CURSO DE CIÊNCIA DA COMPUTAÇÃO – TCC	
(X) PRÉ-PROJETO ( ) PROJETO	ANO/SEMESTRE: 2022/2

PROTÓTIPOS DE MICROSERVIÇOS PARA APLICAÇÃO DE CONCEITOS DE OBSERVABILIDADE

Nestor Kammer

Marcel Hugo – Orientador

1 INTRODUÇÃO

O assunto sobre microsserviços vem ganhando rapidamente um grande espaço em mídias sociais, blogs, discussões, conferências e meio corporativo (ROSSI, 2021).

Embora os conceitos fundamentais por trás dos microsserviços não sejam novos, a aplicação contemporânea da arquitetura de microsserviços é atual, e sua adoção tem sido motivada em parte pelos desafios de escalabilidade, falta de eficiência, velocidade lenta de desenvolvimento e dificuldades em adotar novas tecnologias que surgem quando sistemas complexos de software estão contidos em e são implantados como uma grande aplicação monolítica (FOWLER, 2017).

Rossi (2021), basicamente define sistemas monolíticos como aplicações onde todos os seus componentes e funcionalidades estão combinados dentro de uma única aplicação e arquitetura.

Opus Software (2021) classifica que os principais desafios da arquitetura monolítica são os seguintes:

a) aumento da complexidade e tamanho ao longo do tempo;

b) alta dependência de componentes de código;

c) escalabilidade limitada do sistema;

d) falta de flexibilidade;

e) dificuldade para colocar alterações em produção.

Para Fowler (2017), adotar a arquitetura de microsserviços, seja a partir do zero ou dividindo uma aplicação monolítica existente em microsserviços desenvolvidos e implantados independentemente, resolve esses problemas. Com a arquitetura de microsserviços, uma aplicação pode ser facilmente escalada tanto horizontalmente quanto verticalmente, a produtividade e a velocidade do desenvolvedor aumentam dramaticamente e tecnologias antigas podem facilmente ser trocadas pelas mais recentes (FOWLER, 2017).

Atualmente, existem vários modelos de como um ecossistema de microsserviços pode ser organizado. Um deles, é dividido em quatro camadas (OPUS-SOFTWARE, 2021), sendo elas: hardware; comunicação; plataforma de aplicações; microsserviços.

Fowler (2017), destaca que existem alguns itens que são fundamentais para o sucesso das operações dos microsserviços. São eles: estabilidade; confiabilidade; escalabilidade; alta tolerância a falhas; desempenho; monitoramento; documentação.

Em se tratando de monitoramento, Fowler (2017) considera que um bom monitoramento tem quatro componentes: um adequado *logging* (registro) de todas as informações importantes e relevantes; interfaces gráficas úteis (*dashboards* de controle) que sejam facilmente compreendidas por todo desenvolvedor na empresa e que reflitam com precisão a saúde dos serviços; um sistema de alerta eficaz e acionável sobre as principais métricas; implementar e manter uma rotação das equipes de prontidão.

Um microsserviço pronto para produção é adequadamente monitorado. Um monitoramento adequado é uma das partes mais importantes de construir um microsserviço pronto para produção e garantir maior disponibilidade do microsserviço (FOWLER, 2017).

Com base nisso muitas soluções começaram a surgir para suprir essa demanda não só arquitetural como de identificação de pontos críticos em sistemas e possíveis gargalos. Atualmente, entre as principais ferramentas utilizadas em log e monitoramento há: Fluentd, Prometheus, Grafana, Dynatrace, Stack ELK, Stack Prometheus, Kafka (BRANDÃO, 2020).

Diante deste cenário, o presente trabalho propõe a implementação de uma arquitetura de protótipos de microsserviços que utilizará das facilidades disponibilizadas pela tecnologia de contêineres. Com a execução destes protótipos de microsserviços pretende-se aplicar os conceitos de monitoramento para coletar as informações dos logs distribuídos, centralizá-los em uma base, disponibilizá-los em *dashboards,* assim como, realizar o disparo de eventuais alertas para o público-alvo.

1.1 OBJETIVOS

Desenvolver protótipos de microsserviços e disponibilizá-los em um ecossistema ao qual pretende-se aplicar algumas das soluções de monitoramento atualmente disponíveis e disponibilizar os dados resultantes de monitoramento (métricas e logs) em um *dashboard*.

Neste contexto, este trabalho tem como objetivos específicos:

a) realizar o desenvolvimento de protótipos baseado no conceito de arquitetura de microsserviços;

b) gerar informações de métricas dos microsserviços;

c) gerar *logs* dos microsserviços fazendo uso do conceito de *Domain-observed observability* na instrumentação dos mesmos*;*

d) apresentar os resultados do monitoramento em formato de *dashboard*, efetuando o disparo de alertas nas situações previstas.

2 TRABALHOS CORRELATOS

Nesta seção são apresentados alguns trabalhos correlatos com características semelhantes aos principais objetivos do estudo proposto.

No primeiro trabalho, Rodrigues (2018) apresenta o embasamento e uma proposta de conversão de uma arquitetura de sistema monolítico em uma arquitetura de microsserviços.

No segundo trabalho, Silva (2020) aborda um sistema de monitoramento de servidores, tendo neste caso, o foco mais específico no monitoramento da infraestrutura, que aborda os mecanismos da coleta destes dados e disponibilização em uma aplicação web para consulta por parte de seu público.

Já no terceiro trabalho, Assad (2019) demonstra a investigação de soluções para monitoramento de aplicações orientadas a microsserviços, bem como, propõe uma solução para monitoramento baseado nas métricas principais de servidor e infraestrutura.

2.1 *MS-TRICK*: ARQUITETURA DE *MICROSERVICES* APLICADA

Rodrigues (2018) apresenta uma arquitetura de microsserviços. O objetivo do trabalho é reconstruir partes de uma aplicação existente de maneira que ela seja escalável, demonstrando os resultados obtidos com esta alteração. De maneira complementar, foi desenvolvido um aplicativo móvel para visualização da arquitetura. Os microsserviços desenvolvidos para contemplar a arquitetura, foram implementados na linguagem de programação *Java*, mais especificamente utilizando o *SpringBoot*. Para armazenamento de dados, foi utilizado o banco de dados relacional *MySQL*. O aplicativo móvel (interface visual disponibilizada aos usuários) foi desenvolvido utilizando o *Ionic*, *framework* responsável por gerar aplicativos para *Android* e *IOS*.

Rodrigues (2018) afirma que para o desenvolvimento do projeto foi necessário abordar determinados recursos tecnológicos, como servidores de aplicação e integração contínua. A solução proposta por Rodrigues (2018) utiliza o *Spring Boot* para criação dos microsserviços. Segundo Spring (2017), *Spring Boot* é uma ferramenta pronta para produção, responsável por disponibilizar um ambiente de fácil configuração para desenvolvedores de software.

Outro recurso tecnológico utilizado diz respeito ao trabalho de integração contínua que está ligado ao conceito de *DevOps*. A integração e a entrega contínuas são práticas que automatizam o processo de lançamento de software, da fase de criação à fase de implantação. (AWS, 2022).

Como resultados, Rodrigues (2018) destaca que o seu trabalho conseguiu alcançar as seguintes características: escalabilidade, integração contínua, disponibilidade, métricas exclusivas, modularidade, linguagem de programação *Java* e melhoria no processo de manutenção:

a) a escalabilidade foi alcançada com a possibilidade de execução do número de instâncias necessárias para suprir a necessidade dos usuários;

b) a integração contínua foi alcançada com a implementação de conceitos de *DevOps*; pela implementação de um *script* para *Gitlab* é possível realizar a compilação, o *deploy* e a avaliação do código da aplicação;

c) a disponibilidade está diretamente relacionada com a escalabilidade; no *MS-Trick* é possível aumentar o número de instâncias executantes, tornando a aplicação cada vez mais disponível;

d) as métricas exclusivas foram adquiridas pela implementação da arquitetura, estas foram disponibilizadas no aplicativo móvel implementado para melhor visualização da arquitetura;

e) a modularidade, foi alcançada pela implementação de módulos independentes das demais partes do sistema;

f) foi utilizada *Java* como linguagem de programação para adaptação do código existente e criação do novo código;

g) a melhoria no processo de manutenção, é trazida pela possibilidade de diferentes desenvolvedores trabalharem nos módulos individuais da aplicação, bem pela velocidade trazida pela integração contínua.

2.2 *WOLF* MONITOR: SISTEMA DE MONITORAMENTO DE SERVIDORES

Em seu trabalho Silva (2020) apresenta um projeto para automatizar migrações e monitoramento de servidores, cujo objetivo é melhorar a forma como as empresas lidam com serviços e com arquivos de configuração de programas computacionais em seus servidores. Desta forma, não sendo necessário efetuar uma conexão direta no servidor sendo feito todo o controle em um sistema.

O sistema é dividido em três fases. A primeira etapa, um serviço (agente) em execução no servidor da empresa. A segunda etapa *API* da *Web* (sistema que pode receber informações via *internet*) que receberá as informações enviadas por esse servidor. Por último um sistema/aplicativo para computadores que será responsável por exibir as informações transmitidas em telas amigáveis para o usuário.

Seguem descritos abaixo os principais recursos utilizados:

a) *web API* implementada: na fase de implementação Silva (2020) utiliza os padrões de Micro serviços para criação da *API* e *end-points.* Todo o sistema está nas conformidades de *API Gateway*, em que mesma é desenvolvida pensando no *front-end,* para que assim não sejam realizadas consultas desnecessárias e diminua a quantidade de requisições no servidor, permitindo até mesmo uma internet com menor velocidade acessar os dados.

b) banco de dados: como o sistema apresentado no trabalho utiliza de micro serviços, faz-se válido uma diferenciação no banco de dados, que foi separado em quatro partes para que dessa forma o fluxo de dados seja rápido e não ocorra lentidão no sistema nem congestionamento no fluxo de dados (SILVA, 2020).

c) desenvolvimento *desktop:* conforme Silva (2020), o desenvolvimento para *desktop* foi feito para a visualização das informações enviadas pelo serviço do sistema *Windows/Linux* para a *Web Api* de dados. O sistema *desktop* se conecta com o servidor de informações de dados passando por autenticação no *IdentityServer.* Após ser autorizado o mesmo pode visualizar todos os dados que seu usuário tenha permissão para acessar.

d) agente – serviço do *Linux/Windows*: após esse cadastro o usuário solicitará os arquivos necessários para instalação do *agent* (agente) e deverá configurá-lo para utilização. Estes arquivos serão as *dll's* (arquivos de sistema) e executável do programa de monitoramento, ao qual ficará responsável por verificar os dados de todos os itens cadastrados, sempre buscando as informações no servidor de dados da *Web API* (SILVA, 2020).

Silva (2020), conclui que apesar do presente sistema ser voltado a monitoramento apenas de arquivos e serviços, o mesmo abrange outras possibilidades como: banco de dados, serviços de *IIS* (Provedor de dados *Windows*), entre toda e qualquer informação que seja fundamental para o funcionamento correto dos servidores. A utilização deste sistema irá promover uma melhoria significativa na produção das empresas, para que sejam feitos procedimentos de forma rápida e com qualidade, garantindo a funcionabilidade correta dos servidores, mesmo após um serviço de migração ao qual tenha sido necessário o desligamento de todos os serviços.

2.3 MANGUE *BEAT*! INVESTIGANDO SOLUÇÕES PARA MONITORAMENTO DE APLICAÇÕES ORIENTADAS A MICROSSERVIÇOS

O objetivo principal deste trabalho é investigar e expor algumas soluções possíveis e o melhor caso para a(s) sua(s) utilização(ões) no universo do Kubernetes. Em seu trabalho Assad (2019) adota soluções como: a Stack EFK (Elasticsearch, Fluentd, Kibana), Beats do Elasticsearch e apresenta uma solução desenvolvida por ele próprio, o Mangue *Beat*.

Assad (2019) ressalta que existem dois pontos em comum entre todas as soluções que é o banco de dados, o Elasticsearch, e o Kibana um *framework open-source* empregado para visualização de dados do Elasticsearch. O Elasticsearch é um banco de dados não relacional, orientado a documentos, comumente utilizado para fazer consultas complexas em cima de grandes volumes de dados (COIMBRA, 2018).

Com a possibilidade de distribuir as aplicações em diferentes máquinas virtuais (VM) rodando em diferentes ambientes, a forma sugerida pela documentação original do Kubernetes é levantar um agente/operador

em cada *node* onde existem aplicações rodando no *cluster* (KUBERNETES, 2022). Basicamente cada agente fica responsável por coletar os logs de cada *contêiner* que estejam rodando na *vm* e direcionar para um *backend,* que fica responsável por esta centralização dos logs.

Assad (2019), explica que o Fluentd é uma solução *open source* desenvolvida para centralização de logs, que busca armazenar os logs em forma de JSON, como forma de facilitar as consultas e agrupamentos que pode ser feita no banco de dados.

Assad (2019) ainda ressalta que sua configuração não é feita de forma trivial, tendo uma variedade grande de parâmetros que podem ser configurados. Dentre as soluções apresentadas, a mais desafiadora é configurar em ambientes produtivos, devido a diversidade de integrações de plugins e a diferença na forma como são configurados. Por exemplo: configurar onde serão salvas as informações, a ferramenta de alerta e outros vários plugins.

Já sobre o Beats (parte do grupo do Elasticsearch), que é responsável pela tecnologia de monitoramento de aplicações, Assad (2019) explica que é facilmente integrável com o Logstash e com o banco de dados Elasticsearch.

Dentro do agrupamento do Beats há o Metricbeat e o Filebeat. O Metricbeat é a solução voltada para coleta de métricas das aplicações, e o Filebeat é a solução utilizada para monitoramento dos logs dos contêineres. Ambas são *open source* e assim como o Fluentd armazenam os logs em forma de JSON (ASSAD, 2019). Um diferencial do Beats é que o Kibana já vem com um *dashboard* pronto para visualização das métricas coletadas. Basta apenas importá-lo na página inicial do Kibana.

Na contramão das facilidades, Assad (2019) afirma que as ferramentas do Beats consomem cerca de 80 MB de memória a mais que o Fluentd por operador, o que em um servidor com 1000 máquinas pode chegar a 80GB de utilização deste recurso.

Segundo Assad (2019), ao contrário das demais soluções o MangueBeat integra-se com o servidor de métricas mantido pelo Kubernetes, não precisando de um operador rodando em cada *node* do *cluster*. O fato de não ser um operador reduz drasticamente seu consumo de recursos computacionais por *node* no *cluster*, visto que é necessária apenas uma instância do MangueBeat operacional para que o monitoramento seja feito (ASSAD, 2019).

Além disso, Assad 2019 ressalta que o serviço de monitoramento do MangueBeat possibilita o monitoramento de mais de um *cluster* por vez, podendo haver a centralização do agente de monitoramento, e salvando as informações de forma descentralizada. Como o MangueBeat integra-se apenas com o servidor de métricas, seu uso fica restrito apenas a coleta de informações de consumo de CPU, memória e rede das aplicações (ASSAD, 2019).

Um ponto interessante para a solução do MangueBeat é a facilidade de integração com outros microsserviços, para que sejam feitos por exemplo, alertas em caso de anomalias. Para isto, foi desenvolvida uma integração com um TelegramBot, onde são enviados alertas informando anomalias nos microsserviços e são possíveis a execução de algumas operações com base nas métricas coletadas.

Na análise das soluções apresentadas Assad (2019) evidencia que existem diversas formas para implementar e manter o monitoramento de aplicações no cenário sistemas distribuídos em Kubernetes. O autor destaca que, do ponto de vista de alocação dos sistemas de monitoramento em *VMs* é possível identificar duas possibilidades: a instalação de operadores em cada *worker* ou a integração do sistema de monitoramento ao servidor de métricas do Kubernetes.

Em sua conclusão, Assad (2019) salienta que para o desenvolvimento deste trabalho, focou na granularidade das informações coletadas aliando o consumo de recursos computacionais por microsserviços. Ficou claro que agentes de monitoramento em forma de operadores acabam por consumir muito mais recursos do *cluster*, do que, uma solução capaz de centralizar o monitoramento dos recursos de CPU, memória e rede no próprio servidor de métricas do Kubernetes.

3 PROPOSTA DO PROTÓTIPO

Nesta seção serão apresentadas as justificativas para o desenvolvimento do estudo proposto, bem como um quadro comparativo com os trabalhos correlatos, os requisitos funcionais e não funcionais e a metodologia utilizada no desenvolvimento da solução.

3.1 JUSTIFICATIVA

É apresentado a seguir um comparativo de alguns trabalhos que abordam a questão dos microserviços e a ênfase do monitoramento, conforme o quadro 1.

Quadro 1 - Comparativo dos trabalhos correlatos

Características \ Trabalhos Correlatos	Rodrigues (2018)	Silva (2020)	Assad (2019)
Microsserviços	Sim	Sim	Sim
Escalabilidade	Sim	Sim	Sim
Contêiners	Sim	Não	Sim
Kubernetes	Sim	Não	Sim
Monitoramento	Não	Sim	Sim
Métricas	Não	Sim	Sim
Logging	Não	Sim	Não

Fonte: elaborado pelo autor.

Conforme análise do quadro 1, pode-se concluir que os trabalhos abordam de formas similares a utilização e implementação de microsserviços, bem como a utilização da estratégia do monitoramento voltadas para monitorar a arquitetura de microsserviços ou de uma arquitetura convencional de tecnologia da informação. Nesse contexto, o trabalho de Rodrigues (2018), intitulado *MS-Trick,* aborda a conversão e implementação de uma arquitetura de sistema monolítico em microsserviços. Já o trabalho de Silva (2020) aborda o tema voltando o foco para o monitoramento de infraestrutura de sistemas, sendo que se utiliza das vantagens da arquitetura de microsserviços para realizar o propósito de seu trabalho. Por fim, Assad (2019) aplica os conceitos de construção da arquitetura de microsserviços com o foco voltado também para o monitoramento, só que nesse caso, foca em investigar e expor soluções possíveis e o melhor caso para utilização no universo do Kubernetes.

Assim o presente trabalho se torna relevante no sentido em que pretende coletar, tratar e analisar informações de estado de um conjunto de protótipos de microsserviços em tempo real e permitir que o responsável por eles atue antes ou logo após ocorrer alguma falha no sistema monitorado.

Sobre os protótipos de microsserviços em questão: o primeiro irá executar uma funcionalidade de autenticação de acesso; o segundo irá ter as funcionalidades de CRUD (*create*,*read*,*update*,*delete*) para criação de usuários; o terceiro irá ter as funcionalidades de CRUD para criação de alunos; o quarto enviará notificações por e-mail; o quinto enviará notificações por *push*.

O volume de informações a ser tratado neste tipo de problema pode variar muito de acordo com o tipo de monitoramento desejado. Por exemplo, caso se deseje monitorar apenas informações básicas de estado, como uso de memória, processador ou disco, a quantidade de dados a ser enviada por recurso monitorado é relativamente pequena, dependendo da frequência que estas informações foram programadas para serem enviadas. No entanto, este volume pode crescer rapidamente quando se deseja monitorar muitas máquinas ou quando se deseja monitorar ativos que produzam uma grande quantidade de dados por unidade de tempo, como por exemplo, roteadores que enviam metadados (IP de origem, IP de destino, Porta de origem e de destino) de cada pacote roteado (TAVEIRA, 2015).

Para tanto buscou-se uma ferramenta que permita realizar as integrações necessárias de forma assíncrona e em tempo real, possibilitando atualizações distribuídas e entregando mais taxa de transferência (*throughput*): Kafka. Ele é uma plataforma distribuída de *streaming* que atua como uma fila de mensagens de publicação / assinatura, recebendo dados de vários sistemas de origem e disponibilizando-os para vários sistemas e aplicativos em tempo real (KAMILLA PIMENTEL, 2020).

Nesse contexto as principais vantagens da utilização do Kafka são que ele fornece armazenamento durável, o que significa que os dados armazenados nele não podem ser facilmente violados e são altamente escalonáveis, para que possam lidar com um grande aumento de usuários, cargas de trabalho e transações quando necessário (KAMILLA PIMENTEL, 2020).

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os principais requisitos funcionais e não funcionais estão apresentados nos tópicos seguintes, sendo divididos entre as especificações dos protótipos de microsserviços e da organização do dashboard de monitoramento a partir do Kafka.

3.2.1 Protótipos de microsserviços

Os requisitos funcionais são:

a) o protótipo deve permitir o usuário realizar login;

b) o protótipo deve permitir a inclusão, alteração e remoção de usuários do sistema;

c) o protótipo deve permitir a inclusão de usuários em massa;

d) o usuário deve permitir a inclusão, alteração e remoção de alunos do sistema;

e) o protótipo deve permitir a inclusão de alunos em massa;

f) o protótipo deve emitir mensagens de erro caso algum dado esteja incompleto;

g) o protótipo deve permitir a inclusão de avisos para os alunos e usuários;

h) o protótipo deve enviar mensagens via push notification para alunos e usuários;

i) o protótipo deve enviar mensagens via e-mail para alunos e usuários.

Os requisitos não funcionais são:

a) implementar os microsserviços utilizando a ferramenta Spring Boot;

b) utilizar Github para repositório de fontes e desenvolver a integração contínua;

c) utilizar Docker para execução das instâncias dos microsserviços;

d) utilizar Ionic para implementação da interface do protótipo;

e) utilizar MySQL como ferramenta de banco de dados;

f) utilizar o conceito de *Domain-observed observability* na instrumentação dos *logs* dos microsserviços.

3.2.2 Monitoramento

Os requisitos funcionais são:

a) gerar logs para tópico(s) do Kafka;

b) gerar métricas para tópico(s) do Kafka;

c) as mensagens de log devem ser indexadas;

d) os logs devem ser visualizados no *dashboard*;

e) as métricas devem ser visualizadas no *dashboard*.

Os requisitos não funcionais são:

a) utilizar Docker, Docker Composer como contêiner para execução das ferramentas;

b) utilizar Kafka para obter os logs e métricas;

c) utilizar Logstash e ElasticSearch para indexação dos logs;

d) utilizar Kibana para visualização dos dashboards.

3.3 METODOLOGIA

O trabalho será desenvolvido observando as seguintes etapas:

a) realizar levantamento bibliográfico com relação aos assuntos abordados: arquitetura de microsserviços, estratégias de monitoramento, ferramentas utilizadas (Spring Boot, Kafka, Stask ELK (Elasticsearch, Logstash, Kibana), Ionic e das metodologias aplicadas (DEVOPS, Domain-observed observability);

b) realizar levantamento de requisitos: nesta etapa será feito o refinamento dos requisitos tomando como base a pesquisa realizada;

c) realizar especificação da arquitetura dos microsserviços e base de dados;

d) realizar implementação dos microsserviços: será realizada a implementação dos microsserviços utilizando a linguagem Java e Spring Boot. Será desenvolvida com base na arquitetura especificada no passo anterior;

e) realizar instalação e configuração das ferramentas de monitoramento;

f) realizar definição dos indicadores e métricas a serem gerados e ou coletados;

g) realizar configuração da interface para a visualização dos dashboards e métricas dos serviços monitorados;

h) realizar os testes dos protótipos de microsserviços, aplicando o monitoramento em situações diversas.

As etapas serão realizadas nos períodos relacionados no Quadro 2.

Quadro 2 - Cronograma

	2023									
	fev.		mar.		abr.		mai.		jun.	
etapas / quinzenas	1	2	1	2	1	2	1	2	1	2
Realizar levantamento bibliográfico										
Realizar levantamento de requisitos										
Realizar especificação da arquitetura dos microsserviços										
Realizar implementação dos microsserviços										
Realizar instalação e configuração das ferramentas de monitoramento										
Realizar definição dos indicadores e métricas										
Realizar configuração da interface para a visualização dos dashboards										
Realizar os testes dos protótipos de microsserviços e do monitoramento										

Fonte: elaborado pelo autor.

4 REVISÃO BIBLIOGRÁFICA

Conforme descreve Ibm Cloud Education (2021), a arquitetura de microsserviços é uma abordagem na qual um único aplicativo é composto de muitos serviços menores que são implementáveis de forma independente e têm acoplamento fraco. Esses serviços normalmente:

a) têm a própria *stack* de tecnologia, incluindo o modelo de banco de dados e gerenciamento de dados;

b) comunicam-se sobre uma combinação de APIs de REST, fluxos de eventos e *brokers* de mensagens;

c) são organizados por recurso de negócios, com a linha separando os serviços que, muitas vezes, são chamados de contexto delimitado.

Além de suas características arquitetônicas, seu valor pode ser mais facilmente compreendido por meio de benefícios corporativos e de negócios bastante simples (IBM CLOUD EDUCATION, 2021):

a) O código pode ser atualizado mais facilmente, novos recursos ou funcionalidades podem ser incluídos sem mudar nenhuma parte do aplicativo;

b) As equipes podem usar *stacks* e linguagens de programação diferentes para componentes diferentes;

c) É possível ajustar a escala dos componentes de forma independente, reduzindo o desperdício e o custo associado ao ajuste de escala de aplicativos inteiros, como nos casos em que há um único recurso sobrecarregado.

Para Fowler (2017), um microsserviço pronto para produção é adequadamente monitorado. Um monitoramento adequado é uma das partes mais importantes de construir um microsserviço pronto para produção e garantir maior disponibilidade do microsserviço.

Alguns dos requisitos para construir um microsserviço adequadamente monitorado são (FOWLER, 2017):

a) *logging* e rastreamento adequado em toda a pilha;

b) *dashboards* bem projetados que sejam fáceis de entender e reflitam com precisão a saúde do serviço;

c) alertas eficazes e acionáveis acompanhados de roteiros;

d) implementar e manter uma rotação das equipes de prontidão.

No caso dos dashboards, deve apresentar informações das métricas principais para microsserviços, conforme destaca Fowler (2017). Dentre as métricas principais de microsserviços, podemos ter: métricas específicas da linguagem; disponibilidade; *Service Level Agreement* (SLA); latência; sucesso do *endpoint*; respostas do *endpoint*; tempos de resposta do *endpoint*; clientes; erros e exceções; dependências.

Atualmente, entre as principais ferramentas utilizadas em log e monitoramento há:

a) Fluentd: um coletor de dados *open source* que permite a unificação da coleta de dados e seu consumo para melhor entendimento dos dados (BRANDÃO, 2020);

b) Prometheus: uma aplicação para monitoramento de eventos e alertas. Ele captura métricas em tempo real em um banco de dados temporal. No entanto não conta com uma ferramenta de exibição dos seus dados (BRANDÃO, 2020);

c) Stack ELK: Logstash sendo responsável por coletar os logs de várias fontes, processá-los e em seguida e em seguida enviá-los para o Elasticsearch que por sua vez faz as buscas desses logs de de forma mais performática possível, utilizando um banco de dados noSQL (mongoDB ou Cassandra) para persistir os dados. O Kibana é uma interface que facilita a visualização desses dados em forma de lista e gráficos (BRANDÃO, 2020);

d) Apache Kafka: O Apache Kafka é uma plataforma distribuída de transmissão de dados que é capaz de publicar, subscrever, armazenar e processar fluxos de registro em tempo real. O Apache Kafka é uma alternativa aos sistemas de mensageria empresariais tradicionais. Inicialmente, ele era um sistema interno desenvolvido pela LinkedIn para processar 1,4 trilhão de mensagens por dia. Mas agora é uma solução de transmissão de dados open source aplicável a variadas necessidades empresariais (REDHAT, 2017).

REFERÊNCIAS

ASSAD, Lucas Serra da Cunha. **Mangue Beat! Investigando soluções para Monitoramento de Aplicações Orientadas a Microsserviços**. 2019. 46 f. TCC (Graduação) - Curso de Bacharelado em Sistemas de Informação, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2019.

BRANDÃO, Daniel. **Log e Monitoramento de Microsserviços**: estado da arte, problemas e soluções com autoscaling multi-nuvem. Estado da arte, Problemas e Soluções com Autoscaling multi-nuvem. 2020. Disponível em: https://medium.com/@dcruzb/log-e-monitoramento-de-microsservi%C3%A7os-11c465349a77. Acesso em: 19 set. 2022.

COIMBRA, Vinicius. **Comparando: Elasticsearch vs MongoDB**. 2018. Disponível em: https://medium.com/data-hackers/comparando-elasticsearch-vs-mongodb-4b5932c613d9. Acesso em: 25 set. 2022.

DOBIES, Jason; WOOD, Joshua. **Operadores do Kubernetes**: automatizando a plataforma de orquestração de contêineres. São Paulo: Novatec O'Reilly, 2020. 160 p.

DEVMEDIA. **Simplificando a computação distribuída**. 2014. Disponível em: https://www.devmedia.com.br/hazelcast-simplificando-a-computacao-distribuida/31628. Acesso em: 15 abr. 2022.

IBM CLOUD EDUCATION. **Microsserviços**. 2021. Disponível em: https://www.ibm.com/br-pt/cloud/learn/microservices. Acesso em: 10 out. 2022.

FOWLER, Susan J.. **Microsserviços prontos para produção**: construindo sistemas padronizados em uma organização de engenharia de software. São Paulo: Novatec O'Reilly, 2017. 218 p.

KUBERNETES. **Kubernetes the official documentation:** Logging Architecture. 2022. Disponível em: https://kubernetes.io/docs/concepts/cluster-administration/logging/#cluster-level-log. Acesso em: 25 set. 2022.

OPUS SOFTWARE. **Micro Serviços:** qual a diferença para a arquitetura monolítica?. Qual a diferença para a Arquitetura Monolítica?. 2021. Disponível em: https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/. Acesso em: 19 set. 2022.

KAMILLA PIMENTEL (Brasil). **[TIMEBRA] MÉTRICAS IMPORTANTES PARA O MONITORAMENTO KAFKA**. 2020. Disponível em: https://blogac.me/timebra-metricas-importantes-para-o-monitoramento-kafka/. Acesso em: 26 set. 2022.

REDHAT. **O que é Apache Kafka?** 2017. Disponível em: https://www.redhat.com/pt-br/topics/integration/what-is-apache-kafka. Acesso em: 29 set. 2022.

RODRIGUES, Ariel Rai. **MS-TRICK**: arquitetura de microservices aplicada. 2018. 85 f. TCC (Graduação) - Curso de Sistemas de Informação – Bacharelado, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2018.

ROSSI, Rodrigo. **Entrando no Mundo de Microsserviços**: parte 1. 2021. Disponível em: https://www.linkapi.solutions/blog/entrando-no-mundo-de-microsservicos-parte-1. Acesso em: 15 abr. 2022.

SILVA, Aléff Moura da. **WOLF MONITOR**: sistema de monitoramento de servidores. 2020. 39 f. TCC (Graduação) - Curso de Ciências da Computação, Centro Tecnológico, Centro Universitário Unifacvest, Lages, 2020.

TAVEIRA, Luís Felipe Rabello. **Monitoramento de Ambientes Computacionais Distribuídos em Tempo Real**. 2015. 57 f. Monografia (Especialização) - Curso de Bacharelado em Engenharia da Computação, Departamento de Ciência da Computação, Universidade de Brasília, Brasília, 2015. Cap. 1. Disponível em: https://bdm.unb.br/bitstream/10483/10154/1/2015_LuisFelipeRabelloTaveira.pdf. Acesso em: 26 set. 2022.

FORMULÁRIO DE AVALIAÇÃO BCC – PROFESSOR AVALIADOR – PRÉ-PROJETO

Avaliador(a): Francisco Adell Péricas

Atenção: quando o avaliador marcar algum item como atende parcialmente ou não atende, deve obrigatoriamente indicar os motivos no texto, para que o aluno saiba o porquê da avaliação.

	ASPECTOS AVALIADOS	Atende	atende parcialmente	não atende
ASPECTOS TÉCNICOS	1. INTRODUÇÃO O tema de pesquisa está devidamente contextualizado/delimitado?			
	O problema está claramente formulado?			
	2. OBJETIVOS O objetivo principal está claramente definido e é passível de ser alcançado?			
	Os objetivos específicos são coerentes com o objetivo principal?			
	3. TRABALHOS CORRELATOS São apresentados trabalhos correlatos, bem como descritas as principais funcionalidades e os pontos fortes e fracos?			
	4. JUSTIFICATIVA Foi apresentado e discutido um quadro relacionando os trabalhos correlatos e suas principais funcionalidades com a proposta apresentada?			
	São apresentados argumentos científicos, técnicos ou metodológicos que justificam a proposta?			
	São apresentadas as contribuições teóricas, práticas ou sociais que justificam a proposta?			
	5. REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO Os requisitos funcionais e não funcionais foram claramente descritos?			
	6. METODOLOGIA Foram relacionadas todas as etapas necessárias para o desenvolvimento do TCC?			
	Os métodos, recursos e o cronograma estão devidamente apresentados e são compatíveis com a metodologia proposta?			
	7. REVISÃO BIBLIOGRÁFICA Os assuntos apresentados são suficientes e têm relação com o tema do TCC?			
	As referências contemplam adequadamente os assuntos abordados (são indicadas obras atualizadas e as mais importantes da área)?			
ASPECTOS METODOLÓGICOS	8. LINGUAGEM USADA (redação) O texto completo é coerente e redigido corretamente em língua portuguesa, usando linguagem formal/científica?			
	A exposição do assunto é ordenada (as ideias estão bem encadeadas e a linguagem utilizada é clara)?			