed__c	Quantidade de querys utilizadas

LimitsQueuableJobsUsed__c	Quantidade de trabalhos adicionados à fila de execução
LimitsFutureCallsUsed__c	Quantidade de postagens para execuções futuras
LimitsCpuTimeUsed__c	Quantidade de processamento utilizado na execução
LimitsCalloutsUsed__c	Quantidade de chamadas externas realizadas através de http
StackTrace__c	Rastro do erro
Timestamp__c	Horário da postagem do log
OriginLocation__c	Classe que emitiu o log
APICallResult__c	Resultado da operação que foi executada
APICallType__c	API que foi executada

Fonte: Produção do próprio autor (2024).

3.3 Implementação da Proposta

Para implementar a solução proposta acima, foram necessárias algumas alterações nos códigos do Nebula Framework e também configurações da ferramenta.

A fim de transitar todos os dados da transação em um evento único, foi criado um platform event de nome Log__e, que encapsula tanto o objeto Log__c como todos os objetos LogEntry__c relacionados ao objeto Log__c em questão. Esse evento está sendo emitido após a criação dos objetos, já estando, assim, os objetos relacionados.

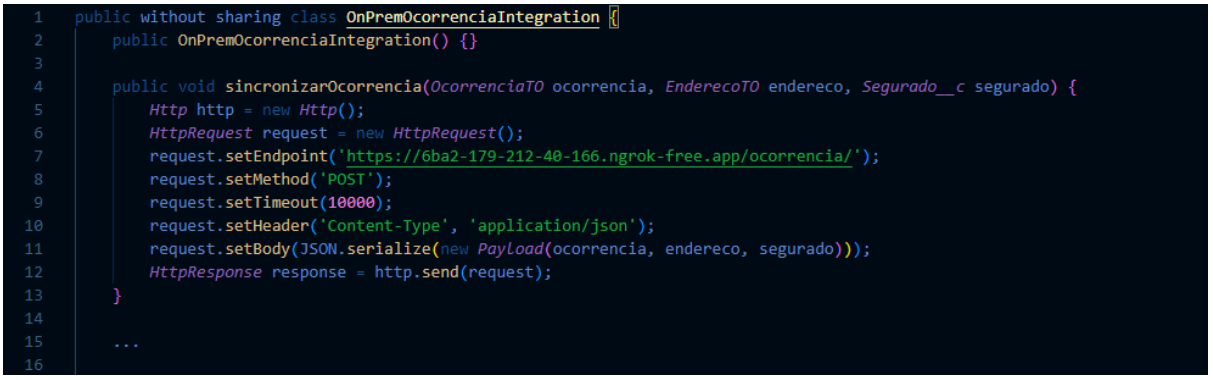
Também foram incluídos dois campos novos dentro do objeto Log__c: APICallType__c e APICallResult__c, em que o primeiro tem por objetivo mapear qual a chamada que foi feita, enquanto o segundo mapeia o resultado da operação, tornando possível que sejam gerados relatórios e gráficos de acompanhamento para verificar a saúde da API que está sendo monitorada.

Após essas personalizações terem sido feitas, foram realizadas as modificações dentro da API, com a implementação de tratativas de erro onde estas não estavam presentes.

Por exemplo: na classe OnPremOcorrenciaIntegration método sincronizarOcorrencia, que tem como responsabilidade sincronizar a ocorrência

recebida na Salesforce com a base oficial da empresa, era feita apenas a requisição sem ocorrer uma tratativa do resultado da requisição, como mostra a figura 9, para que seja possível monitorar essa parte do fluxo a tratativa foi feita por meio da verificação do status de retorno da requisição e lançando um erro caso fosse diferente do status de sucesso, que no caso seria 201, como mostra a figura 10.

Figura 9 - Classe de integração da ocorrência entre sistemas sem tratativa de erro para monitoramento



```
public without sharing class OnPremOcorrenciaIntegration {
    public OnPremOcorrenciaIntegration() {}

    public void sincronizarOcorrencia(OcorrenciaTO ocorrencia, EnderecoTO endereco, Segurado__c segurado) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://6ba2-179-212-40-166.ngrok-free.app/ocorrencia/');
        request.setMethod('POST');
        request.setTimeout(10000);
        request.setHeader('Content-Type', 'application/json');
        request.setBody(JSON.serialize(new Payload(ocorrencia, endereco, segurado)));
        HttpResponse response = http.send(request);
    }

    ...
```

Fonte: Produção do próprio autor (2024).

Figura 10 - Classe de integração da ocorrência entre sistemas com tratativa de erro para monitoramento

```
public without sharing class OnPremOcorrenciaIntegration {
    public OnPremOcorrenciaIntegration() {}

    public void sincronizarOcorrencia(OcorrenciaTO ocorrencia, EnderecoTO endereco, Segurado__c segurado) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://c790-179-212-40-166.ngrok-free.app/ocorrencia/');
        request.setMethod('POST');
        request.setTimeout(10000);
        request.setHeader('Content-Type', 'application/json');
        request.setBody(JSON.serialize(new Payload(ocorrencia, endereco, segurado)));
        HttpResponse response = http.send(request);

        if(response.getStatusCode() != 201) {
            throw new OnPremOcorrenciaIntegrationException('Erro para sincronizar ocorrencia. Status: '+resp
        }
    }
```

Fonte: Produção do próprio autor (2024).

Tal processo foi feito em todos os pontos de integração do sistema, considerando as integrações de busca do segurado e busca de endereço.

Não foi necessário incluir o lançamento de exceções em partes como a transação com o banco de dados, pois, caso tenha algum erro no momento da

execução, um erro já será lançado na pilha de execução. Todo erro lançado na execução de registrar a ocorrência é tratado pelo chamador, sendo esta a classe OcorrenciaController método doPost, o ponto de entrada na API. Neste ponto, foram incluídas as tratativas necessárias para salvar os logs com todos os dados necessários.

Foi incluído um log info registrando o início do processo e este é necessário para os cenários de sucesso também enviarem os dados capturados, visto que a ferramenta apenas faz a criação dos objetos Log__c e LogEntry__c caso exista na lista de logs algum registro dos tipos habilitados na organização.

Utilizando da capacidade de customização do framework, foi utilizado o método setField da classe Logger, para salvar o valor “/ocorrencia/doPost” no campo LogEntryEvent__e.APICallType__c, e no campo LogEntryEvent__e.APICallResult__c resultado da operação, “SUCCESS” ou “FAIL”, significando respectivamente, sucesso ou falha da requisição.

Por fim, quando capturada uma exceção de execução, é salvo o log error, setando a mensagem e o stack trace do erro, informações que são necessárias para a análise dos problemas.

Configurada a integração entre Salesforce e AWS, os dados, a partir do momento em que são salvos, são capturados no evento que foi criado para esse fluxo e passam pela tratativa da lambda implementada, a qual salva as informações no CloudWatch. Como os dados estão no CloudWatch, tornou-se possível realizar análises e construir alertas utilizando as ferramentas.

Por fim, o batch foi configurado na Salesforce para a exclusão dos dados, a fim de não consumir storage, como explicado anteriormente.

4. PLANEJAMENTO E CONSTRUÇÃO DE MONITORAMENTO NO DESENVOLVIMENTO DE UMA APLICAÇÃO DISTRIBUÍDA

A partir do momento que se sabe como realizar a instrumentação da Salesforce para a obtenção dos dados de execução das operações, pode-se voltar o foco para a evolução desse cenário.

Nos cenários em que a Salesforce faz parte do processamento da operação, mas o sistema está distribuído em outros ambientes, ainda há o problema de todos os logs e métricas estarem segregados, e também persiste a complexidade de entender onde exatamente ocorreu o erro.

Para entender melhor a situação, o problema será novamente exemplificado em um contexto prático.

Em um fluxo específico da empresa “Seguros Para a Minha Vida”, na abertura de uma ocorrência de roubo de celular, o sistema recebe as informações imputadas pelo usuário, e precisa notificar uma segunda aplicação que realiza o bloqueio do celular antes de gerar o número da ocorrência, que é gerado automaticamente ao serem salvas as informações. Após salvar as informações com sucesso, é emitido um evento para notificar os sistemas interessados na operação que ocorreu, e, por fim, o número do protocolo é devolvido para o segurado em tela.

Um dos sistemas interessados no processamento ocorrido é o sistema de reserva, que faz os cálculos de quanto será gasto para cobrir o cliente na ocorrência informada, e então faz a separação desse valor em uma conta corrente específica da empresa, atendendo às questões legais impostas pelos órgãos reguladores, como a Susep.

Considerando que esse fluxo está em momento de ideação, há a questão de como realizar o monitoramento de forma a conseguir identificar, quando necessário, onde foi o ponto de falha. Nesse cenário, pode-se utilizar o conceito de trace.

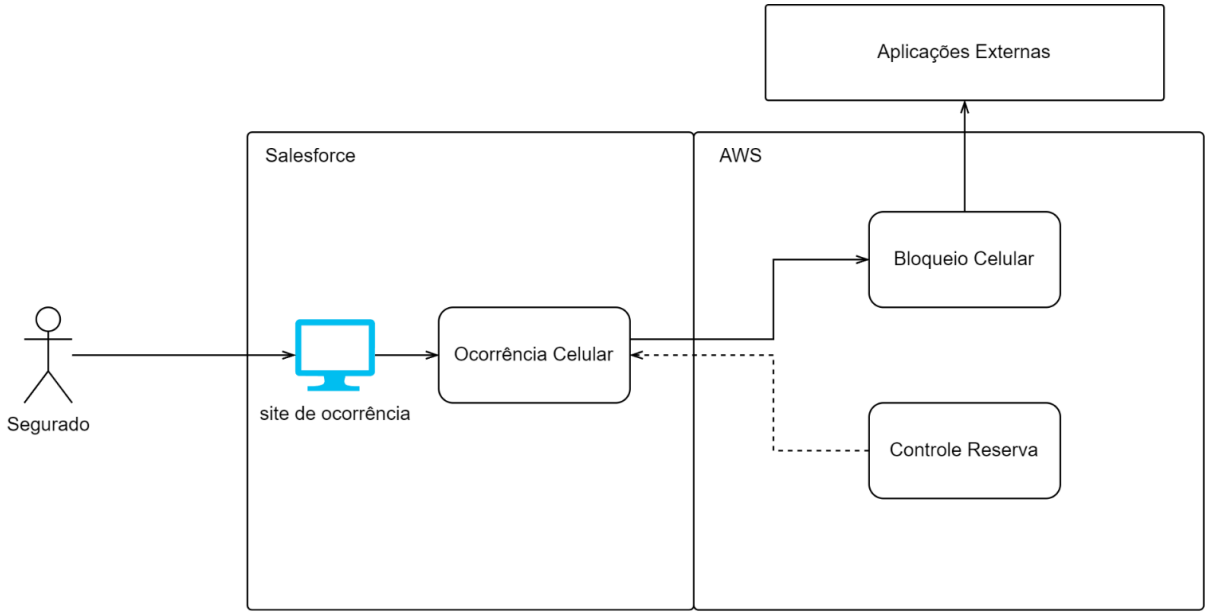
No início deste trabalho o conceito de monitoramento foi definido enquanto captura de logs, métricas e trace, e nesse segundo exemplo o foco será a geração do trace.

4.1 Geração do Trace

Utilizando a definição dada pela Open Telemetry, trace, ou rastros, se traduzido para o português, é “o caminho de uma solicitação através do seu aplicativo” (Open Telemetry, 2024). Dessa forma, pode-se entender por onde a requisição passou, e analisar em caso de falha o lugar exato em que esta ocorreu, além de possibilitar que sejam feitas análises mais aprofundadas para entender problemas de desempenho.

Para a situação mencionada há pouco, como mostra a figura 11, o site que o segurado acessa para informar a ocorrência foi construído na Salesforce, também o processamento dessa ocorrência acontece na Salesforce, já o serviço de bloqueio de celular e o controle da reserva estão dentro da AWS.

Figura 11 - Fluxo de abertura de ocorrência de celular



Fonte: Produção do próprio autor

A Amazon fornece o X-Ray de forma nativa nas suas ferramentas serverless para fazer captura do trace dos fluxos que lá estão sendo executados. Entretanto, apenas o uso do X-Ray não é suficiente nesse cenário. Devido ao fato de que a operação é iniciada na Salesforce, para ter a visão clara de tudo o que foi executado o ideal é o trace mostrar que a execução foi iniciada pela Salesforce no processo de “Ocorrência Celular”, passou pelo “Bloqueio Celular” e por fim o “Controle Reserva” capturou o evento para realizar o seu processamento, como mostra a imagem acima.

Há algumas plataformas que auxiliam os desenvolvedores, realizando o agrupamento das informações de execução que são geradas em ambientes distintos. Para realizar esse trabalho foi utilizada a New Relic, uma plataforma especializada em observabilidade (NEW RELIC, 2024).

Nesta plataforma, há uma ferramenta de monitoramento dos aplicativos chamada APM, que, em resumo, serve para realizar um monitoramento unificado de todos os aplicativos e microsserviços. Dentro dessa ferramenta, uma das funcionalidades é exatamente o trace, que, segundo a documentação da plataforma, fornece o rastro de uma única transação, atendendo exatamente ao objetivo aqui definido (NEW RELIC, 2024). Para o uso dessa funcionalidade, é necessário que o ambiente que executa a operação envie a execução para a New Relic, seguindo as

opções disponibilizadas pela ferramenta, como a instrumentação através do Open Telemetry, ou através de uma chamada http seguindo uma API disponibilizada pela New Relic. Adiante será demonstrado como foram utilizadas as duas formas para o envio da informação.

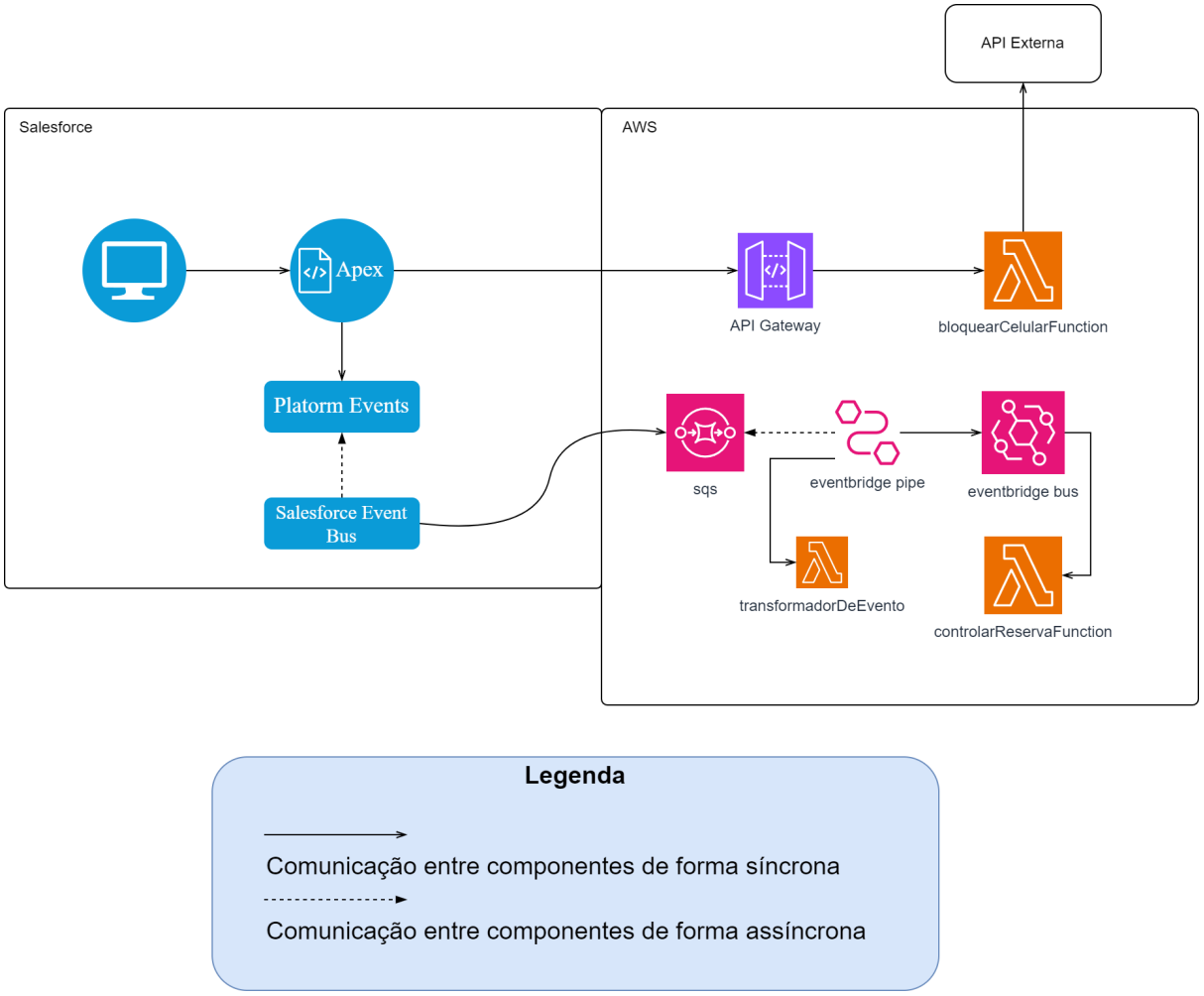
Seguindo o modelo definido pela Open Telemetry, o vínculo das execuções funciona através de hashs identificadores. O trace id, um hash hexadecimal de 32 bits, identifica uma transação enquanto um todo, ou seja, a cada ocorrência informada, deve ser gerado um trace id que representa aquela operação que está sendo realizada.

Para identificar a aplicação que está executando a operação, é utilizado o span id um, hash hexadecimal de 16 bits. Ou seja, nesse exemplo, a Salesforce deve gerar o hash da execução, assim como a aplicação que executa o processo de bloqueio de celular gera o seu próprio hash e também a “Controle Reserva”.

Há ainda o parent span, que é utilizado para ordenar qual aplicação chamou qual durante o processamento da operação. Por exemplo, nesta situação, a Salesforce chama o processo de bloqueio de celular. Então, é correto dizer que o processo de bloqueio de celular foi originado pela Salesforce, sendo um span filho do span gerado na Salesforce.

Ao analisar a figura 12, tem-se uma visão mais aprofundada das tecnologias que estão sendo utilizadas para a execução desse fluxo, e, assim, sabendo quais são as tecnologias utilizadas, foi possível definir como esses códigos serão instrumentados para realizar o envio dos dados para a New Relic.

Figura 12 - Desenho de arquitetura do fluxo de abertura de ocorrência de celular



Fonte: Produção do próprio autor (2024).

É importante salientar que independentemente de como cada aplicação fosse instrumentada, a responsabilidade de gerar o trace id, o valor que identifica toda a transação, ficaria a cargo da execução da Salesforce, ou seja, no código Apex, dado o fato de que o início do processamento se dá nessa parte do fluxo. Essa informação é passada para os demais processamentos seguindo o padrão definido pela W3C, em que são concatenadas as seguintes informações: versão, trace id, span id e trace flag, este último tem a finalidade de indicar que o trace deve estar ativo e ou inativo. Dessa forma, é possível que a aplicação que está recebendo essa informação, continue agregando dados no mesmo trace que já foi iniciado na Salesforce.

A partir do momento que o trace id e o span id foram concatenados seguindo a lógica descrita acima, resultando em algo similar ao seguinte exemplo:

00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01, a informação pode ser transmitida através de um header chamado traceparent. Bibliotecas de instrumentação como a do Open Telemetry são capazes de interceptar essa informação que está sendo transmitida neste header e realizar a configuração para continuar o rastreio no mesmo trace id, bem como vincular o novo span criado com o que foi passado para a aplicação.

Para a instrumentação das lambdas que foram construídas utilizando Python, foram utilizadas justamente as bibliotecas disponibilizadas para lambdas Python da Open Telemetry, que com poucas configurações é possível tornar a instrumentação do código transparente para o desenvolvedor, sem que seja necessário fazer uma linha de código para realizar a criação do span e ao fim do processo fazer o envio do dado para a New Relic. Vale notar que, para o funcionamento desta biblioteca, é necessário que o X-Ray esteja configurado na lambda.

Como a chamada da Salesforce para a lambda bloquearCelularFunction ocorre através de uma chamada http, basta encapsular o trace id e span id dentro do header traceparent, onde o API Gateway da AWS recepciona essa requisição, e nesse cenário trabalha como um proxy, apenas transmitindo header e body para a lambda realizar o seu processamento, como mostra a figura 12.

Já no cenário da comunicação através do evento, foram necessárias mais algumas configurações. Como já comentado anteriormente, é possível fazer uma integração entre Salesforce e AWS através de eventos, cenário descrito no primeiro exemplo desse trabalho, mas, na postagem do evento no Platform Events da Salesforce não é possível colocar a informação do trace id e span id de forma a realizar a instrumentação como feita na lambda bloquearCelularFunction.

Para esse cenário, a solução desenvolvida foi a transmissão do valor encapsulado em um json junto com as outras informações que estão sendo transmitidas no evento. Com a informação dentro da mensagem do evento, ao passar pela lambda de transformação chamada pelo pipe, essa informação e os demais dados do evento são formatados para serem transmitidos para o bus de forma correta.

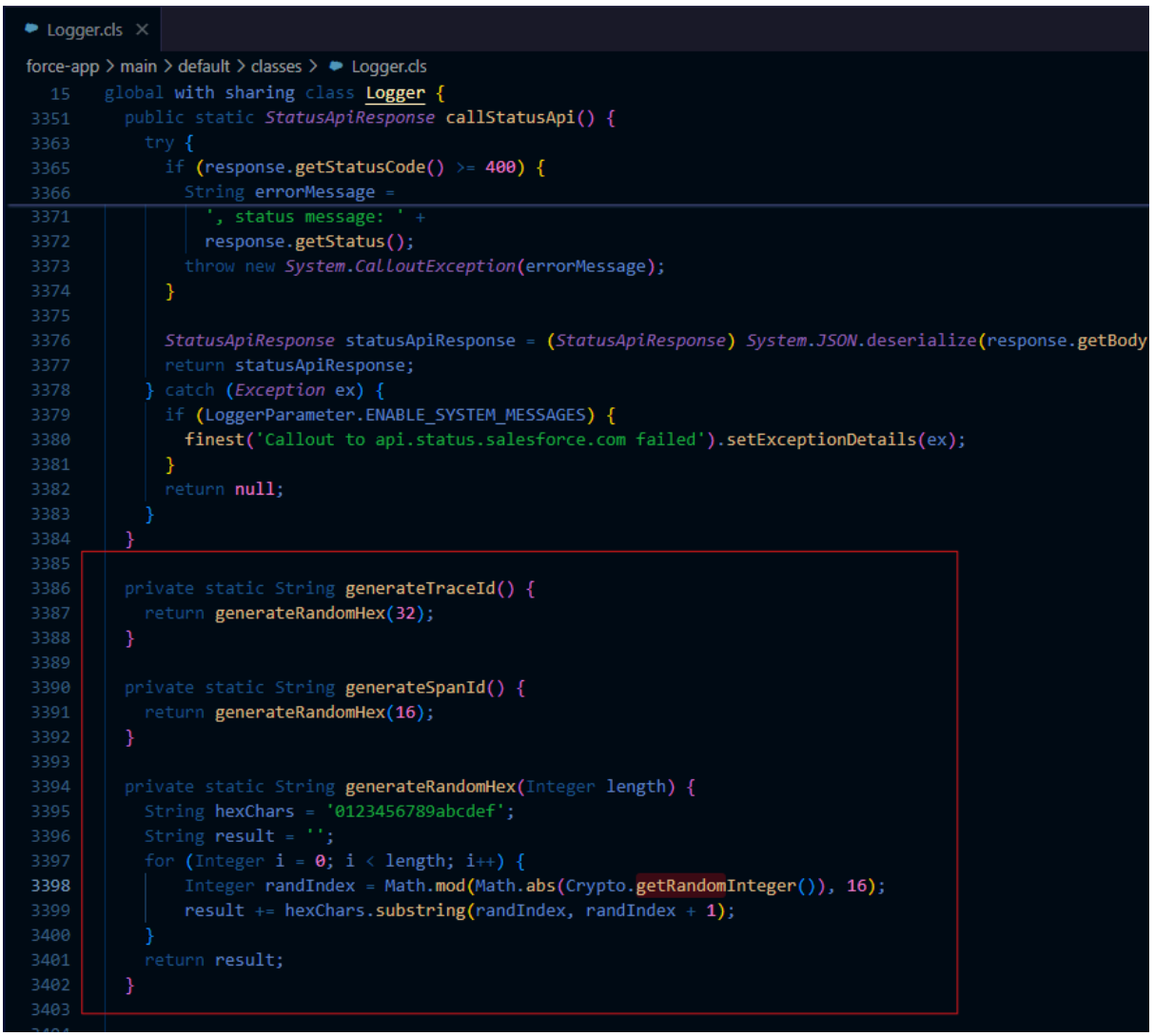
Por fim, sobra a Salesforce. Não há uma ferramenta nativa da Salesforce para fazer a telemetria dos dados, e a biblioteca Nebula Logger não possui suporte para a telemetria das informações no padrão da Open Telemetry como está sendo

utilizado nas lambdas. Por esse motivo, foi necessário realizar a personalização da biblioteca.

Começando pela geração dos trace id e span id, foi desenvolvido o código da figura 13. A Salesforce não tem de forma nativa a geração de códigos randômicos dentro do Apex, e, por isso, há a necessidade de tal código. A sua lógica consiste em gerar de forma aleatória um hash hexadecimal do tamanho ao qual foi transmitido para o método, dessa forma, é possível gerar um hash hexadecimal de 32 bits para o trace id e 16 bits para o span id.

Dentro da lógica de execução, a função getRandomInteger da classe Crypto tem a responsabilidade de gerar um número inteiro randômico. Já o método abs da classe Math tem a função de retornar o número absoluto do número randômico que foi gerado, evitando a chance de erro de execução pelo número gerado ser negativo. Por fim, o método mod da classe Math retorna a sobra de divisão do número gerado por 16, sendo os possíveis valores 0 a 15 e esse valor é utilizado para escolher de forma randômica alguns dos possíveis caracteres hexadecimal. O loop é executado na quantidade de vezes necessárias para gerar o hash do tamanho que foi requerido.

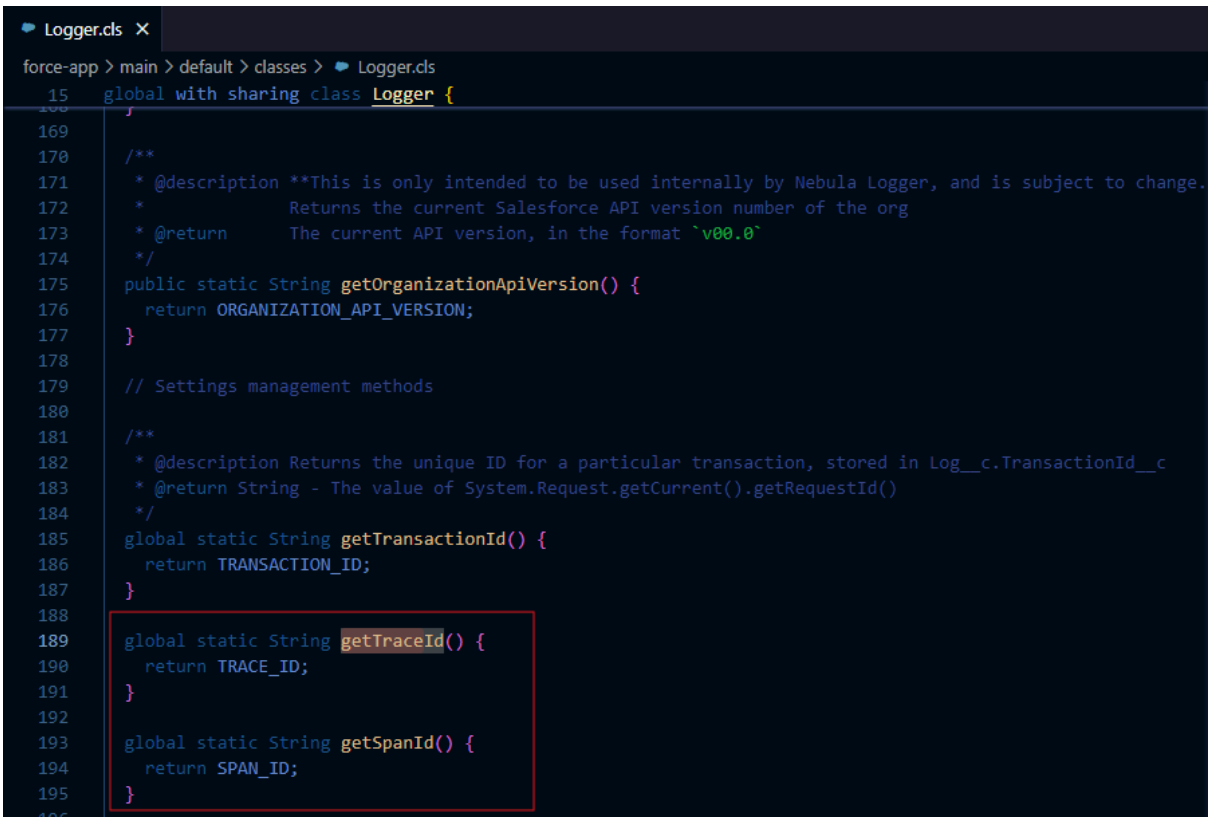
Figura 13 - Alteração na classe Logger, inclusão de métodos para geração de identificadores de trace e span



Fonte: Produção do próprio autor (2024).

Esse código foi colocado estrategicamente dentro da classe Logger que pertence ao framework do Nebula, atribuindo assim a responsabilidade de geração das duas informações necessárias, o trace id e span id, à classe gerenciadora do monitoramento da Salesforce. A partir do momento em que esse objeto é utilizado, é realizada a criação de ambas as informações. Foram construídos dois métodos para obter as informações do trace id e span id, a fim de ser possível obter os valores para construção do header traceparent, como mostra a figura 14.

Figura 14 - Alteração da classe Logger, inclusão de métodos para captura dos identificadores do trace e span



Fonte: Produção do próprio autor (2024).

O envio das informações para a plataforma de observabilidade ocorre apenas no processamento do método saveLog da classe Logger. Novamente aqui neste trabalho foi utilizado o modo de salvamento queueable, que adiciona um processo na fila de execuções da Salesforce. Quando o processo é executado, o salvamento das informações ocorre de forma diferente se comparada ao que foi demonstrado no primeiro exemplo. Foi criada uma classe para ser realizado o salvamento dos dados de forma personalizada para o fluxo aqui em questão, e nessa classe é chamada a API do New Relic para o envio do trace para a plataforma.

Há um detalhe importante que é necessário ressaltar: por mais que o span pai seja gerado na Salesforce, o span filho, que nesse caso é o que foi gerado no processamento da lambda que realiza a lógica para bloquear o celular, é enviado primeiro para a plataforma de observabilidade. Isso acontece porque a biblioteca que instrumenta a lambda para captura das informações e extração para o repositório faz o processo de envio logo após a finalização da execução da lambda.

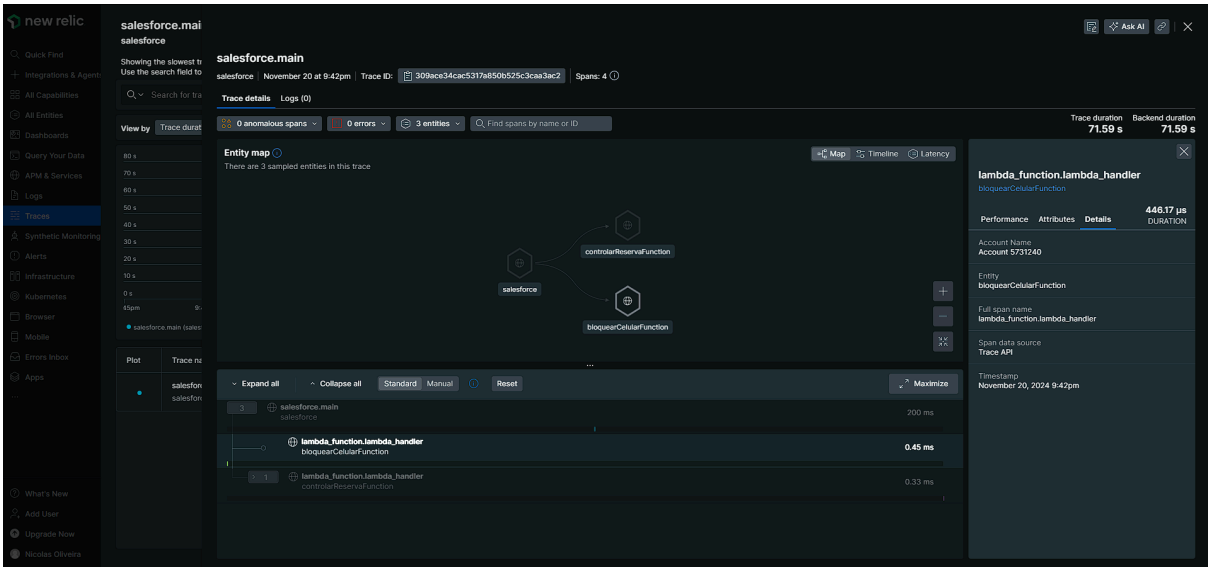
Como o processo de chamada para essa lambda é síncrono, e o envio da Salesforce ocorre apenas no final do processamento, é correto dizer que sempre chegará primeiro o span da lambda. Porém, a ferramenta garante que mesmo chegando em ordens separadas, caso os spans cheguem dentro do intervalo de tempo estipulado pela ferramenta, sempre serão organizados de forma a mostrar a informação correta, isto é, todos ligados à uma única transação e organizados de pai para filho.

Para que fosse feito o envio do trace da Salesforce para o New Relic, como comentado anteriormente, foi utilizado uma API disponibilizada pela New Relic. Dentre as informações que são necessárias passar para essa API, destacam-se o trace id, span id, serviço, host, timestamp. Também é possível enviar alguns dados de métricas, como, por exemplo, a duração da execução do fluxo, dado que é obtido pelo Nebula Logger.

Segundo documentação da New Relic (Docs New Relic, 2024), ao ser recepcionada com sucesso a requisição do envio das informações - o que é indicado por retorno do status 202 Accepted e o campo requestId - o trace enviado é processado de forma assíncrona. Caso haja erro no processamento, este é colocado em uma fila de erros, a qual é possível realizar consultas para verificar o motivo do erro da postagem do trace. Também é possível configurar alertas na ferramenta, para quando houver erros de postagem seja possível notificar a equipe responsável. Dessa forma, torna-se mais fácil sustentar a configuração que foi realizada para captura de trace.

Após a configuração de todas as camadas presentes na figura 12, obteve-se o resultado demonstrado na figura 15, em que uma transação pode ser exibida, mostrando cada aplicação que a transação passou.

Figura 15 - Tela de visualização de trace na plataforma New Relic



Fonte: Produção do próprio autor (2024).

Na imagem, é possível observar também que há outras informações a respeito da execução, como detalhes do ambiente que foi executado, tempo total para a execução, entre algumas outras métricas. Desta forma, pode-se fazer análises para identificar possíveis gargalos e tratá-los antes de tornarem-se problemas.

5. CONCLUSÃO

Durante este trabalho, foram abordados os desafios que os desenvolvedores têm para realizar o monitoramento em sistemas distribuídos em nuvem, destacando a complexidade introduzida por arquiteturas modernas em plataformas como a Salesforce, que, como exemplificado, não possuem formas satisfatórias para atender a esse requisito funcional.

A ausência de monitoramento eficaz, como demonstrado no primeiro exemplo, leva à dificuldade de entender e resolver os problemas existentes em uma aplicação, e, para isso, foi necessário realizar a implementação de uma ferramenta focada em monitoramento para a plataforma em questão.

Além dessa implementação, também foi necessário pensar em uma arquitetura que disponibilizasse o dado para análise feita pelo desenvolvedor, sem ocupar espaço destinado para os dados transacionais, e, assim, não impactar outras partes dos sistemas.

Por fim, foi abordado um exemplo que demonstra como pode ser obtido uma visão macro da execução da transação, e, desta forma, permitir que o desenvolvedor entenda cada aplicação que faz parte da transação.

A proposta desenvolvida neste documento abordou, que mesmo em situações complexas e sem suporte nativo da plataforma, é possível realizar a captura de logs, métricas e traces, viabilizando a observabilidade das operações sistêmicas e contribuindo para o aumento da confiabilidade e desempenho dos sistemas em nuvem.

6. EVOLUÇÕES

Este trabalho não abordou de forma profunda a instrumentação da Salesforce seguindo os padrões definidos pela Open Telemetry para rastreio das execuções sistêmicas. Não foi o foco demonstrar como realizar, por exemplo, o trace de um omniscript que realiza uma chamada remote em um código Apex. Isto posto, não está coberto por esse trabalho o rastreio das execuções da Salesforce com comunicação interna pelos seus diferentes módulos. Com isso, existe a recomendação de evoluir o trabalho nessa direção.

A partir do momento em que esse trabalho demonstrou como obter os dados de execução, é possível realizar análises mais complexas para identificação de possíveis problemas. Com a crescente de inteligências artificiais, seria interessante evoluir o presente trabalho ao abordar o tema da realização do monitoramento com o auxílio de inteligências artificiais.

REFERÊNCIAS[2]

AWS. **Operational excellence: prepare**. Disponível em: https://docs.aws.amazon.com/en_us/wellarchitected/2022-03-31/framework/oe-prepare.html. Acesso em: 19 nov. 2024.

BALDINI, I.; et al. **Serverless computing: current trends and open problems**. Disponível em: https://arxiv.org/abs/1706.03178. Acesso em: 20 nov. 2024.

KLEPPMANN, M. **Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems**. O'Reilly Media, 2017.

MELL, P.; GRANCE, T. **The NIST definition of cloud computing**. National Institute of Standards and Technology, 2011. Disponível em: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Acesso em: 20 nov. 2024.

NEBULA LOGGER. **Documentação oficial.** Disponível em: https://github.com/jongpie/NebulaLogger. Acesso em: 27 nov. 2024.

NEW RELIC. **Documentação oficial**. Disponível em: https://docs.newrelic.com/pt/. Acesso em: 20 nov. 2024.

NEWMAN, S. **Building Microservices: Designing Fine-Grained Systems**. O'Reilly Media, 2019. Capítulo 8: Monitoring.

OPEN TELEMETRY. **Documentação oficial**. Disponível em: https://opentelemetry.io/pt/. Acesso em: 19 nov. 2024.

SALESFORCEa. **Central de ajuda**. Disponível em: https://help.salesforce.com/. Acesso em: 19 nov. 2024.

SALESFORCEb. **Developer documentation**. Disponível em: https://developer.salesforce.com. Acesso em: 18 nov. 2024.

[2] De acordo com a Associação Brasileira de Normas Técnicas (ABNT NBR 10520).

APÊNDICE A - CÓDIGO DESEMPENHO NEBULA LOGGER SEM SAVE LOG

```
@RestResource(urlMapping='/ocorrencia/*')
global without sharing class OcorrenciaController {
    private static RestResponse restResponse;
    private static RestRequest restRequest;
    @HttpGet
    global static String checkStatus() {
        return '1';
    }
}
```

APÊNDICE B - CÓDIGO DESEMPENHO NEBULA LOGGER COM SAVE LOG EVENT BUS

```
@RestResource(urlMapping='/ocorrencia/*')
global without sharing class OcorrenciaController {
    private static RestResponse restResponse;
    private static RestRequest restRequest;
    @HttpGet
    global static String checkStatus() {
        Logger.info('TESTE');
        Logger.saveLog();
        return '1';
    }
```

```
}
```

APÊNDICE C - CÓDIGO DESEMPENHO NEBULA LOGGER COM SAVE LOG QUEUEABLE

```
@RestResource(urlMapping='/ocorrencia/*')
global without sharing class OcorrenciaController {
    private static RestResponse restResponse;
    private static RestRequest restRequest;
    @HttpGet
    global static String checkStatus() {
        Logger.info('TESTE');
        Logger.saveLog(Logger.SaveMethod.QUEUEABLE);
        return '1';
    }
}
```