UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ESPECIALIZAÇÃO EM ARQUITETURA E GERENCIAMENTO DE INFRAESTRUTURA DE TI

RAFAEL ONETTA ARAÚJO

UMA REVISÃO TEÓRICA SOBRE A ARQUITETURA DE MICROSSERVIÇOS

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2021

RAFAEL ONETTA ARAÚJO

UMA REVISÃO TEÓRICA SOBRE A ARQUITETURA DE MICROSSERVIÇOS

Monografia de Especialização, apresentada ao Curso de Especialização em Arquitetura e Gestão de Infraestrutura de TI, do Departamento Acadêmico de Eletrônica – DAELN, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Dr. Joilson Alves Junior

CURITIBA
2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

Diretoria de Pesquisa e Pós-Graduação
Departamento Acadêmico de Eletrônica
Curso de Especialização em Arquitetura e Gestão de Infraestrutura de TI

TERMO DE APROVAÇÃO

UMA REVISÃO TEÓRICA SOBRE A ARQUITETURA DE MICROSSERVIÇOS

RAFAEL ONETTA ARAÚJO

Esta monografia foi apresentada em 20 de dezembro de 2021 como requisito parcial para a obtenção do título de Especialista em Arquitetura e Gestão de Infraestrutura de TI. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Joilson Alves Júnior
Orientador

_____
Prof. Dr. Kleber Kendy Horikawa Nabas
Membro titular

_____
Prof. M. Sc. Omero Francisco Bertol
Membro titular

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso –

Dedico este trabalho à minha família,
amigos, colegas de curso
e professores da UTFPR.

AGRADECIMENTOS

Agradeço aos meus amigos pela ajuda na escolha do tema, ao meu orientador Joilson Alves Júnior pela ajuda na construção desta monografia e a todos os meus colegas e amigos de curso por todo o aprendizado compartilhado ao longo dele. Também, agradeço a minha família por me apoiar não só nesta, mas em todas as fases importantes da minha vida.

RESUMO

ARAÚJO, Rafael Onetta. **Uma Revisão Teórica sobre a Arquitetura de Microsserviços**. 2021. 27 p. Monografia de Especialização em Arquitetura e Gestão de Infraestrutura de TI, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

Neste trabalho, serão apresentados os conceitos, diferenças e vantagens de se adotar como metodologia padrão a arquitetura de microsserviços em ambientes computacionais ao invés dos modelos padrões centralizados, que são mais conhecidos como monólitos. Também serão abordadas as definições de padrões de alta disponibilidade, confiabilidade e escalabilidade de microsserviços, bem como os conceitos de implantação, testes de lógica e carga, monitoramento e princípios importantes sobre documentações.

Palavras-chave: Infraestrutura. Microsserviços. Confiabilidade. Escalabilidade. Disponibilidade.

ABSTRACT

ARAÚJO, Rafael Onetta. **A Theoretical Review of Microservices Architecture**. 2021. 27 p. Monografia de Especialização em Arquitetura e Gestão de Infraestrutura de TI, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

In this work, the concepts, differences and advantages of adopting microservices architecture in computational environments as standard methodology will be presented instead of centralized standard models, which are better known as monoliths. It will also cover definitions of standards for high availability, reliability and scalability of microservices, as well as the concepts of deployment, logic and load testing, monitoring, and important principles about documentation.

Keywords: Infrastructure. Microservices. Reliability. Scalability. Availability.

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho abordará as importâncias e considerações positivas da adoção de uma arquitetura de microsserviços diversamente aos modelos tradicionais que possuem como objetivo manter os sistemas centralizados, conhecidos como monólitos, o que por si só, é uma porta de entrada para problemas de alta disponibilidade e confiabilidade dos sistemas responsáveis pela continuidade de um determinado negócio ou organização.

Todos os sistemas estão suscetíveis a falhas, sejam de performance, hardware ou problemas de tráfego de rede, e justamente por conta disto, os monólitos que utilizam uma arquitetura centralizada, são os que mais sofrem nestes momentos, visto que na maioria das vezes, os monólitos concentram em si a grande parte dos principais sistemas de uma organização, o que significa que, se o monólito apresentar algum problema, a organização ficará com os seus principais sistemas inoperantes.

Por outro lado, a abordagem da arquitetura de microsserviços traz como ponto principal a descentralização e distribuição destes sistemas em ambientes separados, podendo estes ser sistemas iguais ou diferentes, e que mesmo assim, se comunicam entre si normalmente com um objetivo em comum.

1.1 JUSTIFICATIVA

A importância da abordagem do tema de arquitetura de microsserviços se dá sobre a necessidade de empresas e organizações de possuírem agilidade e maior segurança na implantação e na manutenção de suas infraestruturas, sistemas e aglomerados de códigos, garantindo a continuidade de seus negócios e evitando assim, a indisponibilidade de seus sistemas.

1.2 OBJETIVOS

O presente trabalho terá como objetivo central a conceituação de uma arquitetura de microsserviços, trazendo seus principais pontos e vantagens de uso, e, estes conceitos serão comparados com abordagem centralizadas, ou seja, as estruturas conhecidas como monolíticas.

Também como objetivo secundário, serão apresentados os conceitos de escalabilidade, alta disponibilidade e confiabilidade, adjetivos comuns quando se é utilizado uma arquitetura de microsserviços dentro de uma infraestrutura de sistemas de uma determinada empresa.

2 DESENVOLVIMENTO

2.1 FUNDAMENTAÇÃO DOS MICROSSERVIÇOS

Conforme Fowler (2017), com o avanço da tecnologia, nos últimos anos, o setor de tecnologia observou uma rápida mudança na arquitetura de sistemas que é aplicada sobre metodologias distribuídas, o que teve efeito sobre grandes corporações como, por exemplo, Amazon, Netflix, Twitter e Uber, a adotar uma arquitetura de microsserviços e abandonando a construção de sistemas monolíticos.

> Microservices are an architectural style largely based on decoupled autonomous services that can be developed, deployed and operated independently of each other (MAYER; WEINREICH, 2017, p. 66).

Segundo Fowler (2017), os conceitos e definições que circundam o meio dos microsserviços não são novas, a aplicação contemporânea da arquitetura de microsserviços é atual, e a adoção dessa metodologia tem como objetivo sanar os problemas e desafios de escalabilidade, falta de eficiência, problemas no desenvolvimento e a dificuldade de endossar novas tecnologias no projeto. A implantação de uma arquitetura de microsserviços é um passo natural do processo para tornar uma aplicação escalável, o que leva em consideração a desconstrução e descentralização de aplicações monolíticas em microsserviços pelas questões de escalabilidade, disponibilidade e eficiência.

2.2 DE MONÓLITOS A MICROSSERVIÇOS

Segundo Fowler (2017), a maioria das aplicações disponíveis hoje em nosso meio na internet pode ser divididas em três partes, que são elas: o lado *frontend*, ou seja, o lado do usuário, o *backend*, que é onde os processos do sistema são executados e por último, uma camada de armazenamento destas informações. As solicitações e requisições feitas pelo usuário são realizadas no *frontend*, que na sequência, são processadas e modeladas pelo *backend* e enviadas à um sistema de armazenamento, como um banco de dados.

Na sequência, Fowler (2017) afirma que existem três métodos diferentes para combinar os elementos descritos acima para desenvolver uma aplicação, onde boa

parte das aplicações armazena os dados dos dois primeiros elementos, que são o *frontend* e *backend* em repositórios de código, e deixam o terceiro elemento, que é o banco de dados, separado dos dois primeiros. Há aplicações também que separam em repositórios distintos os códigos responsáveis por sustentar o *frontend* e *backend* da aplicação, mantendo ainda o elemento banco de dados separado. As aplicações de empresas que estão iniciando no mercado são pequenas e de baixa complexidade, e, são sustentadas por uma equipe pequena de desenvolvedores. Entretanto, conforme a empresa cresce, a aplicação precisará acompanhar o ritmo da empresa, o que denota a necessidade de mais programadores para sustentar a aplicação. Novos recursos sempre serão adicionados na aplicação, e com isso, três consequências acompanham o crescimento.

Ainda em Fowler (2017), a primeira consequência é o aumento significativo da carga de trabalho, que seria todo o trabalho operacional realizado sobre uma determinada aplicação. A segunda consequência é causada em virtude da adição de novos recursos na aplicação, tornando-a mais complexa, e por fim, a terceira consequência é o necessário dimensionamento horizontal ou vertical da aplicação. Entende-se como dimensionamento horizontal a adição de novos servidores ou instâncias com uma cópia da aplicação sejam adicionados, e após isso, o tráfego desta aplicação será distribuído de forma inteligente com balanceadores de carga ou *load balancers*. Já o dimensionamento vertical é o aumento de recursos de um determinado servidor conforme a necessidade da aplicação, seja o número de processadores disponíveis para uso, quantidade de memória ou armazenamento de disco. Acompanhando esse crescimento da aplicação e de pessoas responsáveis por ela, a complexidade aumentará também, e com isso, inúmeros testes da aplicação deverão ser escritos e executados para garantir que qualquer alteração realizada no código não comprometa as milhares de linhas de código da aplicação. Manutenções e alterações nesses tipos de aplicações acabam se tornando praticamente inviáveis, causando problemas e defasagem técnica da aplicação, que por conta disso e de seu ciclo de vida, são conhecidas como monólitos.

> Most of the Monolithic Architecture uses the same code, programming language, framework and platform. And, during development, all components are combined into a large single application to run in the system (CHEN *et al.*, 2021, p. 110).

Prosseguindo, Fowler (2017) ainda afirma que nem todas as aplicações do tipo monólito são ruins ou sofrem exatamente os problemas listados, mas todas as características de uma aplicação moldada na arquitetura de monólito vão contra os princípios de uma arquitetura de microsserviços, que prezam pela escalabilidade e alta disponibilidade de suas aplicações.

> Microservitization is rapidly increasing; many distributed and cloud-based systems have evolved from monolithic to microservices architectures (HASSAN; ALI; BAHSOON, 2017, p.1).

3 A ARQUITETURA DOS MICROSSERVIÇOS

Conforme Fowler (2017), comparando com o modelo de arquitetura monolítica, a abordagem da arquitetura de microsserviços não são muito diferentes entre si. Um microsserviço, assim como um serviço baseado na abordagem monolítica, também será divido em três componentes, sendo eles o *frontend*, *backend* e um sistema de armazenamento dos dados.

Segundo Fowler (2017), diferentemente, na arquitetura de microsserviços, o *frontend* do sistema será sustentado por uma API (*Application Programming Interface*, ou interface de programação de aplicação), que nada mais são que pontos de acesso estáticos dentro de um sistema, onde cada ponto é responsável por realizar uma tarefa do sistema, seja ela incluir um novo registro no banco de dados, realizar a alteração de um cadastro ou exclusão de algum dado específico. Por exemplo, é possível definir um *endpoint* ou ponto de acesso da API denominado *criar_usuário*, onde esse microsserviço será responsável por criar usuários para o acesso de algum determinado sistema.

> Microservice Architecture and Monolithic Architecture [4] are generally composed of codes to form a complete application service. The difference between the two is the way they build. Most of the Monolithic Architecture uses the same code, programming language, framework and platform. And, during development, all components are combined into a large single application to run in the system. On the other hand, the microservice architecture divides the application into several independent microservices, and then uses the communication interface e.g. API (Application Programming Interface) to connect the microservices one by one to construct a complete application service (CHEN *et al.*, 2021, p. 110).

De acordo com Fowler (2017), os *endpoints* são apenas separados na arquitetura e na teoria, a fim de dar uma maior clareza da estrutura como um todo, mas na prática, todos esses *endpoints* estão alocados um ao lado do outro, trabalho em conjunto como uma estrutura de *backend*. A ideia neste caso é extrair cada função de um sistema centralizado ou monolítico e segregá-los em pequenas funções, ou seja, os microsserviços.

Em conformidade com Fowler (2017), essa arquitetura e abordagem são estritamente necessárias para que os microsserviços funcionem e bem e se comuniquem com clareza, fazendo com que, em conjunto, esses microsserviços

consigam executar ações que antes eram somente executadas por sistemas centralizados.

> To provide the required levels of flexibility and adaptation demanded by modern systems, a set of technologies have to be jointly used. Technologies such as containers, container-orchestrators, horizontal and vertical auto-scalers, load balancers, cache applications and API gateways are commonly seen as part of microservices ecosystem (SÁ, 2020, p. 449).

4 ECOSSISTEMA DOS MICROSSERVIÇOS

Segundo Fowler (2017), os microsserviços em si não são alocados de forma isolada, pois o ambiente onde eles são construídos e executados são de fato, o local onde eles ficarão alocados. Quando esse ambiente é projetado de forma correta, os microsserviços ficam separados de toda a infraestrutura, ou sejam, as camadas de rede, do hardware, de processos automatizados de implantação de código, testes de software e de estruturas de balanceamento de carga. Todos esses elementos citados anteriormente fazem parte a infraestrutura do ecossistema de microsserviços, e como padrão, a infraestrutura necessita ser tolerante a falhas e confiável, bem como escalável e estável, garantindo a plena operação dos microsserviços.

Conforme Fowler (2017), a infraestrutura em si será responsável por sustentar todo o ecossistema de microsserviços, onde o ecossistema pode ser divido em quatro camadas, são elas:

- Camada 1: Hardware – máquinas e servidores.
- Camada 2: Comunicação – rede.
- Camada 3: Plataforma de Aplicação – ferramentas úteis.
- Camada 4: Microsserviços – os pontos de acesso.

4.1 CAMADA DE HARDWARE

Afirmado por Fowler (2017), esta é camada onde estão as máquinas e servidores físicos propriamente ditos, nos quais todos os microsserviços e ferramentas fundamentais serão executados. Os servidores são alocados em *racks,* dentro de estruturas conhecidas como *datacenters*, onde essas estruturas são refrigeradas por equipamentos de ponta, a fim de manter uma temperatura operacional adequada para os servidores e equipamentos, bem como, sistema de energia redundantes à prova de falhas e indisponibilidades.

Conforme Fowler (2017), empresas que possuem seus próprios *datacenters* tem como responsabilidade escolher o hardware mais adequado as necessidades de seu negócio, e, otimizar este hardware quando necessário. Do contrário, caso a empresa escolha utilizar servidores de *datacenter* em nuvem, que é a abordagem mais comum utilizada atualmente, como a AWS, Azure ou Google Cloud, a definição

de escolha do hardware desejado é limitado às opções fornecidas pelo provedor. Neste momento, escolher entre uma estrutura de *datacenter* local ou em nuvem não é uma tarefa fácil, pois devem ser levados em consideração pontos importantes como o custo, disponibilidade, confiabilidade e as despesas operacionais.

De acordo com Fowler (2017), a gerência desses servidores, sejam em *datacenters* locais ou em nuvem, é definida pela camada de hardware. Cada servidor precisará ter um sistema operacional instalado, que será definido pela empresa conforme suas necessidades. Após o sistema operacional ser instalado, pode-se utilizar uma ferramenta de gestão de configuração para instalar todas as dependências e aplicações necessárias para aquele sistema operacional.

4.2 CAMADA DE COMUNICAÇÃO

Conforme Fowler (2017), essa camada se traduz basicamente como a camada de rede, onde ela estará presente e espalhada pelas outras camadas do ecossistema, pois é através dessa camada que toda a comunicação os serviços e demais camadas é realizada. Nesta camada, é possível notar a presença de alguns componentes, como a rede em si, serviços de DNS *(Domain Name System),* os *endpoints* das APIs e a estruturas de balanceamento de carga.

De acordo com Fowler (2017), os microsserviços utilizam a camada de comunicação ou também conhecido como camada de rede para se comunicar entre si, trocando mensagens entre seus *endpoints* de API, trazendo a ideia de que, usando protocolos específicos, um microsserviço realizará o envio via rede de dados formatados conforme um padrão desejado para outro *endpoint* de API de outro microsserviço. Neste meio campo, existe um elemento chamado *broker* que atuará como um intermediário entre os microsserviços, garantindo a entrega dos dados.

É afirmado por Fowler (2017) que o protocolo mais comum utilizado na comunicação de microsserviços é o HTTP *(Hypertext Transfer Protocol)*, onde os dados são enviados e recebidos de *endpoints* específicos, utilizando métodos HTTP como o GET e POST. O formato geralmente utilizado para os dados é do tipo JSON, que depois de formatados, são enviados via protocolo HTTP.

Em conformidade com Fowler (2017), tratando de questões de balanceamento de carga, é possível observar que, em arquiteturas centralizadas ou monolíticas, o tráfego dos dados precisará ser enviado para a aplicação de destino,

e após isso, replicada e distribuída nos demais servidores que executam essa mesma aplicação. No entanto, em uma arquitetura de microsserviços, o tráfego de rede precisará ser direcionado para muitas aplicações, e então, distribuídos adequadamente para cada servidor que estiver sustentando um microsserviço específico. Para realizar essa tarefa de forma adequada, a arquitetura de microsserviços requer uma estrutura de balanceamento de carga para realizar a correta distribuição de informações dentro do ambiente.

4.3 CAMADA DE PLATAFORMA DE APLICAÇÃO

Conforme Fowler (2017), esta camada possui presente todas as ferramentas e serviços internos importantes que são independentes dos microsserviços. A camada de plataforma de aplicação precisa ser projetada para que o desenvolvimento de microsserviços seja facilitado para as equipes responsáveis, onde estas equipes se preocuparão unicamente com a construção dos microsserviços, excluindo tarefas desnecessárias a essas equipes, poupando tempo para focar no que é importante, ou seja, o desenvolvimento de microsserviços. Pode-se levar em consideração uma plataforma de aplicação de qualidade aquela que possui as ferramentas certas para os desenvolvedores, processos de desenvolvimento padronizados, sistemas de gerenciamento de código, bem como sistemas que cuidarão da construção, empacotamento, distribuição, validação e implantação deste código.

De acordo com Fowler (2017), quando o processo de desenvolvimento de microsserviços é padronizado, o processo se tornará mais eficaz e tolerante a falhas ou erros, melhorando a entrega final. A automação também é muito importante, pois será responsável por executar tarefas repetitivas e melhorar o ciclo de desenvolvimento dos microsserviços. Um dos primeiros requisitos é um sistema de controle de versão de código centralizado, no qual qualquer código possa ser receber etiquetas de versão específicas, bem como ser armazenado, rastreado e pesquisado de uma forma mais prática. Neste caso, é utilizado como por exemplo, o GitHub ou similares, como GitLab.

Prosseguindo com Fowler (2017), o segundo requisito é um ambiente de desenvolvimento separado do ambiente de produção. O ambiente de desenvolvimento deverá ser igual ou similar ao ambiente de produção, tanto em

questões de recursos disponíveis como a arquitetura em si. Com isso, no ciclo de desenvolvimento, qualquer alteração ou modificação que for realizada no código, deverá ser testada e avaliada de forma automatizada através de processos de integração contínua, e para isso, existem muitas ferramentas disponíveis no mercado, como o Jenkins. Essa automação diminui os erros pois após implantado, o contato humano com o processo não será mais necessário.

Segundo Fowler (2017), juntando todos os requisitos definidos acima, pode ser criado um elemento conhecimento *pipeline* ou esteira de implantação, onde todo o processo de instalação de novas versões do código do microsserviço podem ser implantadas de forma automatizada, bem como as etapas de construção do código e testes dele. A *pipeline* torna a implantação muito mais simplificada, visto que, por exemplo, se uma determinada empresa possui inúmeros microsserviços distribuídos sobre sua infraestrutura, a implantação de novas versões ou alterações de código sem um processo de *pipeline* podem transformar o processo em um verdadeiro problema.

Ainda segundo Fowler (2017), após a descritiva dos demais elementos acima, é necessário levar em consideração também a importância de monitorar esses recursos, onde todos os microsserviços devem ser monitorados e possuir implantados sistemas de *logging* ou registro de informações, para que seja possível catalogar mensagens de erro ou de sucesso, tornando a resolução de possíveis problemas ou *bugs* nos microsserviços mais prática. Um monitoramento em tempo real dos microsserviços traz como vantagem a visualização da saúde e o status de usabilidade do microsserviço, tornando o processo mais dinâmico.

> The independent nature of service development and evolution in a microservice-based system and its nature as a highly dynamic distributed system require infrastructure support in terms of monitoring and managing such services (MAYER; WEINREICH, 2017, p. 66).

4.4 CAMADA DE MICROSSERVIÇOS

Conforme Fowler (2017), esta é a camada onde estão alocados os microsserviços em si, completamente isolados das três camadas anteriormente mencionadas, ou seja, nesta camada, os microsserviços são separados da camada de hardware, comunicação, rede ou balanceamento de carga. Os elementos que não são separados da camada de microsserviços são as configurações de ferramentas específicas para cada microsserviço.

5 DISPONIBILIDADE DE PRODUÇÃO

Segundo Fowler (2017), a adoção de uma arquitetura de microsserviços traz consigo uma maior liberdade para os desenvolvedores, entretanto, para garantir uma alta disponibilidade do ecossistema de microsserviços como um todo, é necessário que cada microsserviço, individualmente, possua padrões operacionais e organizacionais, bem como uma arquitetura bem definida.

Conforme Fowler (2017), á medida que uma empresa ou organização cresce, a aplicação e os microsserviços precisam acompanhar essa evolução, sendo necessário redimensioná-los. Caso no início de definição de projeto, tenha sido selecionado a abordagem de uma arquitetura monolítica, o desenvolvedor estará limitado na questão de escolha da linguagem de desenvolvimento, as bibliotecas disponíveis para uso, ferramentas disponíveis, entre outros. Já numa abordagem de arquitetura de microsserviços, o desenvolvedor estará livre para escolher a linguagem que deseja trabalhar, o sistema de banco de dados que será utilizado ou as ferramentas de desenvolvimento, trazendo como ponto central no trabalho do desenvolvedor a tarefa de construir um microsserviço que realize uma única função, e que a realize de forma esplendida.

Ainda segundo Fowler (2017), dentro dos ecossistemas de microsserviços, o sucesso de um serviço pode ser medido através de sua disponibilidade. Serviços que possuem alta disponibilidade são mais confiáveis, pois devido a sua alta disponibilidade, passam pouquíssimo ou nenhum momento indisponível, entretanto, serviços que estão suscetíveis a problemas de disponibilidade tendem a ser menos confiáveis, pois se estão indisponíveis, estão afetando de forma negativa diretamente a atividade de alguma empresa ou organização.

6 PADRÕES DE DISPONIBILIDADE DE PRODUÇÃO

Segundo Fowler (2017), o conceito básico de disponibilidade em um ambiente de produção é quando uma aplicação ou microsserviço está pronto e adequado para a produção, sendo por sua vez, confiável de que irá executar suas tarefas com alto desempenho e qualidade, garantindo a continuidade do fluxo das aplicações. É necessário entender de forma clara quais os requisitos que um serviço deve cumprir para estar pronto para atender ao tráfego do ambiente produção. Para atender essa questão, existem oito princípios quantificáveis que, quando utilizados em conjunto, tendem a atender a necessidade de garantir a disponibilidade em produção. Os requisitos são: estabilidade, confiabilidade, escalabilidade, tolerância a falhas, desempenho, monitoramento e documentação.

Conforme Fowler (2017), fazer o uso de somente um ou alguns dos requisitos citados acima não sustentarão argumentos sólidos para classificar um determinado microsserviço ou aplicação como apto para a disponibilidade em ambiente de produção, entretanto, a construção de microsserviços levando em conta estes oito requisitos tornam a arquitetura mais sólida, garantindo o sucesso do serviço ou aplicação na retenção do tráfego de produção.

6.1 ESTABILIDADE

Segundo Fowler (2017), se traduz como estabilidade o conceito de trabalhar de forma responsável com alterações, atualizações ou disponibilidade de novos recursos no ambiente de produção. Entende-se como um microsserviço estável aquele que, mesmo passando por alterações ou modificações, se mantêm estável e trabalhando com o padrão de qualidade esperado, e para garantir isso, é possível utilizar abordagens como um ciclo de desenvolvimento e implantação de novos códigos estável, bem como procedimentos de introdução e descontinuação que visem a estabilidade do microsserviço. Com isso, é possível atenuar problemas que possam vir a ocorrer durante algum procedimento de implantação ou atualização de um microsserviço.

6.2 CONFIABILIDADE

Conforme Fowler (2017), a confiabilidade de um microsserviço é definida pelo nível de qualidade em que ele atende o tráfego do ambiente de produção. O requisito de estabilidade por si só não é suficiente para suprir a disponibilidade de um microsserviço, portanto, se o conceito de estabilidade está ligado em diminuir os efeitos negativos que podem ser gerados com atualizações do sistema e a confiabilidade está ligada diretamente na confiança que aquele sistema transmite, é correto considerar que os dois conceitos estão interligados entre si.

Para Fowler (2017), todo e qualquer requisito de estabilidade traz consigo um requisito de confiabilidade, pois ao agregar confiabilidade em nossos sistemas e microsserviços, o requisito de disponibilidade é atingido também, como consequência da ação de confiabilidade. Um exemplo de confiabilidade seria que, durante um processo de automação de implantação de código em ambiente de produção, todos os testes sejam devidamente executados, a fim de garantir que, o código que está sendo alocado em produção é confiável, não ferindo o quesito de disponibilidade.

6.3 ESCALABILIDADE

Segundo Fowler (2017), a disposição de tráfego em microsserviços dificilmente será estática ou constante, e uma das principais características de um microsserviço é que ele pode ser escalável, ou seja, se adequar conforme a demanda de recursos do momento, garantindo a estabilidade e disponibilidade do serviço. Microsserviços que não são escaláveis tendem a apresentar problemas de latência, redução no percentual de disponibilidade e nos piores casos, o aumento no incidente de interrupções ou quedas no serviço.

Para Fowler (2017), uma boa estratégia de escalabilidade precisa ser definida para os microsserviços em produção, pois com isso, não apenas em momentos que o microsserviço necessite de mais recursos para funcionar corretamente para garantir a continuidade do negócio, o microsserviço também estará pronto para escalar para um menor número de recursos, garantindo uma economia significativa. Para isso, é importante identificar gargalos nos sistemas e o porquê eles ocorrem, para definir uma boa estratégia de escalabilidade.

6.4 TOLERÂNCIA A FALHAS

Segundo Fowler (2017), um microsserviço projetado levando em consideração a tolerância falhas é aquele que é capaz de se manter disponível tanto com falhas causadas por problemas internos ou externos. Entende-se como uma falha interna, por exemplo, a implantação de um código que não passou em todos os devidos testes na *pipeline*, causando problemas no ambiente de produção, e uma falha externa, por exemplo, pode ser a indisponibilidade em algum *datacenter* que é responsável por sustentar um ou mais microsserviços, problemas com operadoras de rede, entre outros problemas externos.

Para Fowler (2017), para construir um microsserviço de fato tolerante a falhas, o primeiro passo é identificar todos os pontos importantes e potenciais cenários de falhas ou catástrofes. Após a identificação desses cenários, é dado início a etapa de construir uma estratégia de desenvolvimento de um ou mais microsserviços que sejam tolerantes aos problemas identificados. Com isso, é possível melhorar a resposta a incidentes, caso algum ocorra mesmo os riscos sendo mapeados.

Conforme Fowler (2017), alguns exemplos que podem ser utilizados para garantir a tolerância a falhas seriam a adoção de testes de resiliência e de carga dos microsserviços, além dos testes de lógica que devem ser aplicados nos códigos, isso diminuiria a taxa de problemas internos. Um exemplo para se utilizar em problemas externos seria implantar uma cópia de cada microsserviço em *datacenters* em regiões diferentes, para que, caso um dos *datacenters* apresente problema, uma cópia do microsserviço esteja disponível para execução em ambiente de produção em outro *datacenter*.

6.5 DESEMPENHO

Conforme Fowler (2017), um bom desempenho de um microsserviço pode ser definido pela quantidade de requisições e solicitados que ele trata de forma correta em um determinado período. Um microsserviço de alto desempenho trata estas solicitados em alta velocidade, de forma adequada e com uso eficiente de recursos. O quesito de desempenho está intimidade ligado ao requisito de escalabilidade, visto que, se um microsserviço possui uma boa estratégia de escalabilidade definida conforme a necessidade de recursos ou outras necessidades previamente

apontadas, ele sempre operará com o máximo de desempenho para aquele momento.

Para Fowler (2017), fornecer recursos ilimitados (por exemplo, recursos computacionais) a um microsserviço, não será garantia completa de que ele irá operar com o máximo de desempenho, por isso, é necessário definir uma estratégia sólida de escalabilidade do microsserviço, bem como o a estrutura de código deve possuir qualidade e coesão. Levando esses fatores em consideração, é possível manter um microsserviço sempre operando em alto desempenho, e reduzir custos desnecessários com o conceito de escalabilidade.

6.6 MONITORAMENTO

Para Fowler (2017), o monitoramento adequado de um microsserviço é aquele que leva em consideração três componentes: o armazenamento de registros de logs para consultas posteriores, gráficos com *dashboards* de controle de informações e um sistema de alertas bem definido para gerar alertas importantes sobre as principais métricas.

Conforme Fowler (2017), as informações de log são aquelas que mostram a quantidade de erros que ocorreram em um determinado período em um microsserviço. Com estas informações de log, é possível identificar rapidamente problemas nos microsserviços e mitigá-los de forma prática.

De acordo com Fowler (2017), quanto as métricas, todas as principais, por exemplo, como consumo de hardware, tráfego de rede, número de requisições por período deve ser monitoradas e exibidas em um *dashboard* de fácil acesso e visualização. *Dashboards* bem construídos ajudam os responsáveis a monitorar a saúde do microsserviço em tempo real, e, a identificar problemas rapidamente, tornando toda a operação dos microsserviços mais dinâmica e prática.

Ainda em Fowler (2017), seguindo de um bom sistema de monitoramento com armazenamentos de informações de log importantes e *dashboards* monitorando as principais métricas de um microsserviço, o último requisito seria a capacidade desses *dashboards* enviarem alertas quando algum microsserviço estiver com a saúde comprometida, alguma métrica esteja em desacordo ou também, problemas de indisponibilidade. Para as métricas, é necessário configurar limiares do tipo

normal, elevado e crítico, tornando melhor a interpretação das métricas presentes no *dashboard.*

6.7 DOCUMENTAÇÃO

Conforme Fowler (2017), a documentação correta de um microsserviço torna sua interpretação mais clara, e, a torna entendível para algum desenvolvedor que não participou de seu desenvolvimento diretamente. Além disso, documentar é uma boa prática, tornando a resolução de dúvidas sobre o funcionamento de um determinado microsserviço mais fácil. Nestas documentações, também podem explícitas as relações de vários microsserviços entre si e como elas funcionam, a arquitetura geral de comunicação dos microsserviços e regulamentação de recursos deles.

7 CONCLUSÃO

Após apresentar todos os conceitos e informações acerca do tema do trabalho, é possível concluir que, a adoção de uma arquitetura de microsserviços será melhor do que a abordagem de uma arquitetura centralizada, visto que, numa arquitetura de microsserviços, os sistemas possuirão uma maior disponibilidade e escalabilidade, pois não estão alocados todos na mesma arquitetura centralizada ou monolítica. Somando a disponibilidade e a escalabilidade maior utilizando uma arquitetura de microsserviços, a confiabilidade nos sistemas em produção será maior, trazendo menos preocupações quanto a infraestrutura para as organizações e empresas.

A abordagem de uma arquitetura centralizada ou monolítica não é de toda problemática, mas quando colocada frente à abordagem de microsserviços, a abordagem monolítica deixa a impressão de defasada e obsoleta, pois esse tipo de arquitetura é mais resistente para conceitos de uma arquitetura distribuída, como a alta disponibilidade e escalabilidade.

REFERÊNCIAS

CHEN, H. M.; *et al*. Y. **A Deployment Management of High-Availability Microservices for Edge Computing.** 2020 International Symposium on Computer, Consumer and Control (IS3C), 2020. Disponível em: <https://ieeexplore.ieee.org/document/9394019>. Acesso em: 26 nov. 2021.

FOWLER, S. J. **Microsserviços Prontos para Produção.** 1. ed. São Paulo, 2017.

HASSAN, S.; ALI, N.; BAHSOON, R. **Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity.** 2017 IEEE International Conference on Software Architecture (ICSA), 2017. Disponível em: <https://ieeexplore.ieee.org/document/7930193>. Acesso em: 10 dez. 2021.

MAYER, B.; WEINREICH, R. **A Dashboard for Microservice Monitoring and Management**. 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 2017. Disponível em: <https://ieeexplore.ieee.org/document/7958458>. Acesso em: 26 nov. 2021.

SÁ, M. P. de. **Emergent Microservices in Emergent Ecosystems.** 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), 2020. Disponível em <https://ieeexplore.ieee.org/document/9302824>. Acesso em: 16 dez. 2021.