UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Raisson Adrian Grangeiro Souto

CONTROLE DE ACESSO UTILIZANDO SPIRE E MONITORAMENTO DE UM CLUSTER KAFKA NO AMBIENTE DO *SMARTCAMPUS* NA UFCG

CAMPINA GRANDE - PB

2024

Raisson Adrian Grangeiro Souto

Controle de Acesso Utilizando SPIRE e Monitoramento de um Cluster Kafka no Ambiente do *SmartCampus* na UFCG

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Andrey Elísio Monteiro Brito

CAMPINA GRANDE - PB

2024

Raisson Adrian Grangeiro Souto

CONTROLE DE ACESSO UTILIZANDO SPIRE E MONITORAMENTO DE UM CLUSTER KAFKA NO AMBIENTE DO *SMARTCAMPUS* NA UFCG

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Andrey Elísio Monteiro Brito
Orientador – UASC/CEEI/UFCG

Reinaldo Cezar de Morais Gomes
Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro
Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 17 de maio de 2024.

CAMPINA GRANDE - PB

RESUMO

No contexto do projeto do *SmartCampus*, da Universidade Federal de Campina Grande (UFCG), a disponibilização de um Kafka contendo informações de consumo de energia apresenta desafios complexos relacionados à segurança e controle de acesso. Diversas entidades, incluindo desenvolvedores, usuários finais e operadores dos sistemas de produção, necessitam interagir com esse Kafka a partir de ambientes variados, criando a necessidade de diferenciar e controlar o acesso de maneira eficaz. Este trabalho explora estratégias fundamentadas no modelo *Zero Trust*, que preconiza a autenticação contínua e autorização granular, garantindo que cada acesso seja verificado e autenticado. Adotando o SPIRE em conjunto com uma série de outros microsserviços, busca-se assegurar a autenticação segura por meio de identidades SPIFFE, bem como configurar um serviço de autorização personalizado de acordo com o perfil do usuário. O objetivo é prevenir acesso inadequado, vazamento de dados e manipulações indevidas, visando obter um ambiente seguro e confiável para a implantação do projeto. Por fim, também é desejado obter métricas para monitoramento das ferramentas utilizadas, a fim de identificar anomalias e falhas rapidamente.

Access Control Using SPIRE and Monitoring of a Kafka Cluster in the SmartCampus Environment at UFCG

ABSTRACT

In the context of the SmartCampus project at the Federal University of Campina Grande (UFCG), the provision of a Kafka containing energy consumption information presents complex challenges related to security and access control. Various entities, including developers, end-users, and production system operators, need to interact with this Kafka from various environments, creating the need to differentiate and control access effectively. This work explores strategies based on the Zero Trust model, which advocates for continuous authentication and granular authorization, ensuring that each access is verified and authenticated. By adopting SPIRE along with a series of other microservices, the aim is to ensure secure authentication through SPIFFE identities, as well as configure a personalized authorization service according to the user's profile. The goal is to prevent inappropriate access, data leakage, and improper manipulations, aiming to achieve a secure and reliable environment for the project deployment. Finally, obtaining metrics for monitoring the tools used is also desired to quickly identify anomalies and failures.

Controle de Acesso e Monitoramento de um Cluster Kafka Utilizando SPIRE no Ambiente do *SmartCampus* na UFCG

Raisson Adrian Grangeiro Souto
Universidade Federal de Campina Grande - UFCG
Campina Grande, Brasil
raisson.souto@ccc.ufcg.edu.br

Andrey Brito
Universidade Federal de Campina Grande - UFCG
Campina Grande, Brasil
andrey@computacao.ufcg.edu.br

RESUMO

No contexto do projeto do *SmartCampus*, da Universidade Federal de Campina Grande (UFCG), a disponibilização de um Kafka contendo informações de consumo de energia apresenta desafios complexos relacionados à segurança e controle de acesso. Diversas entidades, incluindo desenvolvedores, usuários finais e operadores dos sistemas de produção, necessitam interagir com esse Kafka a partir de ambientes variados, criando a necessidade de diferenciar e controlar o acesso de maneira eficaz. Este trabalho explora estratégias fundamentadas no modelo *Zero Trust*, que preconiza a autenticação contínua e autorização granular, garantindo que cada acesso seja verificado e autenticado. Adotando o SPIRE em conjunto com uma série de outros microsserviços, busca-se assegurar a autenticação segura por meio de identidades SPIFFE, bem como configurar um serviço de autorização personalizado de acordo com o perfil do usuário. O objetivo é prevenir acesso inadequado, vazamento de dados e manipulações indevidas, visando obter um ambiente seguro e confiável para a implantação do projeto. Por fim, também é desejado obter métricas para monitoramento das ferramentas utilizadas, a fim de identificar anomalias e falhas rapidamente.

KEYWORDS

Kafka, *SPIRE*, *Controle de Acesso*, *Segurança*, *Monitoramento*

1 INTRODUÇÃO

A Universidade Federal de Campina Grande (UFCG)[1] está implantando um sistema de *SmartCampus*[2], tendo como objetivo primário desenvolver um ambiente que fomente a pesquisa e a inovação através de estratégias aplicadas à gestão universitária. Em sua primeira fase, o projeto prevê a implantação de serviços básicos para disponibilizar o acesso aos dados do sistema de controle acadêmico[3] e de consumo de energia em algumas edificações, como salas de aula e laboratórios nos campi da UFCG. Essas aplicações serão inicialmente disponibilizadas para projetos escolhidos por meio de um processo seletivo. Dos serviços propostos, aqueles relacionados ao consumo de energia serão oferecidos pelo LiteMe[4], projeto de um dos laboratórios da universidade, e podem ser divididos em três categorias, conforme ilustrado na Figura 1, sendo estas:

(1) Atuação em sistemas de IoT (Internet das Coisas) presentes nos campi;
(2) Solicitações de dados armazenados coletados pelos sensores;
(3) Acesso a dados coletados pelos sensores em tempo real através de um cluster Kafka[7].

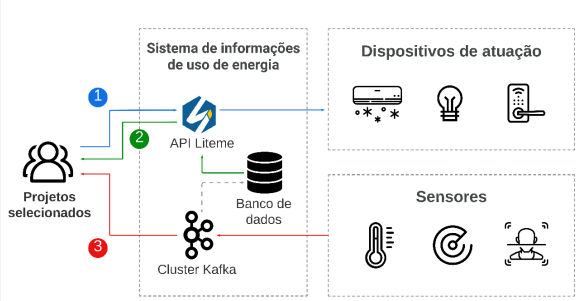

Figura 1: Diagrama de casos de uso das APIs de consumo de energia.

Essas operações serão disponibilizadas através de APIs (interface de programação de aplicação) para uma variedade de usuários, incluindo desenvolvedores e operadores de sistemas em produção, em diversos cenários. Com isso, surgem diversos desafios relacionados à segurança dos dados. Esses desafios incluem a importância de diferentes níveis de acesso para diferentes usuários, a prevenção de vazamento de dados, além da proteção contra manipulações indevidas em serviços de IoT. O monitoramento eficaz também desempenha um papel crucial na identificação e resposta rápida a possíveis incidentes, colaborando na garantia da integridade e confidencialidade dos dados. Nesse trabalho, o foco reside no controle de acesso do caso de uso 3 e na implantação dos microsserviços necessários para tal.

As soluções convencionais de controle de acesso no Kafka enfrentam desafios significativos em ambientes tecnológicos modernos, como o almejado pelo *SmartCampus* da UFCG. Práticas como o compartilhamento de credenciais e a criação manual de identidades para cada usuário são condutas desatualizadas e inadequadas, especialmente diante do aumento do uso de dispositivos IoT. Essas abordagens também enfrentam dificuldades de escalabilidade, tornando-se cada vez mais ineficazes à medida que o número de usuários cresce, resultando em um aumento exponencial das demandas de gerenciamento. Operar estas soluções se revela uma atividade complexa, requerendo uma sobrecarga significativa de

[1]Site da UFCG: https://portal.ufcg.edu.br/
[2]Site do sistema de informações de uso de energia: https://ufcg.liteme.com.br/
[3]Site do controle acadêmico: https://pre.ufcg.edu.br:8443/ControleAcademicoOnline/
[4]Site do LiteMe: https://www.liteme.com.br/

recursos e esforços administrativos. Portanto, é evidente que as abordagens tradicionais não são mais suficientes para atender às demandas de segurança, eficiência e conformidade em ambientes tecnológicos contemporâneos.

Com o objetivo de atender aos requisitos de segurança do *SmartCampus* da UFCG, é apresentada a implantação de um *cluster* Kubernetes[15] entre os clientes e o sistema de informações de uso de energia. Esse cluster terá um segundo Kafka, que replica o já existente no LiteMe, e outros serviços para roteamento de requisições, emissão de identidades confiáveis e monitoramento. A principal vantagem dessa abordagem é a capacidade elevar a segurança sem modificar a arquitetura atual do LiteMe. Por exemplo, ao adicionar um Kafka replicado, protegemos o Kafka original contra ataques DoS (negação de serviço) e escalada de privilégios.

A solução também inclui a disponibilização de uma máquina virtual na nuvem privada do laboratório, onde os projetos selecionados poderão implantar suas aplicações. O propósito dessa abordagem é estabelecer um ambiente confiável, permitindo a execução de um agente SPIRE[21]. Dessa forma, em vez de solicitar credenciais ao operador, os clientes poderão desenvolver suas aplicações e utilizar o SPIFFE *helper*[19], uma ferramenta que simplifica a integração com o SPIRE, para lidar com as requisições de certificados.

Este trabalho segue a seguinte estrutura: Primeiramente, na Seção 2, é apresentada a Fundamentação Teórica, onde são discutidos os conceitos necessários para compreender o contexto do problema. Em seguida, na Seção 3, é delineado o Modelo de Ameaça, que identifica e analisa os potenciais riscos e vulnerabilidades. Posteriormente, a Seção 4 descreve a abordagem proposta para mitigar os riscos identificados. Em seguida, na Seção 5, a Análise de Segurança examina a eficácia das soluções implementadas em relação aos requisitos de segurança apresentados na seção 3. Na Seção de Avaliação, no Capítulo 6, são apresentados os resultados da avaliação prática do sistema. Com base nos resultados obtidos, são apresentadas as Conclusões, no Capítulo 7, que destacam os pontos fortes e as limitações da abordagem proposta.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta Seção são introduzidos alguns conceitos e tecnologias importantes para entender o contexto do trabalho apresentado.

2.1 Apache Kafka

O Kafka é uma plataforma para transmissão de dados distribuída amplamente adotada na indústria para a construção de *pipelines* de dados em tempo real. Sua arquitetura robusta e altamente escalável torna-o uma escolha popular para lidar com os desafios de processamento, armazenamento e transmissão de grandes volumes de dados de forma contínua e desacoplada.

Sua capacidade de lidar com um grande número de produtores e consumidores simultaneamente torna-o ideal para cenários onde a demanda por dados em tempo real é alta. O Kafka possui uma robusta e flexível configuração de segurança, permitindo múltiplas formas de autenticação e a criação de ACLs, entre outras funcionalidades.

No contexto do LiteMe, o Apache Kafka desempenha um papel fundamental ao fornecer uma infraestrutura confiável para receber e distribuir dados de consumo de energia em tempo real. Esses dados são coletados por sensores distribuídos pelo o campus, permitindo que os consumidores realizem a coleta e análise dos dados para melhorar a eficiência energética no campus.

2.2 SPIFFE

O aumento na distribuição de sistemas tem gerado um desafio significativo em relação à escalabilidade da autenticação entre serviços. Para abordar essa questão, o *framework* SPIFFE[20] (*Secure Production Identity Framework for Everyone*) foi desenvolvido como um conjunto de padrões de código aberto para identificar com segurança sistemas de software em ambientes dinâmicos e heterogêneos. Sistemas que adotam o SPIFFE podem se autenticar mutuamente facilmente e de forma confiável onde quer que estejam em execução.

2.3 SPIRE

O SPIRE é a implementação das APIs do SPIFFE, a ferramenta realiza a atestação de nós e cargas de trabalho para emitir identidades na forma SVIDs (*SPIFFE Verifiable Identity Document*) para essas cargas de trabalho. Para realizar a atestação de cargas de trabalho, a carga deve ser registrada no SPIRE *server* com base em um conjunto predefinido de condições e quando a carga solicita seu SVID, um agente do SPIRE verifica se ela atende aos critérios especificados, se sim, a identidade é entregue.

2.4 Envoy

O Envoy[10] é um *proxy* frequentemente utilizado em arquiteturas de microsserviços para facilitar o roteamento de tráfego entre serviços, gerenciamento de carga, segurança e observabilidade. O Envoy também oferece suporte para adicionar mTLS (*Mutual Transport Layer Security*) à comunicação, garantindo a criptografia dos dados em trânsito e autenticação mútua.

2.5 Graphite

Graphite[8] é uma ferramenta de monitoramento, especializada em armazenar e visualizar séries temporais de dados. Ao armazenar e visualizar dados em séries temporais, o Graphite permite análise e a identificação de tendências ao longo do tempo. Além disso, o Graphite adota um modelo de coleta de dados conhecido como *push-based*. Nesse modelo, os serviços que desejam ser monitorados enviam ativamente suas métricas para o Graphite em intervalos regulares.

2.6 Prometheus

O Prometheus[13] é uma aplicação utilizada para monitoramento de eventos e alertas em sistemas distribuídos. Ele registra métricas em tempo real em um banco de dados de séries temporais (permitindo alta dimensionalidade) e sua arquitetura se baseia em um modelo de coleta de dados *pull-based*. Nesse modelo, o servidor Prometheus faz solicitações HTTP regulares para os serviços que estão sendo monitorados. Essas solicitações são direcionadas para *endpoints* específicos nos serviços, os quais expõem as métricas no formato adequado para o Prometheus entender e coletar.

2.7 Grafana

O Grafana[11] é uma plataforma de análise e visualização de dados projetada para atender às necessidades de monitoramento e análise

de sistemas complexos e distribuídos. Através de painéis personalizáveis e configuráveis, os administradores podem acompanhar o desempenho do sistema em tempo real, identificar tendências e anomalias, e tomar decisões informadas para otimizar o uso de recursos e garantir a eficiência operacional. Para isso, o Grafana oferece suporte para a integração com uma ampla variedade de fontes de dados, incluindo bancos de dados relacionais e sistemas de monitoramento como o Graphite e o Prometheus.

2.8 JMX Exporter

O Kafka não oferece nativamente a exposição de dados de telemetria, sendo então necessário utilizar sistemas auxiliares, como o JMX (*Java Management Extensions*) Exporter[14]. O JMX é uma ferramenta utilizada para exportar métricas de aplicações Java, no caso, sobre o *broker* Kafka ao qual está associado, como volume de requisições, operações no disco, etc. Essas métricas são expostas num *endpoint* acessível para ferramentas de monitoramento e análise, como o Prometheus, permitindo uma melhor compreensão do desempenho e da saúde do cluster Kafka.

2.9 Kafka Exporter

Similar ao JMX Exporter, o Kafka Exporter[9] é uma ferramenta utilizada para coletar métricas do Kafka, expondo-as para para ferramentas de monitoramento. No entanto, ele fornece métricas em mais alto nível, como detalhes sobre os líderes de tópicos e a disponibilidade dos *brokers*. Esta ferramenta complementa o JMX Exporter, oferecendo uma visão mais abrangente do funcionamento do Kafka.

2.10 SPIFFE Helper

O SPIFFE *helper* é uma ferramenta que facilita a integração do SPIRE em aplicações que não foram projetadas para lidar com a troca de certificados proposta pela ferramenta. Ele fornece integrações prontas para uso e abstrai a complexidade subjacente, permitindo que os usuários se concentrem em desenvolver e implantar seus aplicativos, sem se preocupar com os detalhes de implementação do SPIFFE.

2.11 Kubernetes

Kubernetes é uma plataforma de orquestração de contêineres de código aberto que automatiza a implantação, o dimensionamento e a gestão de aplicativos em contêineres. Com Kubernetes, os aplicativos são implantados em um cluster, composto por nós de trabalho, onde cada nó é uma máquina virtual ou física. Isso permite que os aplicativos sejam executados de forma eficiente e escalável. Ele oferece recursos avançados para implantação e escalabilidade de aplicativos, permitindo que as equipes de desenvolvimento implementem e dimensionem facilmente novos serviços conforme necessário.

3 MODELO DE AMEAÇA

O modelo de ameaça proposto considera que o atacante pode assumir diferentes formas e motivações. Ele pode ser uma entidade legítima, como um projeto autorizado, tentando obter dados para os quais não tem permissão, ou um terceiro mal-intencionado sem credenciais válidas. As ações maliciosas que um atacante pode empreender incluem tentativas de acesso não autorizado aos tópicos do Kafka, manipulação de mensagens nos tópicos, escalada de privilégios e ataques que visam comprometer a disponibilidade do sistema.

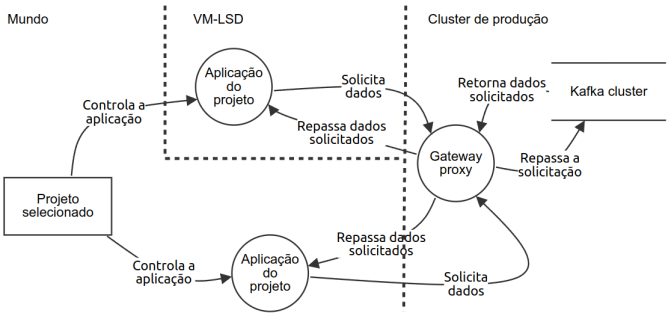

Figura 2: Diagrama de fluxo de dados.

Com objetivo de entender melhor o sistema e os potenciais pontos de ataque, foi feita uma modelagem de ameaça utilizando STRIDE[18], uma metodologia de análise de ameaças de segurança da informação. A análise considerou os projetos selecionados como atores, esses atores podem executar processos em duas fronteira de confiança: "VM-LSD", que corresponde à máquina virtual disponibilizada para os atores colocarem suas aplicações, e "mundo", que se refere a qualquer outro contexto para um processo executar. Além disso, dentro do limite de confiança do cluster de produção, um processo denominado "gateway-proxy"age como intermediário do Kafka, que na análise foi classificado como armazenamento, conforme mostrado na Figura 2.

Além de entender as entidades, é preciso entender as comunicações entre elas, especialmente aquelas que atravessam fronteiras de confiança. Começando pelas aplicações do projeto, que para se comunicarem com o Kafka, precisam passar pelo limite de confiança do cluster de produção e acessar o *gateway-proxy*, independentemente do contexto em que estão rodando, com o objetivo de obter os dados dos sensores em tempo real. Outra comunicação relevante é referente ao projeto selecionado emitindo comandos para controlar sua aplicação executando na VM-LSD, que também atravessa um limite de confiança.

Por fim, existem duas comunicações que não atravessam limites de confiança, sendo estas a comunicação entre a aplicação do projeto que executa fora do contexto da VM-LSD e o projeto selecionado, que está fora do escopo desta análise, e a comunicação entre o *gateway-proxy* e o Kafka. A partir do entendimento do diagrama de fluxo de dados, foi possível fazer a análise da figura e perceber três vetores que necessitam ser protegidos:

(1) A comunicação entre uma aplicação qualquer controlada pelo projeto e o cluster de produção atravessa um limite de confiança, por isso, requer autenticação e autorização para ter acesso aos tópicos. Também precisa de criptografia para proteger contra vazamento de dados e manipulações no meio do caminho.
(2) O acesso do projeto selecionado à máquina virtual do laboratório deve ser autenticado e criptografado para assegurar

que apenas usuários legítimos possam realizar o acesso SSH à máquina, pois atravessam o limite de confiança. Além disso, cada projeto deve ter limitações impostas ao seus usuários na máquina para não acessar ou manipular conteúdos de outros projetos, ou mesmo para não realizar uma escalada de privilégios, desligar a máquina, etc.

(3) O *gateway-proxy* precisa ser protegido contra ataques de negação de serviço, que podem sobrecarregar o sistema com um volume excessivo de solicitações, tornando-o inacessível para usuários legítimos.

4 SOLUÇÃO

Nesta Seção, será apresentada a solução para o controle de acesso e monitoramento do Kafka no cluster de produção, incluindo sua arquitetura e os desafios enfrentados durante a implantação.

4.1 Visão Geral

Com o objetivo de atender aos requisitos de segurança do *SmartCampus* UFCG, propõe-se a implantação de um cluster Kubernetes entre os clientes e o sistema de informações de uso de energia. Esse cluster abrigará um Kafka secundário, um SPIRE, um serviço proxy e os serviços de monitoramento, como pode ser observado na Figura 3. O propósito dessa abordagem é garantir a fácil operação e manutibilidade, além da segurança necessária sem modificar a arquitetura atual do LiteMe, por exemplo, ao adicionar um Kafka replicado, protegemos o Kafka original contra ataques DoS (negação de serviço) e escalada de privilégios. Assim, mesmo se as medidas de segurança forem insuficientes, o funcionamento do Kafka principal não será afetado.

Além disso, no Kafka secundário, foi configurada uma *Access Control List* (ACL), utilizando como base identidades SPIFFE. Esta abordagem visa garantir um controle granular de acesso aos tópicos no ambiente Kafka, permitindo apenas que entidades autorizadas e devidamente autenticadas possam interagir com os recursos e apenas realizar as operações que lhe são permitidas. Ao utilizar identidades SPIFFE como base para a construção da *Access Control List*, é possível estabelecer políticas de segurança robustas e flexíveis, alinhadas às práticas de *Zero Trust*[16].

A implantação de um conjunto de microsserviços de monitoramento também se torna fundamental para garantir a observabilidade do sistema. Esses microsserviços serão responsáveis por coletar e analisar métricas em tempo real sobre o desempenho, a integridade e a disponibilidade dos diversos componentes da infraestrutura. Através da observação contínua dessas métricas, será possível identificar rapidamente qualquer anomalia ou falha no sistema, permitindo uma resposta proativa a eventuais problemas e contribuindo para a manutenção da estabilidade e confiabilidade das operações.

A solução também visa a disponibilização de uma máquina virtual na nuvem privada do laboratório, onde os projetos selecionados poderão implantar suas aplicações. O propósito de disponibilizar essa máquina é estabelecer um ambiente confiável, permitindo a execução de um agente SPIRE. Dessa forma, em vez de solicitar credenciais ao operador, os clientes poderão desenvolver suas aplicações e utilizar o SPIFFE *helper* para lidar com as requisições de certificados ao SPIRE.

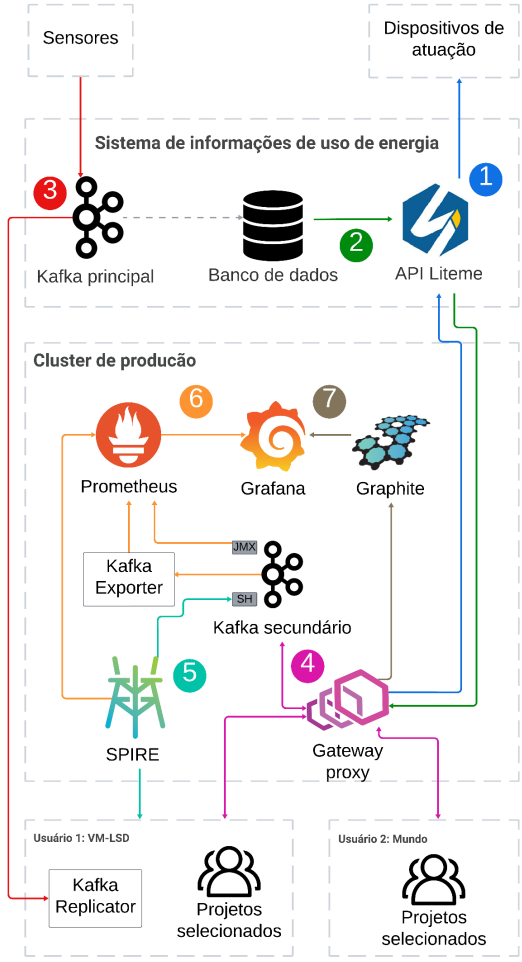

Figura 3: Diagrama da solução.

4.2 Arquitetura

A arquitetura é mostrada na Figura 3 e possui os seguintes componentes:

- O **cluster Kafka secundário** tem como objetivo adicionar uma camada de segurança necessária sem interferir no Kafka principal. Para alcançar isso, é implementada uma lista de permissões de acesso baseada nas identidades fornecidas pelo SPIRE. A solução utiliza um cluster Kafka com 3 *brokers* para aumentar a tolerância a falhas e emprega volumes persistentes para armazenar os dados e os *logs* dos *brokers*.
- O **SPIRE**, no contexto do *SmartCampus*, desempenha um papel fundamental ao gerenciar as identidades dos serviços internos e, principalmente, ao fornecer identidades para os serviços construídos pelos projetos selecionados. Uma das grandes vantagens de utilizar o SPIRE neste cenário é a facilidade de gerenciamento das identidades, juntamente com a renovação automática das mesmas. Isso não só simplifica significativamente a administração do sistema, mas também reduz os custos operacionais, já que não é necessário um

esforço humano constante para gerenciar e renovar as identidades manualmente.
Comparativamente, o OpenSSL[5], embora seja uma ferramenta amplamente reconhecida para gerenciamento de certificados e chaves criptográficas, tende a exigir mais esforço manual para configuração e manutenção. As operações de gerenciamento de certificados, como renovação e revogação, geralmente requerem intervenção humana direta e supervisionada. Além disso, a complexidade do OpenSSL pode aumentar o risco de erros humanos e atrasos na atualização das identidades, o que pode resultar em potenciais vulnerabilidades de segurança e custos operacionais mais altos a longo prazo.

- O **Envoy** desempenha um papel importante na implantação do cluster Kafka no Kubernetes. Em um ambiente com vários *brokers* Kafka, a comunicação dos clientes deve ser direcionada ao líder do tópico, porém, no Kubernetes os *services* realizam balanceamento de carga, o que pode resultar na comunicação sendo direcionada para a instância incorreta na maioria das vezes. Uma alternativa aos *services* tradicionais é o *headless service*, que não realiza balanceamento de carga e mas que também não expõe o serviço associado para fora do cluster. Ao utilizar o Envoy como gateway para o Kafka, em conjunto com o *headless service*, torna-se possível tornar o Kafka acessível de fora do cluster, garantindo que as mensagens alcancem o *broker* correto. Além disso, o Envoy facilita o monitoramento das requisições, fornecendo *insights* valiosos para diagnóstico e depuração.
- O **SPIFFE Helper** é utilizado como *sidecar* pelos *brokers* do Kafka secundário para obter o respectivo certificado. Além disso, as aplicações desenvolvidas pelos projetos podem incorporá-lo para simplificar a integração com o SPIRE.
- O **JMX exporter** é empregado como um *sidecar* para a coleta de métricas do Kafka. Ele é adicionado dentro do contêiner de cada instância do Kafka para capturar as métricas do *broker* ao qual está associado.
- O **Kafka Exporter** como explicado na Seção 2, realiza uma conexão com o cluster Kafka para coletar informações sobre o estado do cluster Kafka associado.
- O **Prometheus** é responsável por coletar e armazenar métricas de todos os serviços, exceto do Envoy. Essas métricas incluem dados provenientes do JMX Exporter, SPIRE e Kafka Exporter, conforme ilustrado na Figura 3, representado pelo fluxo 6.
- Conforme mencionado anteriormente, o Prometheus coleta métricas de todos os microsserviços, exceto do Envoy, devido a questões de compatibilidade. Para coletar métricas do Envoy, foi adotado o **Graphite**, uma ferramenta compatível com o Envoy que coleta e armazena métricas, representada no fluxo 7 da figura 3.
- O **Grafana** é a ferramenta central que reúne todos os dados relacionados às aplicações presentes no cluster. Através dela, o operador tem a capacidade de criar dashboards personalizados para monitorar e avaliar a saúde do sistema de acordo com as necessidades específicas.
- A **VM-LSD** representa a máquina virtual disponibilizada pelo laboratório e é uma parte importante da solução, pois provê um ambiente confiável e controlado para os projetos executarem suas aplicações, além de permitir um ambiente mais dinâmico com o agente SPIRE fornecendo os SVIDs.
- O **Kafka Replicator** é um script em Python projetado para estabelecer uma conexão com o Kafka principal como consumidor e, simultaneamente, criar uma conexão como produtor no Kafka secundário. Sua função é receber os dados do Kafka principal e encaminhá-los para o Kafka secundário, permitindo assim que o segundo Kafka replique o primeiro.

4.3 Controle de acesso

O primeiro passo para implementar controle de acesso no Kafka usando identidades SPIFFE envolve a configuração do *truststore* e do *keystore*[17]. Esses são os locais onde o Kafka respectivamente armazena os certificados das autoridades certificadoras (CAs) confiáveis e o próprio certificado juntamente com sua cadeia de certificação e chave privada. Para isso, é necessário obter os SVIDs e convertê-los para o formato `JKS`[6]. O caminho para os arquivos deve ser especificado no arquivo de configuração de cada *broker*, e cada um deve possuir seu próprio *keystore*, enquanto o *truststore* pode ser compartilhado entre eles.

Também deve ser adicionado um *KafkaPrincipalBuilder*[4] personalizado ao sistema, que suporte o SPIFFE. O *KafkaPrincipalBuilder* no Kafka é uma interface que facilita a construção de objetos representando princípios de segurança, como usuários e grupos, essenciais para implementar políticas de segurança, como controle de acesso baseado em ACLs. Para adicionar o *SpiffePrincipalBuilder*, o arquivo `.jar` correspondente deve ser colocado na pasta `./lib`, considerando o ponto como a pasta raiz do Kafka, e o caminho Java deve ser indicado em `principal.builder.class`[3], no arquivo de configurações.

Para habilitar a autorização, é necessário configurar o `authorizer.class.name`[5], que é a classe responsável por gerenciar as permissões. Para o Kafka secundário foi utilizado o *StandardAuthorizer*[6], a abordagem padrão para controle de acesso baseado em ACLs. Após a definição do autorizador, é preciso declarar pelo menos um super usuário com acesso privilegiado[7], este super usuário terá permissões abrangentes para administrar e configurar o Kafka, incluindo a ACL, sendo essencial para operações críticas de gerenciamento e manutenção.

Por fim, é necessário acessar a CLI do Kafka[8] para configurar as ACLs e adicionar as regras tanto para permitir a produção dos dados pelo Kafka Replicator quanto para habilitar o consumo pelos projetos, permitindo que os clientes se conectem. Na configuração das ACLs, é fundamental definir as permissões de forma precisa, garantindo que apenas as operações necessárias sejam permitidas. Do lado do cliente, é simples: basta obter um SVID válido e adicioná-lo aos parâmetros do cliente Kafka para garantir a autenticação correta e segura durante a conexão.

[5]Site do OpenSSL: https://www.openssl.org/

[6]Uma das formas de fazer a conversão é mostrada em: https://github.com/ufcg-lsd/kafka-spiffe-principal/?tab=readme-ov-file#create-keystore-and-trustore

[7]Exemplo de criação de super usuário: https://jaceklaskowski.gitbooks.io/apache-kafka/content/kafka-demo-acl-authorization.html

[8]CLI da ACL Kafka: https://jaceklaskowski.gitbooks.io/apache-kafka/content/kafka-tools-kafka-acls.html

4.4 Monitoramento

A arquitetura apresentada também demonstra uma capacidade robusta de monitoramento, sendo esta composta pelos fluxos 6 e 7 da Figura 3. No fluxo 6, é possível visualizar todas as comunicações realizadas pelo Prometheus, que periodicamente faz requisições aos serviços para obter suas métricas. Simultaneamente, o Grafana extrai periodicamente as métricas armazenadas no Prometheus, permitindo que os operadores construam *dashboards* com dados em tempo real. Vale ressaltar que o JMX exporter não é incluído diretamente na imagem, é necessário inseri-lo no container junto de seu arquivo de configuração e ativá-lo na flag `EXTRA_ARGS`, como pode ser visto no Código 1.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kafka-config
data:
  ...
  EXTRA_ARGS: "-Xms128m -Xmx256m -javaagent:/tmp\
  /jmx-agent/jmx-agent.jar=7071:/tmp/jmx-config\
  /jmx-config.yml"
```

Código 1: Configuração para habilitar o JMX no Kafka

O fluxo 7 corresponde à integração com o Graphite, onde o Envoy envia as métricas para essa ferramenta, que são posteriormente coletadas pelo Grafana em intervalos regulares. Um ponto de melhoria identificado nesta etapa é que as configurações atuais do Envoy utilizam um endereço sem resolução de nome. Atualmente, o Envoy só resolve nomes na configuração de clusters, o que significa que é necessário fornecer diretamente o endereço IP do serviço do Graphite para que o Envoy possa enviar as métricas. Esse cenário apresenta um desafio em ambientes dinâmicos, como o Kubernetes, no qual o IP do serviço pode mudar a cada implantação ou alteração no sistema. Uma abordagem mais adequada para esse contexto seria o uso do endereço DNS do Kubernetes.

4.5 Implantação

Implantar um Kafka replicado no Kubernetes e torná-lo acessível fora do cluster apresenta desafios significativos, especialmente considerando as necessidades específicas da solução, como a utilização de uma fonte de autenticação não convencional e ferramentas de monitoramento não nativas à aplicação. Assim como outros sistemas *stateful*, como bancos de dados, o Kafka requer garantias específicas ao ser executado no Kubernetes, uma vez que precisa manter um estado consistente para acompanhar tópicos e partições. Essa exigência torna difícil para o Kubernetes redistribuir ou escalar facilmente uma carga de trabalho do Kafka.

Entretanto, ao utilizar o Kubernetes, obtém-se um sistema facilmente replicável, flexível a mudanças, como adição ou remoção de nós, bem como modificações nas configurações, como a versão de um determinado serviço ou o IP de outra aplicação. Além disso, o uso do Kubernetes possibilita uma gestão centralizada e automatizada dos recursos, facilitando a escalabilidade horizontal do Kafka, conforme a demanda, e garantindo uma maior confiabilidade e disponibilidade do sistema.

Ao optar pelo uso do *StatefulSet* em vez de *Deployments*, obtemos diversas garantias adicionais em relação à unicidade e à ordenação dos pods. Por exemplo, os pods mantêm nomes persistentes, o que facilita a comunicação entre eles e evita problemas de dependência de nomes dinâmicos. Além disso, o StatefulSet assegura a preservação do estado dos volumes durante a exclusão, garantindo que os dados persistam mesmo quando os pods são removidos.

Além disso, conforme mencionado na seção que aborda o Envoy na arquitetura, a utilização de um serviço *headless*[12] para direcionar as requisições ao pod correto, juntamente com o uso do Envoy como NodePort, foi crucial para garantir a comunicação eficaz. A complexidade das configurações específicas de cada *broker*, incluindo a necessidade de um ID único e um *advertised listener* personalizado, exigiu uma abordagem alternativa para a definição das variáveis de ambiente no manifesto do *StatefulSet*. Nesse sentido, foi utilizado o *hostname* e o *pod-index* para gerar o valor dessas variáveis.

4.6 Fluxo do usuário

Como mencionado anteriormente, o cliente tem a opção de executar sua aplicação em dois contextos distintos: em uma máquina virtual fornecida pelo laboratório ou em qualquer outro contexto disponível. Nesta seção, vamos nos aprofundar para explicar mais detalhadamente o significado de cada um desses contextos e as diferenças entre eles.

Ao optar por executar a aplicação na VM-LSD, o projeto selecionado receberá credenciais de acesso para um usuário específico, criado para esse fim em uma máquina virtual localizada no laboratório. Esta máquina virtual será controlada e gerenciada pelo suporte do laboratório, visando restringir as capacidades de outros usuários, tais como acesso a arquivos e pastas, à rede e execução de comandos, garantindo assim a integridade do ambiente.

A partir dessas credenciais, o usuário terá a capacidade de realizar SSH e implantar suas aplicações conforme necessário. Para autenticar-se no Kafka sem a necessidade de solicitar certificados ao operador, o cliente utilizará solicitações ao agente SPIRE, que estará implantado na máquina. Para isso, ele pode optar por desenvolver sua aplicação de modo que seja compatível com o SPIRE ou utilizar o SPIFFE *helper* para facilitar o processo.

Outra alternativa é executar a aplicação em qualquer outro contexto, embora essa abordagem deva ser desencorajada e tratada como exceção. Isso ocorre porque implica que os clientes precisarão solicitar certificados ao operador sempre que necessário, diminuindo a eficiência operacional. Ao padronizar o uso da VM-LSD e do agente SPIRE, podemos garantir um ambiente controlado para a implementação das aplicações, reduzindo assim a sobrecarga operacional e garantindo uma experiência mais confiável para os usuários.

No entanto, existem cenários em que é justificável abrir exceções. Por exemplo, determinadas aplicações podem exigir recursos ou capacidades que não estão disponíveis na infraestrutura ofertada. Ou também, se uma aplicação depende de tecnologias ou plataformas não suportadas pela máquina virtual, uma exceção pode ser necessária para sua execução adequada.

5 ANÁLISE DE SEGURANÇA

A solução apresentada foi construída para mitigar as ameaças discutidas na Seção 3. Essa seção revisita os vetores e como a solução lida com eles.

No vetor 1, o acesso ao Kafka é restrito a usuários com identidades emitidas pelo SPIRE presente no cluster. Após autenticados, os usuários só podem acessar os tópicos para os quais têm permissão e realizar as operações permitidas, que se limitam à leitura, de acordo com a ACL. Além disso, a comunicação é protegida por mTLS, garantindo a confidencialidade dos dados em trânsito.

Para o vetor 2, membros de projetos selecionados só podem acessar as máquinas virtuais com um par de chaves SSH válido, gerado pelo suporte do laboratório. Além disso, a solução espera que o operador implemente políticas de segurança para isolar os usuários e limitar suas capacidades.

Por fim, o vetor 3 não é mitigado. Idealmente, o *gateway-proxy* deveria incluir configurações de *rate limiting* para proteger contra possíveis sobrecargas no sistema. Apesar de não ter sido implementado, a solução para este problema é conhecida e está prevista dentre os próximos passos.

6 AVALIAÇÃO

Nesta seção, detalharemos o ambiente de testes utilizado, apresentaremos um exemplo de lista de controle de acesso e forneceremos uma demonstração de um *dashboard* que pode ser construído a partir das métricas coletadas durante o monitoramento.

A infraestrutura disponível para os testes consiste em quatro máquinas virtuais. Três dessas máquinas estão configuradas com 2 vCPUs, 4GB de RAM e 20GB de espaço em disco, designadas para os *workers*. A quarta máquina, com uma configuração de 4 vCPUs, 8GB de RAM e 20GB de espaço em disco, é designada para o *master*.

```
ubuntu@master-k8s:~/raisson/ufcg-energia-2024/kafka/k8s$ kubectl top nodes
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master-k8s     296m         7%     2838Mi          36%
worker-k8s-1   354m         17%    2112Mi          55%
worker-k8s-2   361m         18%    1941Mi          50%
worker-k8s-3   582m         29%    2005Mi          52%
ubuntu@master-k8s:~/raisson/ufcg-energia-2024/kafka/k8s$
```

Figura 4: Captura de tela mostrando uso dos recursos computacionais dos nós no cluster Kubernetes.

Após a implementação dos serviços no cluster e a execução do Kafka replicator, foi utilizado o comando `kubectl top nodes` para monitorar o uso de recursos nos nós do cluster, incluindo CPU e memória. Os resultados estão representados na Figura 4, revelando que o sistema está operando com uma margem de reserva significativa dentro dos limites esperados de recursos. No momento, cada nó *worker* utiliza cerca de 2GB de RAM dos 4 disponíveis, e nenhum deles consome mais de 30% da capacidade da CPU da máquina.

Já na Figura 5 é possível evidenciar que as instâncias do Kafka consomem uma parcela significativa dos recursos em comparação com os outros microsserviços, devido a ser o serviço alvo das requisições, como era esperado. Além disso, para o funcionamento adequado do cluster é necessário a disponibilidade de 8,3Gb em armazenamento para os volumes persistentes, distribuído entre as máquinas. Esses volumes que guardam dados e *logs* do Kafka, alterações na configuração dos *dashboards* do Grafana e dados do SPIRE *server*.

No que diz respeito ao controle de acesso, esperava-se que a adoção do SPIRE como provedor de identidades não complicasse a

```
ubuntu@master-k8s:~/raisson/ufcg-energia-2024/kafka/k8s$ kubectl top pods
NAME                                  CPU(cores)   MEMORY(bytes)
envoy-6749d48c6c-wmwqk                5m           20Mi
grafana-5dcc9dcd7b-tv6j8              8m           74Mi
graphite-77f548d648-kjn2b             20m          191Mi
kafka-0                               209m         416Mi
kafka-1                               230m         429Mi
kafka-2                               233m         415Mi
kafka-exporter-7f59945b98-hh59h       3m           9Mi
prometheus-85748ff959-7v4sv           31m          189Mi
ubuntu@master-k8s:~/raisson/ufcg-energia-2024/kafka/k8s$ kubectl top pods -n spire
NAME                CPU(cores)   MEMORY(bytes)
spire-agent-ntpwt   3m           25Mi
spire-agent-qk7sz   4m           16Mi
spire-agent-zqn4k   4m           22Mi
spire-server-0      9m           88Mi
ubuntu@master-k8s:~/raisson/ufcg-energia-2024/kafka/k8s$
```

Figura 5: Captura de tela mostrando uso dos recursos computacionais dos pods no cluster Kubernetes.

configuração da ACL, e de fato não complicou. Ao criar uma nova regra utilizando SPIFFE em vez do CN (*Common Name*) ou qualquer outro campo presente nos certificados x509 convencionais, foi possível estabelecer as regras normalmente. No geral, escolher o SPIRE como fonte de autenticação resultou em um nível de segurança similar ao oferecido pelo OpenSSL, com a vantagem de simplificar os processos e aumentar a eficiência operacional.

A Figura 6 demonstra como ficaria uma configuração, o exemplo exposto contém permissões de criação de tópicos e escrita para o Kafka Replicator, de descrição das informações do cluster para o Kafka Exporter e de leitura para cada projeto, separada por tópico.

```
ubuntu@worker-k8s-2:~/kafka_2.13-3.6.1$ bin/kafka-acls.sh --bootstrap-server 10.5.9.141:30002 --list --command-config ./acl.properties
Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=LICENSE, patternType=LITERAL)`:
        (principal=SPIFFE:spiffe://example.org/kafka-replicator, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=SPIFFE:spiffe://example.org/kafka-replicator, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=SPIFFE:spiffe://example.org/kafka-exporter, host=*, operation=DESCRIBE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=GROUP, name=LICENSE, patternType=LITERAL)`:
        (principal=SPIFFE:spiffe://example.org/projeto1, host=*, operation=READ, permissionType=ALLOW)
        (principal=SPIFFE:spiffe://example.org/projeto2, host=*, operation=READ, permissionType=ALLOW)
        (principal=SPIFFE:spiffe://example.org/kafka-exporter, host=*, operation=DESCRIBE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=electric_measurements_LSD, patternType=LITERAL)`:
        (principal=SPIFFE:spiffe://example.org/projeto1, host=*, operation=READ, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=electric_measurements_POP, patternType=LITERAL)`:
        (principal=SPIFFE:spiffe://example.org/projeto2, host=*, operation=READ, permissionType=ALLOW)

ubuntu@worker-k8s-2:~/kafka_2.13-3.6.1$
```

Figura 6: Captura de tela mostrando exemplo de ACL.

Por fim, a Figura 7 apresenta um exemplo de *dashboard*, exibindo informações importantes, como o número de *brokers* Kafka ativos e a taxa de mensagens recebidas e enviadas por segundo, segmentadas por tópico. Além disso, são exibidas métricas do SPIRE, como o status do servidor e a quantidade de SVIDs gerados manualmente.

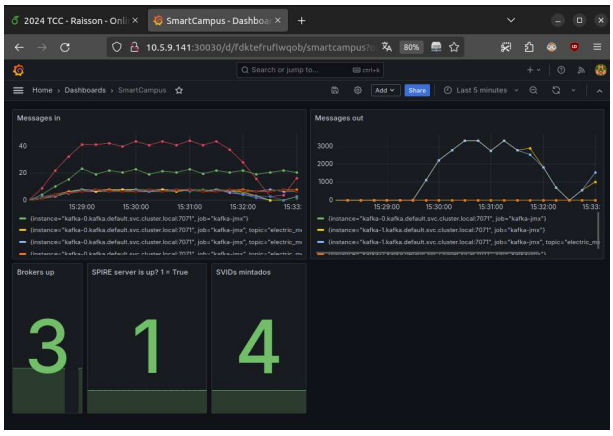

Figura 7: Captura de tela mostrando exemplo de *dashboard*.

7 CONCLUSÕES

Este trabalho exemplifica a implementação do controle de acesso de um cluster Kafka utilizando o SPIRE como provedor de identidade. Para realizar essa implantação em uma infraestrutura, foram utilizados manifestos Kubernetes, os quais expressam de forma declarativa a arquitetura desejada. Essa abordagem não só facilita a replicação, escalabilidade e modificação da arquitetura, como também assegura uma gestão eficiente do ambiente.

Além disso, o trabalho destaca a importância do monitoramento, não apenas do Kafka, mas de todos os serviços presentes no cluster Kubernetes. Como resultado, é possível prover o controle de acesso e o monitoramento do Kafka sem interferir na estrutura principal do Kafka no LiteMe. Todas as contribuições realizadas durante este período serão disponibilizadas nos repositórios de usuário[2] e de operador[1] do *SmartCampus* UFCG no GitHub. Este repositório incluirá todos os manifestos utilizados, juntamente com documentação detalhada sobre a implantação e o uso das ferramentas, visando facilitar a compreensão e a adoção por parte da comunidade interessada.

Por fim, é importante ressaltar que esta solução ainda oferece oportunidades significativas para melhorias adicionais, sendo estas:

- Implementação de mecanismos de *rate limiting* no *gateway-proxy* para prevenir ataques de negação de serviço.
- Aprimoramento da segurança dos volumes persistentes, visto que a solução atual depende do uso de *host paths*.
- Adicionar um mecanismos de rotação de SVIDs para os *brokers* Kafka sem gerar indisponibilidade momentânea.
- Adoção da comunicação mTLS entre os microserviços do cluster de produção pode ser implementada para garantir uma abordagem alinhadas às práticas de *Zero Trust*.
- Explorar maneiras de aprimorar a experiência do usuário final com o Kafka, como a introdução de um *client-proxy* entre o cliente e a arquitetura proposta, permitindo autenticação sem a necessidade de modificação da aplicação do cliente.
- Melhorar a comunicação entre os agentes SPIRE e o Prometheus. Atualmente, o Prometheus envia solicitações usando o DNS do *service* designado para a telemetria dos agentes. Porém, o *service* é configurado como um clusterIP, ocorre *load balancing*, resultando na recepção de métricas de apenas um dos agentes, em vez de coletar dados de todos os agentes.
- Melhoria da flexibilidade e modularidade dos manifestos, reduzindo sua dependência e tornando-os mais dinâmicos, especialmente do Kafka e do Envoy. Por exemplo, o Envoy atualmente não consegue acessar o Graphite usando o DNS do *service* no cluster, sendo necessário utilizar o seu IP. No entanto, a cada implantação desse serviço, um IP diferente é atribuído, exigindo que o endereço correto seja manualmente ajustado no Envoy em cada implantação.

AGRADECIMENTOS

Gostaria de expressar minha gratidão a todas as pessoas que contribuíram de diversas maneiras para a realização deste trabalho:

Primeiramente, minha eterna gratidão à minha mãe, Osilene Bezerra Grangeiro, cujo apoio incondicional e incentivo foram fundamentais em todos os momentos desta jornada acadêmica. Seu amor e dedicação são fontes inesgotáveis de inspiração.

Ao meu orientador, professor Andrey Elísio Monteiro Brito, por sua orientação não só para este trabalho, mas durante boa parte da minha jornada acadêmica e valiosos *insights* ao longo deste processo.

Gostaria de expressar minha sincera gratidão a Brenda Louisy Morais Alves pela valiosa colaboração nas discussões sobre a solução proposta e no trabalho apresentado. Sua contribuição foi fundamental, focando na resolução dos casos de uso 1 e 2, conforme exemplificado na Figura 1.

Também gostaria de agradecer a Lívia Buriti pela construção do Kafka Replicator e ao projeto "Ecossistema baseado em IoT para gerência de água e energia"(edital 09/2021, Demanda Universal), uma parceria entre a UFCG e a FAPESQ-PB.

Por fim, expresso minha gratidão a todos os professores, colegas, amigos e familiares que, de alguma forma, contribuíram para esta conquista. Seu apoio e encorajamento foram fundamentais para minha jornada acadêmica.

REFERÊNCIAS

[1] 2024. Repositório para operadores do *SmartCampus no GitHub*. https://github.com/ufcg-lsd/smartufcg-energia-ops Acessado em 18/5/2024.
[2] 2024. Repositório para usuários do *SmartCampus no GitHub*. https://github.com/ufcg-lsd/smartufcg-energia Acessado em 18/5/2024.
[3] Apache Software Foundation. 2010. Documentação sobre o *PrincipalBuilder* no arquivo de configurações. https://kafka.apache.org/documentation/#brokerconfigs_principal.builder.class Acessado em 4/5/2024.
[4] Apache Software Foundation. 2010. Javadoc sobre *KafkaPrincipalBuilder*. https://kafka.apache.org/28/javadoc/org/apache/kafka/common/security/auth/KafkaPrincipalBuilder.html Acessado em 4/5/2024.
[5] Apache Software Foundation. 2023. Documentação sobre o *Authorizer* no arquivo de configurações. https://kafka.apache.org/documentation/#brokerconfigs_authorizer.class.name Acessado em 4/5/2024.
[6] Apache Software Foundation. 2023. Documentação sobre qual *Authorizer* usar. https://kafka.apache.org/documentation/#security_authz Acessado em 4/5/2024.
[7] Apache Software Foundation. 2023. Kafka introduction. https://kafka.apache.org/intro Acessado em 4/5/2024.
[8] Chris Davis. 2008. Overview of Graphite. https://graphite.readthedocs.io/en/latest/overview.html Acessado em 4/5/2024.
[9] Daniel (Shijun) Qian. 2023. Repositório do Kafka Exporter. https://github.com/danielqsj/kafka_exporter Acessado em 4/5/2024.
[10] Envoy Project Authors. 2024. What is Envoy? https://www.envoyproxy.io/docs/envoy/latest/intro/what_is_envoy Acessado em 2/5/2024.
[11] Grafana Labs. 2024. Introduction to Grafana. https://grafana.com/docs/grafana/latest/introduction/ Acessado em 3/5/2024.
[12] Linux Foundation. 2024. Documentação sobre o *headless service*. https://kubernetes.io/docs/concepts/services-networking/service/#headless-services Acessado em 6/5/2024.
[13] Linux Foundation. 2024. Prometheus overview. https://prometheus.io/docs/introduction/overview/ Acessado em 3/5/2024.
[14] Linux Foundation. 2024. Repositório do JMX Exporter. https://github.com/prometheus/jmx_exporter Acessado em 4/5/2024.
[15] Linux Foundation. 2024. Visão geral Kuberntes. https://kubernetes.io/pt-br/docs/concepts/overview/ Acessado em 4/5/2024.
[16] National Institute of Standards and Technology. 2020. *NIST Special Publication 800-207: Zero Trust Architecture*. Special Publication 800-207. National Institute of Standards and Technology, Gaithersburg, MD. 59 pages. https://doi.org/10.6028/NIST.SP.800-207 Available free of charge from https://doi.org/10.6028/NIST.SP.800-207.
[17] Oracle Corporation. 2010. Página da Oracle sobre keystore e truststore. https://docs.oracle.com/cd/E19509-01/820-3503/ggffo/index.html Acessado em 4/5/2024.
[18] OWASP Foundation. 2010. Documentação OWASP sobre modelagem de ameaça utilizando STRIDE. https://owasp.org/www-community/Threat_Modeling_Process Acessado em 4/5/2024.
[19] The SPIFFE authors. 2024. Repositório do SPIFFE helper. https://github.com/spiffe/spiffe-helper Acessado em 2/5/2024.
[20] The SPIFFE authors. 2024. SPIFFE Overview. https://spiffe.io/docs/latest/spiffe-about/overview/ Acessado em 4/5/2024.
[21] The SPIFFE authors. 2024. SPIRE Concepts. https://spiffe.io/docs/latest/spire-about/spire-concepts/ Acessado em 3/5/2024.