RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

APLICAÇÃO PARA CONTROLE E MONITORAMENTO REMOTO DE INTEGRAÇÕES DE DADOS HETEROGÊNEAS

APPLICATION FOR REMOTE CONTROL AND MONITORING OF HETEROGENEOUS DATA INTEGRATIONS

APLICACIÓN PARA CONTROL REMOTO Y MONITOREO DE INTEGRACIONES DE DATOS HETEROGÉNEOS

Leonardo Elias de Oliveira[1], Rodrigo Daniel Malara[2]

e3122339
https://doi.org/10.47820/recima21.v3i12.2339

PUBLICADO: 12/2022

RESUMO
Durante a expansão de tecnologias e projetos dentro das companhias, aplicações responsáveis por realizar a integração de dados entre a origem e destino são criadas, essas aplicações podem e normalmente não se encontram centralizadas ou compartilham algum estado entre si. Contanto, a aplicação apresentada durante este trabalho visa auxiliar a manutenção destas aplicações realizando a centralização do controle e visualização de *logs*, através da *Stack* ELK e comunicação via protocolo HTTP. Concluindo, através da centralização e controle dos *logs* de execução da aplicação, tornou-se facilitado o rastreamento de situação de cada uma das aplicações, diminuindo-se o tempo necessário para resolver exceções e consequentemente tornando os serviços disponíveis.

PALAVRAS-CHAVE: Logs de execução. Integração. HTTP. ELK.

ABSTRACT
During the expansion of technologies and projects within companies, applications responsible for performing the integration of data between source and destination are created, these applications can and usually are not centralized or share some state between them. Therefore, the application presented during this work aims to assist the maintenance of these applications by centralizing the control and visualization of logs, through the Stack ELK and communication via HTTP protocol. In conclusion, through the centralization and control of the application's execution logs, it became easier to track the situation of each application, reducing the time needed to solve exceptions and consequently making the services available.

KEYWORDS: *Execution logs. Integration. HTTP. ELK.*

RESUMEN
Durante la expansión de las tecnologías y proyectos dentro de las empresas, se crean aplicaciones encargadas de integrar datos entre origen y destino, estas aplicaciones pueden y generalmente no están centralizadas ni comparten ningún estado entre sí. Siempre y cuando, la aplicación presentada durante este trabajo tiene como objetivo ayudar al mantenimiento de estas aplicaciones mediante la realización de la centralización del control y visualización de logs, a través de stack ELK y comunicación vía protocolo HTTP. En conclusión, a través de la centralización y el control de los registros de ejecución de la aplicación, se hizo más fácil rastrear la situación de cada una de las aplicaciones, reduciendo el tiempo requerido para resolver excepciones y, en consecuencia, poniendo a disposición los servicios.

PALABRAS CLAVE: *Registros de ejecución. Integración. HTTP. ALCE.*

[1] Universidade de Araraquara - UNIARA
[2] Universidade de Araraquara - UNIARA

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

INTRODUÇÃO

Dados são transmitidos o tempo todo, informações de negócio, venda, marketing, monitoramento são recebidos e enviados através da rede entre diferentes fontes de dados e empresas.

Tornando necessária a implantação de sistemas de integração para que dados críticos e estratégicos sejam trafegados de forma segura, performática e com integridade.

Diferentes aplicações autônomas são criadas, conforme necessidade, pelo time de desenvolvimento, para suprir as necessidades de integração do modelo de negócios proposto pela empresa, sistemas aos quais serão mantidos por algum período, porém, levando em consideração que as aplicações podem estar em ambientes distintos ou em redes separadas, até mesmo por questões de performance e segurança, a manutenção de configuração, monitoramento e acertos pontuais ao sistema integrador pode ser uma tarefa complexa, redundante ou custosa no quesito tempo.

Durante a usabilidade produtiva da integração, pode surgir a necessidade da realização de alterações em parâmetros que influenciam sua execução e definem regras de negócio, no entanto, manter essas variáveis separadas do servidor onde a integração está ocorrendo, deixando-as consultáveis através de um servidor central, pode soar muito interessante, prevenindo a necessidade de futuras alterações na aplicação para atualização dos valores dessas variáveis.

Além da parametrização, manter o controle de execução, estados e atualizações sobre o código da integração pode ser uma tarefa árdua, pois os desenvolvedores necessitaram acessar o servidor final a qual ela está hospedada e aplicar as devidas atualizações, sendo necessário gerenciar credenciais e ter acesso a infraestrutura a qual a integração está sendo executada.

Portanto, como solução para os problemas citados acima, esse trabalho visa a criação de um ecossistema web capaz de atender múltiplos clientes e centralizar a configuração, monitoramento, *logs* de execução de uma ou mais integrações remotamente, sem a necessidade do conhecimento inicial e direto da existência dos servidores finais em que as integrações estarão em execução.

As aplicações integradoras citadas nesse trabalho utilizarão como linguagem de programação Java, no entanto, será utilizado o *GRADLE* como ferramenta de *build* e gerenciamento de dependências do projeto para realizar a atribuição de dependências necessárias para comunicação com o servidor controlador.

Toda a comunicação a ser realizada entre o servidor controlador e a aplicação integradora será através do protocolo Https, garantindo a segurança e criptografia dos dados trafegados, sendo que, toda implementação de comunicação Https e demais protocolos de comunicação será realizada em diferentes módulos *GRADLE,* cada qual com sua função.

O ecossistema geral da aplicação controladora será dividido em cinco partes agregadas, interface com usuário *(front-end),* micro serviços *REST* para lidar com consultas realizadas pela interface, ElasticSearch como armazenador dos *logs* e estados da aplicação, *Socket* API para recepção e ingestão de diferentes *logs* e os módulos responsáveis por envio de *logs*, consulta de parâmetros e consumo de eventos enviados pela aplicação controladora.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Através das estruturas apontadas acima, o usuário final será capaz de realizar consultas ao *log* de execução de diferentes integrações, filtrando por campos pertinentes, controlar a integração através de comandos específicos e atualizar valores de parâmetros remotamente através de uma interface *WEB* amigável ou então através diretamente de chamadas à *API REST*, sem a necessidade de acesso direto ao servidor e com a capacidade de criar um melhor controle e monitoramento das aplicações existentes.

JAVA

Devido a necessidades de um projeto, novas ferramenta na área da tecnologia surgem, dentre elas, as chamadas linguagens de programação, cada qual desenvolvida sobre seu paradigma com intuito de resolver problemas específicos.

Segundo JAVA, Java é uma linguagem de programação orientada a objetos, criada pela Sun Microsystems em 1995, multiplataforma compilada e interpretada, ou seja, o programador gera um código binário *(bytecode)* dos códigos escritos em tempo de compilação e esse código binário será executado pela Java Virtual Machine, por este fato e por possuir diversos algoritmos implementados em sua API, ela foi escolhida para ser utilizada como linguagem base de desenvolvimento deste trabalho.

No entanto, como citado por REDKO a linguagem é uma das mais utilizadas no mundo, de acordo com o índice TIOBE, composta por uma poderosa biblioteca, sintaxe simples, a linguagem é responsável por manter aplicativos móveis, *backends* modernos e até mesmo sistemas *desktop* legados, isso por conta da sua grande facilidade de execução em ambientes diversos e poder ser escrito uma única vez, dando ênfase ao slogan criado pela Sun em 1995, *"Write once, run anywhere"*.

Ferramentas de *build* e *Gradle*

Com o intensivo uso da linguagem de programação Java, foi surgindo a necessidade de realizar passos pós e pré *build* do projeto, desde instalação, gerenciamento de dependência, até configurações sobre o empacotamento da aplicação em JAR, WAR ou EAR para execução pela JVM, no entanto, ferramentas para realizar tal ação começaram a ser desenvolvidas, como *Maven, Gradle* e *Ant*.

Segundo a organização GRADLE (2022), Gradle é uma ferramenta de automação de *build open-source* focada na flexibilidade e performance de *build scripts* escritos utilizando as linguagens Groovy ou Kotlin DSL. Com o apoio de todas demais funcionalidades que o *Gradle* suporta, o suporte ao build com sub-projetos será utilizado por esse projeto para construção de bibliotecas e compartilhamento de pacotes e classes, possibilitando a reutilização da orientação a objetos.

Árvore de atividades de *build*

A ferramenta de *build GRADLE* utiliza do conceito de árvores de atividades dependentes que formam a dependência entre as atividades responsáveis por realizar algum passo durante a construção do projeto, assim como representado na figura 1.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

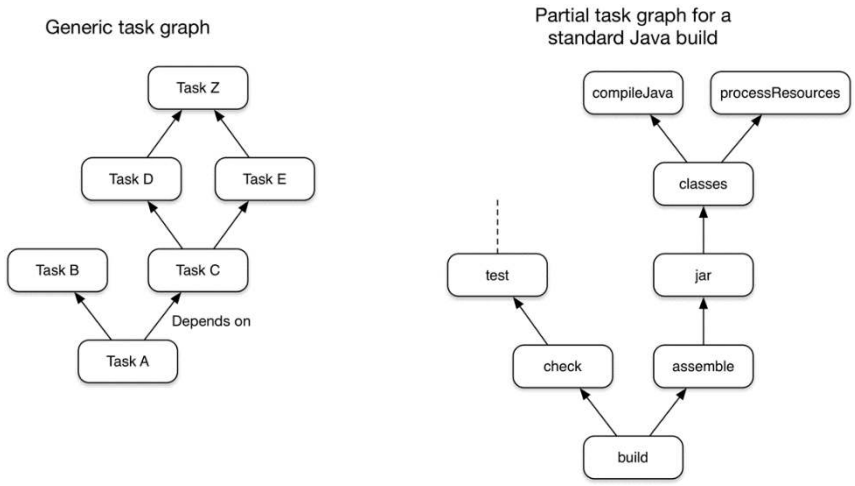

Figura 1: Árvore de atividades de *build*
Fonte: https://docs.gradle.org/current/userguide/what_is_gradle.html

Micro serviço

De acordo com Fowler (2014), sendo breve, micro serviço é o nome dado ao estilo de arquitetura onde uma aplicação independente e pequena, com seus próprios mecanismos de comunicação como HTTP, forneça uma função específica dentro do escopo de uma aplicação maior, sendo implantado independente de uma aplicação maior.

Ainda como citado por Fowler (2014), fica claro a utilização intensiva dessa arquitetura nos últimos anos, e os resultados positivos gerados pelo uso dela e o baixo acoplamento gerado por esse estilo de arquitetura.

Na figura 2, é possível visualizar a utilização de uma API *Gateway*, responsável por fazer o papel de API principal e porta de entrada para comunicação externa ao cliente, fazendo-se o uso de micro serviços internamente orquestrados para manter suas funcionalidades, modularizando o código fonte de acordo com cada tipo de serviço a ser consumido.

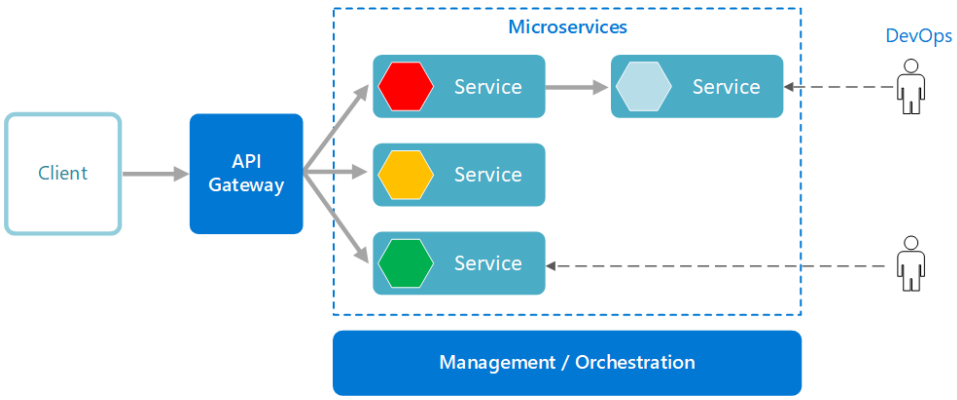

Figura 2 Estilo de arquitetura de micro serviço
Fonte: https://docs.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/microservices

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Diferente da arquitetura monolítica, através do uso de micro serviços é possível garantir alta disponibilidade e auto replicação de serviços, através da separação em módulos, pois, caso um dos micros serviços for interrompido por algum motivo inesperado, demais serviços continuarão em execução e disponíveis para o *Gateway* API.

Frontend

Interface criada para tornar acessível e amigável a usabilidade de um sistema ao usuário final, sendo possível acesso via *Desktop, Web* ou *Mobile* de uma aplicação, sendo que, essa interface pode ser desenvolvida do lado do servidor ou do cliente.

De acordo com a equipe de desenvolvimento TOTVS (2021), *front-end* é onde se desenvolve a aplicação com a qual o usuário irá interagir diretamente, seja em *softwares*, *sites*, aplicativos. Portanto, é essencial que o desenvolver tenha uma preocupação com experiência do usuário.

Serviço de fila

Dependendo do comportamento esperado, aplicações necessitam processar mensagens em tempos diferentes das quais foram produzidas, para permitir que mensagens sejam consumidas e processadas em tempo assíncrono ao qual foram produzidos de forma ordenada, serviços de fila podem ser utilizados.

Durante o desenvolvimento de um projeto de *software*, pode ser comum a necessidade da utilização de uma estrutura de dados fila, armazenar estados, eventos, mensagens, entre outras estruturas de dados, sendo necessário controlar a transmissão das mensagens entre uma ou mais aplicações.

De acordo com Feofiloff (2018), uma fila é uma estrutura de dados dinâmica que admite a remoção de elementos e inserção de novos objetos. Mais especificamente, uma fila é uma estrutura sujeita à seguinte regra de operação: sempre que houver uma remoção, o elemento removido é o que está na estrutura há mais tempo, em outras palavras, o primeiro objeto inserido na fila é também o primeiro a ser removido, essa política é conhecida pela sigla FIFO *(First-In-First-Out)*.

No entanto, existem diferentes algoritmos, serviços e formas de se usufruir de uma fila, uma delas é utilizando um *Middleware* Orientado a Mensagens, ela permite a comunicação assíncrona orientada a mensagens entre uma ou mais entidades, através de filas oferecidas por provedores.

De acordo com a VMWARE, RabbitMQ é considerado um corretor de mensagens open- source com alta gama de usuários desenvolvedores, utilizado por startups e grandes empresas.

Logs de execução

Com a finalidade de manter o histórico de execuções e chamadas principais de um *software*, o recurso de armazenagem de *logs* de execução é essencial, através dele o programador poderá criar monitoramento e utilizá-lo como ferramenta para análise caso ocorra futuros problemas na programação realizada.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Como pode ser visualizado na figura 3, o *log* é formado de alguns campos essenciais, esses são: data/hora da execução a qual o *log* faz referência, nível do *log* (e.g. informação, erro, aviso) e sua mensagem, ou seja, a aplicação fará o controle de adicionar esses *logs* a saída padrão ou salvá-los em arquivo, ficando como encargo de como foi feita a implementação dessa configuração de *log*.

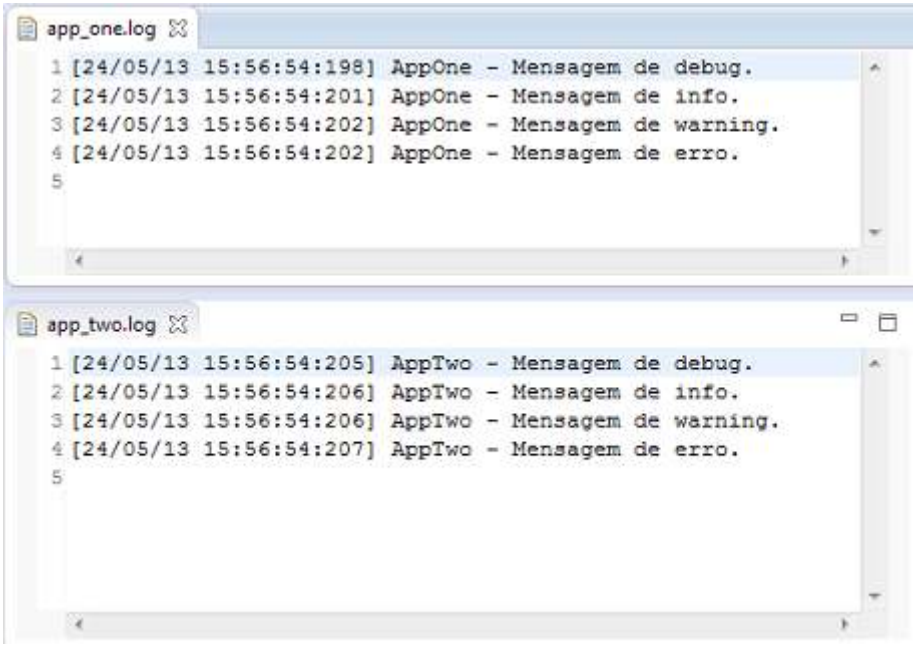

Figura 3: Console de log utilizando LOG4J
Fonte: https://blog.cvinicius.com.br/2013/05/criando-arquivos-de-log-com-log4j.html

REST API

Padrões de comunicação estão sendo desenvolvidos e melhorados ao decorrer da evolução das demais tecnologias. De acordo com REDHAT (2020), *"REST* é a sigla em inglês para *Representational State Transfer*, que em português significa transferência de estado representacional. Essa arquitetura foi criada pelo cientista da computação Roy Fielding.

De acordo com Redhat (2020), *API* é o conjunto de definições e protocolos utilizados durante a integração de dados entre um consumidor e um provedor, fornecendo base e documentação de como a conversa deve acontecer entre ambos para que se conclua com sucesso a comunicação.

A arquitetura *REST* utiliza como protocolo de comunicação HTTP ou HTTPS, com a utilização de modelos de dados para envio das mensagens como JSON, HTML, XLT, PHP ou até mesmo TEXT sem formatação, permitindo que o consumidor tenha acesso a um recurso através da sua *Uniform Resource Identifier* (URI).

Elasticsearch (Log Manager)

De acordo com a organização ELASTIC (2022), com o intuito de melhorar as análises e facilitar a encontrar um problema ou solução em *logs* de forma mais ágil e centralizada, o *Elasticsearch* foi concebido em sua primeira versão em 2010. Sendo declarado como mecanismo de busca e análise de

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

dados distribuído e com suporte a diferentes tipos de dados, texto, número, espacial, estrutura e não estruturado.

Sua função é armazenar os *logs* e prover de *REST APIs* simples para realização de consulta de *logs* utilizando filtro convenientes, como ilustrado na figura 4 pertence ao *Elastic Stack*, nome dado ao conjunto de ferramentas utilizadas para ingestão, armazenamento e acesso aos dados.

Figura 4. Conjunto de ferramentas Elastic para gerenciamento de logs
Fonte: https://www.elastic.co

FileBeat (Log Ingestor)

De acordo com ELASTIC (2022), "O *Beats* é uma plataforma gratuita e aberta para agentes de dados de finalidade única. Eles enviam dados de centenas ou milhares de computadores e sistemas para o *Logstash* ou o *Elasticsearch*."

Existem diferentes tipos de *Beats*, cada qual com sua função, métricas, *logs*, monitoramento de disponibilidade, auditoria, o *FileBeat* é um deles e ele trata da recepção e ingestão dos *logs* de diferentes tipos de aplicações, comunicando com outros produtos da *Stack Elastic.*

DESENVOLVIMENTO

O desenvolvimento de múltiplas aplicações de integração de dados dentro de uma corporação pode ser considerada uma tarefa relativamente simples de ser realizada, porém ainda mais importante, o acompanhamento de *logs*, execuções e agendamento das execuções pode ser o diferencial que irá definir o sucesso ou fracasso do projeto.

Para empresas provedoras de consultoria e serviço, o desafio de controlar diversos tipos de integrações locais simultaneamente em diferentes clientes pode ser ainda mais penoso, novos projetos e novas demandas chegam o tempo todo, sendo elas novidades ou não, novas aplicações são criadas para que o dado seja enviado de um ponto ao outro.

A dificuldade em controlar integrações entre diferentes tipos e clientes, se dá pelo fato da diversidade de possíveis ambientes aos quais as aplicações integradoras estão sujeitas, em sua maioria em redes virtuais privadas, inacessíveis por terceiros, as aplicações devem estar o mais

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

próxima possível entre os enlaces da origem e destino a quais os dados serão enviados, diminuindo a latência e tempo de comunicação.

Além do requisito rede para a execução de uma integração, outro ponto importante a ser levado em consideração é o isolamento e autonomia dos sistemas integradores, que devem independer da plataforma a qual executam e não devem afetar a execução das demais integrações.

A aplicação controladora de integração desenvolvida durante este trabalho será responsável por unificar todo o controle de integrações heterogêneas de origens distintas em uma só interface, permitindo usuários administrarem e consultarem sobre o estado das integrações utilizando uma única porta de entrada, sem se preocupar com acessos e sem que haja a necessidade da execução de integrações em um só ambiente, garantindo que a distribuição de diversos sistemas integradores pela rede estejam controlados transparentemente por uma só interface.

Arquitetura geral do projeto

Para o funcionamento robusto e congruente do sistema, módulos serão utilizados para manter o sistema de integração central funcionando, onde cada módulo faz seu trabalho específico, podendo ser acoplado a diferentes módulos para criação de uma funcionalidade.

Como pode ser visualizado na figura 5, a arquitetura central se baseia na tecnologia cliente-servidor, as integrações poderão adicionar como dependência módulos responsáveis por enviar os dados e controlar a integração remotamente, de forma a não gerar um forte acoplamento entre o código do módulo cliente que mantém o controle da integração e o código de negócio responsável por executar a integração dos dados do ponto A ao B.

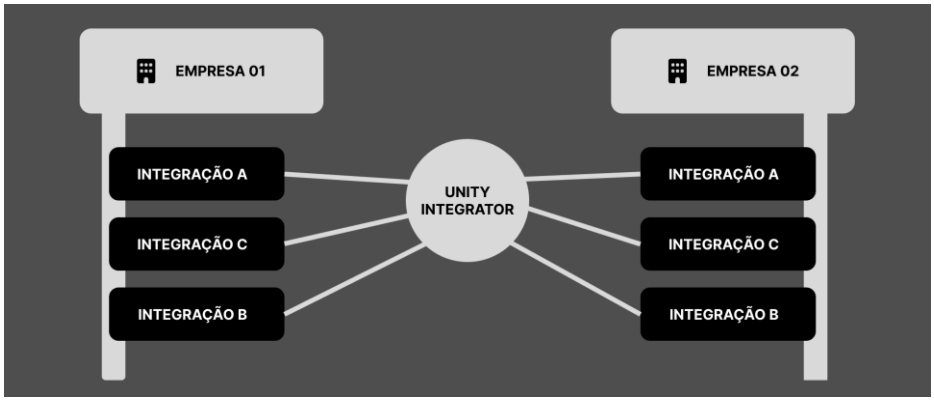

Figura 5. Diferentes integrações em diferentes empresas conectadas ao sistema único de integração
Fonte: Autor

Os clientes remotos, serão plugáveis ao projeto integrador, junto com as credenciais responsáveis por autenticar o integrador e possibilitar o envio dos dados à plataforma central, em geral os módulos remotos conectados aos integradores são três.

O módulo remoto de *log* é o responsável por enviar todos os *logs* gerados pela aplicação para o servidor central e enviar também o sinal de vida da aplicação.

O módulo remoto de parâmetros é o responsável por acessar e controlar remotamente os parâmetros que estão centralizados na interface central de configuração.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Já o módulo de controle remoto dá ao servidor central a capacidade de enviar comandos para controlar a execução da integração em execução remotamente.

Entidades

Como primeira entidade de controle, a autenticação de acesso ao sistema é realizada através do conjunto de usuários cadastrados no sistema, cada qual com sua identificação, credencial e cargo de acesso, por sua vez, podem estar atribuídos a uma ou mais.

Cada empresa terá consigo um conjunto de usuários e seus respectivos cargos, permitindo controle de acesso aos recursos de uma empresa. Através de chaves secretas utilizadas durante o processo de autenticação, as aplicações integradoras poderão realizar a troca de informações com o servidor central e por fim realizar o gerenciamento de uma ou mais aplicações registradas.

Determinada integração só poderá ser acessada via chamadas externas caso exista uma chave secreta que tenha permissão sobre essa aplicação e esteja sendo utilizada pelo cliente remoto durante o processo de autenticação, esta deverá ser configurada junto a biblioteca cliente.

Autenticação da integração

Garantir que os dados de uma aplicação sejam acessados somente por quem realmente possui acesso para tal é uma tarefa essencial realizada pela API central, no entanto, toda interação realizada sobre a plataforma deverá estar autenticada.

Como primeiro passo para controlar uma aplicação integradora, o usuário deve realizar o cadastro de uma nova aplicação integradora utilizando a *REST API* ou a interface gráfica diretamente, cadastrando os respectivos módulos remotos a serem utilizados pela aplicação, junto aos campos identificadores desta.

Após realizado o cadastro da aplicação integradora em uma empresa, o código de registro será gerado e retornado ao usuário, através deste código, uma nova chave secreta deverá ser criada com autorização de acesso a esta nova integração.

Com o código de registro da integração e a chave secreta em mãos, a aplicação integradora pode ser configurada, na lista de bibliotecas da aplicação integradora, a biblioteca dos módulos remotos será adicionada, após adicionado, a chave secreta e o código da aplicação deve ser especificada na inicialização da configuração da biblioteca.

Depois de configurado as credenciais na aplicação integradora, esta poderá ser executada, e a integração com a central já começará a acontecer, através do token de autenticação especificada no cabeçalho das requisições e poderá ser validada através do *endpoint* de status da aplicação, devendo responder com a tag de estado “Em Execução”.

Consumo de *logs*

Com a autenticação realizada com sucesso, o módulo cliente de *log* iniciará o envio dos *logs* ao servidor central, através de chamadas assíncronas HTTP ao *endpoint* do servidor o cliente enviará todos os *logs* gerados pela integração.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

De forma não bloqueante e paralela, as requisições para envio dos *logs* ao servidor central serão controladas através de executores escritores na linguagem Java, que farão a ordenação do envio de acordo com políticas internas, priorizando a performance no envio e garantindo que todos os *logs* serão enviados ao servidor.

Toda implementação do envio dos logs ao servidor está implementada seguindo o padrão SLF4J e os conceitos do *Logback*, ou seja, a aplicação integradora não precisará ter o conhecimento de como os *logs* são enviados, apenas que eles estão sendo enviados, facilitando a configuração desta para o uso.

Após enviado o *log* pelo cliente, este trafegará pela rede até chegar ao servidor *Ingestor*, que fará a recepção da requisição pelo consumo do *log*. Como primeiro passo após a chegada da requisição, através do código de registro da aplicação fornecido e a chave secreta, um filtro será realizado, verificando se a aplicação do código especificado existe e se a chave secreta fornecida é válida e possui acesso a integração do código de registro provido.

Validado o acesso a aplicação através das credenciais disponibilizada pelo cliente, o *log* será processado, validado e inserido no *ElasticSearch*, utilizando-se do índice específico criado para a integração durante o processo de cadastro desta, tornando possível futuras análises e buscas que serão feitas pela aplicação central.

Visualização dos *logs* de forma unificada

Após a integração configurada e as primeiras trocas de informações de *logs* ocorrerem, será possível visualizar de forma unificada os *logs* armazenados das aplicações através da API central, via chamada a API REST ou via interface do usuário.

Todos os *logs* armazenados no índice de cada código de integração poderão ser consultados posteriormente e visualizados pelo usuário final, que poderá filtrar pelo nível do *log* e sua prioridade de tratamento.

Através da interface, o usuário poderá filtrar pelo nível de *log* desejado, buscando de forma rápida evidências da execução de cada integração, sem precisar acessar a infraestrutura ao qual a aplicação integradora está em execução, apenas usando o acesso a interface web.

Visão geral das integrações

Autenticado na *interface web*, o usuário poderá visualizar a lista de empresas ao qual possui acesso e acessá-las de acordo com a necessidade, dentro de cada empresa o usuário poderá visualizar de forma fácil e analítica informações sobre as integrações que estão cadastradas.

Um *dashboard* contendo gráficos sobre a saúde das aplicações de acordo com o nível dos *logs* recebidos nos últimos momentos será apresentado ao usuário logo após escolhida a empresa desejada, podendo visualizar quantidade de *logs* recebidos e adentrar nos detalhes de cada nível de log.

Em um painel central da empresa, é possível visualizar a situação atual das integrações, através da data da última comunicação feita pelo cliente, junto com *tags* que representam a situação

atual de cada integração, podendo estar em EXECUÇÃO, INATIVA e com ERRO, tornando fácil mapear integrações que merecem a atenção do usuário final.

Disponibilidade dos serviços

Pela arquitetura em camadas dividida em módulos, os micro serviços responsáveis pela recepção e controle geral do sistema poderão estar disponíveis de forma replicada e balanceados através de balanceadores de carga, persistindo a alta disponibilidade da plataforma, garantindo a recepção dos *logs.*

Coleta de *logs* de execução

A partir do momento em que uma aplicação de integração registrada na plataforma esteja em execução, ela começará o envio dos *logs* de execução em tempo real para a plataforma central, através da implementação das interfaces do SLF4J. De acordo com ... SLF4J é uma biblioteca para implementação de logs em sistemas empresariais.

Essa implementação será composta de um *thread-pool* com uma quantidade fixa de número de *threads* para realizar o trabalho de enviar cada *log* realizado pela aplicação para o *FileBeat*, como pode ser visualizado na figura 6, após chegada a requisição de adicionar um novo log, a classe de log repassará o trabalho para algum dos *threads* disponíveis (marcadas em verde), dessa forma o log não dependerá da saída e poderá ser executado de forma assíncrona.

Figura 6. Exemplificação de log assíncrono
Fonte: Autor

De acordo com Oliveira (2019), O *FileBeat* é um *log shipper,* ou seja, através dele é possível coletar e enviar vários tipos de *log* para o *Elasticsearch.* Isso é super útil para coletas de *logs* de servidores em massa, pois se no seu ambiente existirem centenas ou até mesmo milhares de *hosts*, é inviável que em um momento de crise você entre em cada um deles buscando algo.

Pós realizada a ingestão pelo *FileBeat,* ele enviará os logs para o *ElasticSearch*, que fará a armazenagem do *log*, indexando através do identificador único da integração, tornando possível a consulta através de filtros.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Através das informações de logs armazenadas no *ElasticSearch*, o micro serviço de LOGS estará apto a realizar consultas e obter os *logs* e acordo com a necessidade, sendo que esse micro serviço será responsável por limitar o acesso do usuário somente aos *logs* a qual ele tem acesso.

Execução de comandos remotos

Para permitir a comunicação direcional do servidor central para respectivas integrações registradas, o micro serviço Controle Remoto, será responsável por gerenciar filas em RabbitMQ para cada integração cadastrada, onde cada fila receberá uma sequência de eventos a serem executados remotamente na integração, assim como pode ser visualizado na figura 7.

A aplicação integradora por sua vez, irá consumir em *background* os eventos dessa fila, dessa forma, o servidor central não precisa do acesso direto via rede ao servidor que armazena a aplicação de integração e caso um comando seja enviado para a aplicação e ela esteja desligada, o evento não será perdido, pois aguardará na fila até que a aplicação volte a executar.

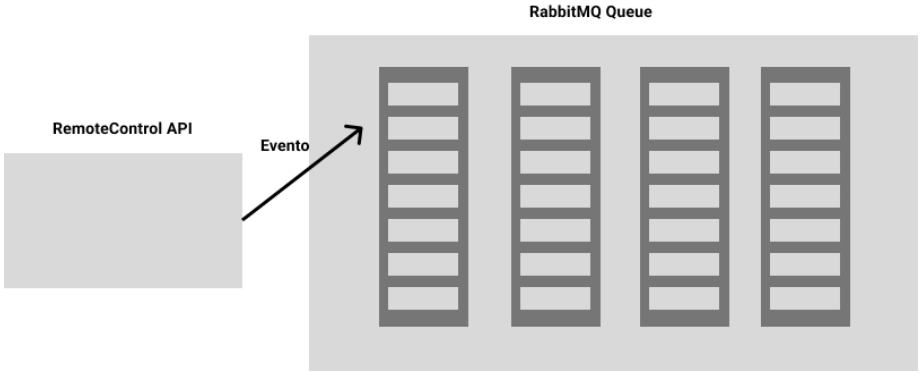

Figura 7. Comunicação entre servidor central e integrador via fila de mensagens
Fonte:Autor

Parâmetros remotos

Através da centralização da configuração das integrações, o micro serviço de PARÂMETROS proverá uma interface ao usuário, para configuração de chave, valor como no exemplo da tabela 1, que poderão ser obtidos e utilizados pela aplicação integradora em tempo de execução, possibilitando a alteração de valores e passagem de parâmetros sem que haja necessidade de intervenção direta ao servidor ao qual a integração está configurada.

Chave	**Valor**
URL_BASE	https://example.com
NUM_THREADS_EXECUCAO	30

Tabela 1. Exemplificação de parâmetros remotos configuráveis

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Os parâmetros serão configuráveis através da seleção de uma integração, no menu "Parâmetros", sendo que poderão ser dos tipos primitivos: "*String, Decimal, Integer".*

CONSIDERAÇÕES FINAIS

Com as aplicações integradoras centralizadas e sendo controladas remotamente através de uma única interface, a parametrização de valores, controle de execução do estado das integrações e principalmente as sessões de descobertas e análise de *logs* foram extremamente facilitadas.

O ponto abordado é muito importante a todo momento, a transparência durante as etapas de integração e rastreabilidade, rapidez na ingestão e pesquisa pelos *logs* das aplicações, devido a utilização das otimizações de índices providos pela tecnologia do *ElasticSearch*.

Concluindo, todas as aplicações se tornaram transparentes as áreas responsáveis da companhia, facilitando a manutenção, diminuindo o tempo gasto com análise na busca de *logs* de execução de um incidente, como também a diminuição do tempo de desenvolvimento de novas aplicações integradoras, devido a não haver mais a necessidade de implementar toda configuração de controle de *logs.*

REFERÊNCIAS

BARBOSA, A.; BIANCARDI, C.; SILVESTRE, L. **Integração de Dados Heterogêneos em Ambiente Web**. 2016. Monografia (Especialização) – Universidade Federal do Espírito Santo, Vitoria, 2016.

ELASTIC. **Elasticsearch: O que é?** [S. l.]: Elastic, 2022. Disponível em: https://www.elastic.co/pt/what-is/elasticsearch. Acesso em: 12 jun. 2022.

ELASTIC. **FileBeat**: Beat para logs. [S. l.]: Elastic, 2022. Disponível em: https://www.elastic.co/pt/beats/filebeat. Acesso em: 12 jun. 2022.

FEOFILOFF, P. **Filas**. 2018. TCC (Projeto) – Universidade de São Paulo, São Paulo, 2018. Disponível em: https://www.ime.usp.br/~pf/algoritmos/aulas/fila.html. Acesso em: 14 jun. 2022.

FOWLER, M. **Microservices**: a definition of this new architectural term. [S. l.]: Martin Flower, 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 14 jun. 2022.

GRADLE. **What is gradle?** [S. l.]: Gradle, 2022. Disponível em: https://docs.gradle.org/current/userguide/what_is_gradle.html. Acesso em: 14 jun. 2022.

JAVA. **What is java?** [S. l.]: Java, 2022. Disponível em: https://www.java.com/en/download/help/whatis_java.html. Acesso em: 14 jun. 2022.

MICROSOFT. **Estilo de arquitetura de microsserviço**. [S. l.]: Microsoft, 2022. Disponível em: https://docs.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/microservices. Acesso em: 13 jun. 2022.

OLIVEIRA, B. **Filebeat**: Instalação e configuração. [S. l.]: Medium, 2019. Disponível em: https://medium.com/sysadminas/filebeat-instala%C3%A7%C4%81o-e-configura%C3%A7%C4%81o-9f41b0d5dbe4. Acesso em: 14 jun. 2022.

REDHAT. **O que é uma REST API?** [S. l.]: REDHAT, 2020. Disponível em: https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api. Acesso em: 13 jun. 2022.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

REDHAT. **Stateful vs. Stateless**. [S. l.]: REDHAT, 2020. Disponível em: https://www.redhat.com/pt-br/topics/cloud-native-apps/stateful-vs-stateless. Acesso em: 13 jun. 2022.

TOTVS. **Front end**: O que é, como funciona e qual a importância. [S. l.]: TOTVS, 2021. Disponível em: https://www.totvs.com/blog/developers/front-end. Acesso em: 14 jun. 2022.