DAQHUB.APP – PROPOSTA DE ARQUITETURA E APLICAÇÃO PARA UM REPOSITÓRIO CENTRAL DE RESULTADOS DE ENSAIO PARA LABORATÓRIOS E LINHAS DE PRODUÇÃO

Johnathan Radünz Nunes

SUMÁRIO EXECUTIVO

O sistema proposto – chamado de *daqhub* – tem como objetivo ser um produto de software que centraliza os resultados de diversos e diferentes sistemas que são desenvolvidos pela DAQSYS, bem como prover um acesso rápido aos parâmetros de produto e de ensaios de qualquer lugar, sem a necessidade de acesso direto ao computador que está efetivamente sendo usado na aquisição de dados e ensaio dos produtos.

É uma suíte completa que permite o acompanhamento dos ensaios, sejam estes em laboratórios (como por ex., desenvolvimento de produtos) ou em linha de produção, permitindo uma maior rastreabilidade dos produtos produzidos, uma melhor visão sobre os testes realizados e dados adquiridos e um melhor entendimento de várias métricas de qualidade e performance relevantes. Além disso, é uma ferramenta de integração que permite que os dados sejam expostos de forma mais efetiva e de forma a permitir a interoperabilidade com sistemas proprietários do cliente ou outros softwares de terceiros, com geração de relatórios e até controle remoto das estações quando isso for desejável, tudo de um lugar centralizado e com um melhor gerenciamento e maior segurança.

A arquitetura do *daqhub* é pensada de forma que seja possível escolher entre a opção gerenciada e hospedada na nuvem ou sua versão *on-premise* que permite que o cliente hospede a sua própria instância localmente.

PROBLEMATIZAÇÃO

Atualmente, todos os sistemas desenvolvidos e entregues pela empresa são totalmente autocontidos e operam de forma isolada, até mesmo os sistemas que têm uma 'base comum' ou ainda os que possuem várias réplicas de um mesmo *software*.

Toda a configuração do sistema, alterações de parâmetros de produto e de ensaio, visualização de resultados dos ensaios e *logs* de erros só podem ser acessados diretamente pelo *software* desenvolvido especialmente para aquela aplicação – ou, nos casos em que são arquivos legíveis pelo ser humano, qualquer interação requer que o computador seja acessado fisicamente na maioria das vezes.

A proposta que surge com a apresentação do *daqhub* é que a única responsabilidade do sistema de teste seja de realmente executar as ações pertinentes à rotina de ensaio do produto. Todas as preocupações relativas aos dados passam a ser centralizados numa única plataforma, abrangendo desde o controle de usuários até a auditoria das alterações que foram realizadas. Esta abordagem começa a fazer sentido quando vários sistemas ou diversas 'cópias' de um sistema são implementados num mesmo cliente, já que permite a agregação e convergência de todos estes sistemas para um único local em que os usuários podem ter acesso a todos os seus dados e permite uma verificação geral da planta como um todo.

Historicamente, o setor de manufatura é bastante resistente para a adoção de tecnologias de nuvem e até mesmo para o uso de qualquer tipo de serviços externos à sua rede corporativa. Por esse motivo, soluções para este setor costumam seguir o mesmo padrão de manter todo o ecossistema necessário contido na mesma máquina, apenas com comunicação local e usando o mínimo possível de recursos da própria organização. Todos os resultados são sempre salvos no disco da máquina – no máximo, uma pasta compartilhada quando é realmente imprescindível. A máquina em que o ensaio é executado sempre é bastante restrita ou sequer está conectada na rede corporativa. Eventualmente, recuperar um resultado de ensaio pode ser um trabalho a parte. Não é necessário dizer também que, eventualmente, dados são perdidos junto com discos rígidos.

JUSTIFICATIVA E OBJETIVO GERAL

Vê-se um movimento – embora ainda bastante tímido, principalmente nas empresas brasileiras – de maior adoção de soluções externas e até uso de provedores de nuvem pelas fábricas de forma geral. Nesse contexto e na expectativa de auxiliar na transformação digital destes parceiros, o sistema proposto é uma peça fundamental para que os dados coletados estejam disponíveis, facilmente acessíveis e que sejam realmente utilizados para trazer mais inteligência para o negócio.

De forma geral, o *daqhub* pode ser visto como um meio de prover toda a infraestrutura necessária, de forma padronizada e com uma 'única fonte da verdade', para todos os softwares de teste e de aquisição de dados que estão inseridos no cliente. Apesar da proposta ser bastante ambiciosa, como um *MVP* (Mínimo Produto Viável) ela é bastante factível levando em consideração os softwares que já foram desenvolvidos pela DAQSYS e o conhecimento adquirido pela implementação de sistemas parecidos – embora mais simples – entregues recentemente pela empresa.

Além de incluir interfaces para os softwares proprietários, prevê-se integração com vários softwares já bastante adotados pelo chã de fábrica.

Uma das principais particularidades para uma proposta como a apresentada – e de certa forma uma das mais ignoradas – é a necessidade de levar em conta o ambiente atual e as necessidades do negócio, de forma realista e enxuta. Por isso, esse projeto de intervenção está focado em resolver um problema real com base no cenário atual disponível. Isso significa levar em conta a equipe responsável pelos sistemas subjacentes (sistemas de teste já existentes), a equipe responsável pela nova plataforma (uma única pessoa) e a pessoa responsável por toda a infraestrutura (também uma única pessoa).

Além disso, é importante levar em conta a maturidade do projeto: Para a validação do conceito e inserção do produto no mercado, o quanto mais simples o sistema for, melhor para a empresa. Entretanto, alguma simplicidade eventualmente precisa ser sacrificada para que uma maior flexibilidade no longo prazo seja alcançada. Os resultados apresentados nesse documento são o que se acredita ser um balanço ideal de comprometimentos para que o sistema seja entregue na sua melhor forma possível para o estágio atual, mas ainda assim, pensando numa rápida (e saudável) evolução.

Num mundo em que os dados tomaram a dianteira e em que são considerados fundamentais e de extrema importância para as organizações, permitir que eles sejam realmente explorados traz muita relevância para a intervenção proposta. Manter os dados em pequenos silos vai na contramão do que a realidade atual exige e, portanto, o *daqhub* tem um campo abrangente, e de certa forma bastante inexplorado, para conquistar.

Têm-se como objetivo geral para este projeto a proposta da arquitetura inicial do sistema, levando em conta as capacidades e demandas atuais, bem como uma prospecção de crescimento que faça sentido para a empresa que irá fornecer esse produto. A arquitetura proposta deve também ser desenhada de forma que permita uma evolução do sistema tanto no quesito de suas capacidades quanto funcionalidades.

Também é previsto, nos anexos deste artigo, a apresentação de algumas sugestões para que a evolução da arquitetura e dos sistemas seja feita da forma mais organizada e estruturada possível.

Espera-se que, após a apresentação deste estudo sobre a arquitetura proposta, o sistema seja implementado em um estágio bastante primitivo, apenas para a validação do conceito e verificar sua aceitação, mas já direcionado em todos os aspectos do ciclo de vida do produto. Todos os pontos relevantes aos quesitos de segurança, observabilidade e agilidade que serão conceituados, entretanto, são considerados requisitos mínimos para o lançamento da primeira versão do *daqhub*.

FUNDAMENTAÇÃO TEÓRICA

Espera-se que mudanças constantes ocorram. Uma visão arquitetural pré planejada pode não funcionar com o dinamismo atual do desenvolvimento de software. Por isso, é extremamente importante que as arquiteturas sejam pensadas já com a ciência que o panorama técnico vai mudar [1]. Até que a arquitetura seja desenhada, implementada, atualizada e que mudanças inevitáveis tenham sucesso, um arquiteto não pode julgar a viabilidade ao longo prazo da sua proposta. Arquiteturas são hipóteses e necessitam serem provadas por sua implementação e pela sua medição [1,2].

Uma boa arquitetura não apenas atende às necessidades dos usuários, desenvolvedores e donos em um ponto no tempo, mas também atendem ao longo do tempo. Uma boa arquitetura é o entendimento que jornada é mais importante que o destino e é um processo constante, não um artefato congelado [2].

Por meio da utilização de *Fitness Functions* integradas ao ciclo de desenvolvimento e de implementação, é possível que arquitetos e desenvolvedores tenham uma alta confiança de que evoluções no sistema e na arquitetura não violarão as diretrizes do projeto. Essas funções guiam o processo de decisão, permitindo que mudanças no negócio ou nas tecnologias evoluam a arquitetura. Um ponto crucial no planejamento da arquitetura é a definição das dimensões que são importantes para o projeto: escalabilidade, performance, segurança etc. [1]. Um exemplo de *fitness function* é o *Chaos Monkey* - da *Netflix* – que termina instâncias de serviços em produção para garantir que os engenheiros implementaram a resiliência corretamente [3,4].

Sistemas passam muito mais tempo de sua vida em operação do que em desenvolvimento. A fase em que o time menos conhece sobre as necessidades estruturais do software costuma ser também, ironicamente, a qual as decisões mais impactantes costumam ser tomadas. [1,4]. Uma boa arquitetura deve permitir que decisões não críticas sejam postergadas: Banco de dados, *frameworks*, ferramentas, servidores... [2]

O sucesso de uma arquitetura é alcançado com a cultura de *DevOps*. Automatizar procedimentos é o fator mais importante para trabalhar de forma consistente e confiável. Toda e qualquer atividade recorrente deve ser automatizada: Geração de código, compilação de documentação, testes de software, provisionamento de recursos, etc. [1,5]

Com foco em entrega mais rápida e melhorias incrementais, é possível colocar o software em produção e aprender como ele irá se comportar frente aos estímulos do mundo real. Coisas ruins irão acontecer – espera-se que aconteçam. No mundo real o programa não terá recursos infinitos. Por meio de técnicas de mitigação (*Circuit Breakers*, *Timeouts* e *Bulkheads*) é possível conter uma falha (é possível adaptar e recuperar ou falhar de forma controlada), impedindo que todo o sistema seja impactado. [4,5]

DevSecOps é uma nova abordagem que traz a segurança pra dentro de todo o ciclo de vida de desenvolvimento. A segurança torna-se uma responsabilidade compartilhada e vários *checks* de segurança podem ser adicionados à esteira de entrega contínua [7,8,9].

Além disso, muitas empresas não percebem que podem usar o ambiente de produção para testes exploratórios. No lugar de colher requisitos, é possível usar o método científico por meio do desenvolvimento baseado em hipóteses [1]. Com a ajuda de *DevOps* e uma arquitetura evolutiva, não só novas funções podem ser colocadas em produção rapidamente como também serviços inteiros podem ser substituídos para uma parcela dos usuários, por meio de técnicas de *release* como *Canary Release* [1,7,8].

O monitoramento do sistema é um dos pilares da metodologia de *DevOps*. Apesar de vários dos componentes desta categoria serem opcionais, cabe ao arquiteto julgar o custo-benefício de cada ferramenta. Coleta de logs e monitoramento ajudam com análises de *postmorten*, recuperação de incidentes e descoberta de defeitos, por exemplo [4,6].

O time de desenvolvedores precisa de monitoramento para poder otimizar o projeto. O time de operações precisa do monitoramento para entender as condições atuais, tendências históricas ou fazer projeções [1,4]. Depurar um sistema transparente é muito mais fácil e, portanto, ele evolui mais rapidamente. Não só isso, mas as mudanças técnicas e arquiteturais dependem totalmente dos dados coletados da infraestrutura atual [4].

Por fim, do ponto de vista evolutivo, não é rigorosamente necessária uma arquitetura do tipo micro serviços. Um monolito modular pode apresentar a mesma habilidade de aceitar mudanças incrementais. Quando um sistema é bem feito, ele requer apenas uma fração de recursos humanos para criar e manter: Alterações são rápidas e simples; Defeitos ocorrem com menos frequência; Funcionalidade e flexibilidade são maximizados [1,2]. Padrões que fazem sentido para grandes organizações normalmente são o exato oposto do que funcionam para pequenos sistemas. Nesse sentido, monolitos são extremamente mais fáceis e ágeis de serem mantidos por pequenos times [10].

PERCURSO METODOLÓGICO DA INTERVENÇÃO

A análise dos requisitos para este projeto resultou no levantamento das seguintes características arquiteturais: Simplicidade, Extensibilidade, Manutenibilidade, Agilidade, Testabilidade, Disponibilidade, Segurança e Resiliência.

Uma das primeiras decisões que precisam ser tomadas é entre criar o sistema na forma de um monolito ou sua separação em micro serviços. Por ora, não é necessário pensar em uma arquitetura complexa e distribuída globalmente. É melhor falhar rápido.

Tendo isso em mente e com base na estrutura organizacional que será dedicada ao projeto, prefere-se a adoção do "monolito primeiro". Alguns serviços específicos serão usados como aplicações satélites (como por exemplo a autenticação).

A aplicação principal deve ser construída de forma monolítica em um *framework* ágil e produtivo e que seja de conhecimento da equipe. Por este motivo, optou-se pelo uso do *Ruby on Rails*. Em sua versão mais recente, até mesmo o *frontend* da aplicação pode ser construído de forma bastante rápida. A arquitetura do software deve prever uma boa isolação entre os módulos para que seja possível, no futuro, a transformação desses em novos serviços se necessário.

A arquitetura ideal e que permite o melhor balanço entre as características desejadas é a Arquitetura Baseada em Serviços, representada na Figura 1.

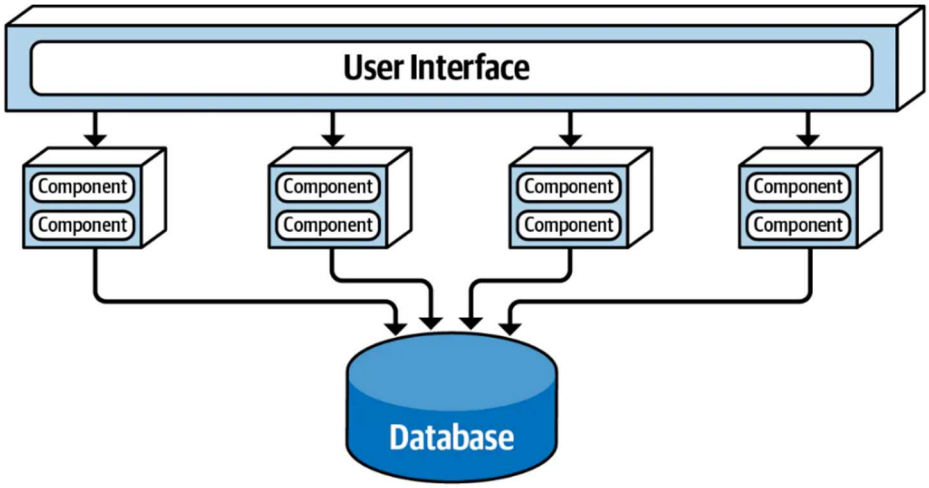


Figura 1- Arquitetura Baseada em Serviços. [11]

É importante destacar que o *Ruby on Rails (RoR)* segue a o padrão arquitetural de camadas e isso traz o primeiro comprometimento relevante para a escolha de padrão de arquitetura *vs. framework*. Com o uso de algumas bibliotecas, é possível deixar o *RoR*

mais amigável ao desenvolvimento orientado ao domínio e permite que os módulos da aplicação sejam segregados de forma a respeitar a arquitetura baseada em serviços.

Para uma primeira iteração, a maturidade do ecossistema *Rails* vai permitir um desenvolvimento muito mais rápido de um protótipo para que a ideia seja validada. Uma sugestão de evolução é a mudança para o *framework Hanami*. Também desenvolvido com a linguagem *Ruby*, o *Hanami* é um projeto que é baseado em dois princípios: Arquitetura Limpa e Monolito Primeiro, se adequando muito melhor à necessidade do *daqhub*. Outra opção é a extração dos domínios que serão criados em *rails engines* (módulos separados).

A qualidade é indiscutível e essencial para o sucesso do projeto e a segurança digital é uma exigência de todos os sistemas que rodam atualmente na *internet*. Para o setor manufatureiro, essa é uma das maiores preocupações e deve-se levar a segurança como um aspecto crítico desde o começo. Para isso, serão adotadas várias boas práticas de *DevOps* e *DevSecOps* discutidas durante as aulas. Também serão usados testes automatizados desde o primeiro dia. Além dos benefícios de qualidade e segurança, a esteira de implementação também permitirá garantir mais agilidade para o lançamento de novas funcionalidades. O *RoR* continua sendo uma ótima escolha, já que é seguro por padrão e sua comunidade segue uma filosofia de alta qualidade e de testes constantes.

Testes unitários, de sistema e de integração serão usados para garantir a qualidade do código, juntamente com ferramentas para verificar a cobertura dos testes de forma automatizada. Do ponto de vista de segurança, bibliotecas de análise estática e ferramentas de análise de vulnerabilidade também serão incluídas na esteira de implementação para verificar não apenas o código, mas também realizar testes dinâmicos na aplicação e na infraestrutura implementada. Os fluxos de implementação ainda devem garantir a execução das *fitness functions* que ajudarão a guiar a evolução arquitetural.

A arquitetura precisa ser desenhada para uma alta disponibilidade e para que seja resiliente. Como uma plataforma pensada para o setor fabril, deve-se garantir que o sistema funcionará sempre que algum cliente estiver produzindo – o que pode significar 24h/dia. Para isso, propõe-se a utilização de uma observabilidade agressiva sobre todos os sistemas. Optou-se pelo uso do *Elastic Stack* para isso, pois além de ser uma das principais escolhas quando se fala em monitoramento, também é uma ferramenta que faz todo o sentido para o projeto proposto pois ela permite amarrar as aplicações e a infraestrutura de ponta-a-ponta, garantindo uma visão centralizada para os times de

desenvolvimento, de operação e de segurança da informação. É possível monitorar as cadeias de implementação contínua diretamente pelo *Kibana*, bem como visualizar os *logs* e métricas das aplicações pertencentes ao *daqhub* e as aplicações de ensaio que ficam distribuídas pelos clientes, monitorar toda a infraestrutura alocada, além de permitir investigações e *insights* relativos à segurança.

Empresas do setor manufatureiro – ao menos para o chão de fábrica – tem bastante resistência com sistemas externos. Por isso, o *daqhub* é pensado para que seja *cloud-native* mas precisa também prever o atendimento aos clientes que permitem apenas o uso de alternativas *on-premise*.

Essa restrição leva a algumas decisões que não ficam só entre desenvolver ou comprar, mas também entre usar alguma solução gerenciada ou não. Como exemplo, pode-se citar a autenticação/autorização e o balanceamento de carga dos servidores.

Inicialmente, é proposto a utilização do *Keycloak* como solução para a autenticação e autorização dos usuários, de forma a limitar acesso aos recursos e garantir a auditoria. A implementação deve prever uma abstração para que seja possível usar um serviço gerenciado em nuvem – como o *Cognito* da *AWS* – ou até mesmo um que seja gerenciado pelo próprio cliente internamente – como o *Active Directory*, da *Microsoft*.

Já para o balanceamento de carga, optou-se pelo uso do *NGINX* ou *NGINX+* em detrimento dos serviços já gerenciados. Essa estratégia deve atender bem os requisitos do começo do projeto, mas deve-se ter a noção que pode ser necessário, ao menos, reavaliar essa decisão antes do *release* para o mercado. Incluindo módulos no *NGINX*, é possível adicionar várias funções de *API Gateway* (autenticação, testes A/B e limite de chamada).

Deve-se prever também a automação da infraestrutura para garantir que não seja necessário a configuração de novos servidores e aplicações. Ferramentas como *Terraform* e *Ansible* serão utilizadas para isso.

A necessidade de garantir que o sistema possa operar *on-premise* e de forma totalmente autônoma, sem necessidade de conexão com qualquer serviço em nuvem e a possibilidade de fazer a implementação de todo o sistema em vários provedores diferentes permite não só uma maior resiliência por meio da estratégia *multicloud,* mas também garante que o projeto não fique preso a nenhum provedor específico e se torna uma vantagem estratégica para clientes que pensam em fazer a migração para a nuvem: Permite que eles hospedem o sistema inicialmente em sua estrutura e migrem

posteriormente para o provedor que for de seu interesse, ao qual o seu time de TI já esteja acostumado ou para o provedor com o qual ele já tem relação comercial.

Todas as ferramentas consideradas, inclusive todos os softwares pertencentes à *Elastic Stack,* foram selecionados especificamente para permitirem essa estratégia de manter a implementação totalmente agnóstica ao provedor que será utilizado, mesmo que isso signifique rodar no *data-center* proprietário do cliente.

Futuramente, será necessário pensar em uma estratégia *multicloud* para garantir o funcionamento mesmo que todo um provedor caia. Nesses casos, não é necessário que todos os sistemas estejam funcionando, mas algumas *APIs* (parâmetros e resultados, por exemplo) precisam estar disponíveis a todo momento pois são essenciais para a continuidade das linhas de produção dependentes do *daqhub*. Nas fases iniciais desta intervenção, a sugestão é que seja mantido uma réplica dos parâmetros localmente na infraestrutura do cliente e os *softwares* devem manter um *buffer* para que os resultados não sejam perdidos caso o *daqhub* esteja temporariamente indisponível. Essa estratégia demanda um pouco mais de energia inicialmente no desenvolvimento das aplicações rodando na borda, mas garante o funcionamento mesmo que todos os *links* de conectividade do cliente ou provedor estejam fora de operação. Para isso, bancos *NoSQL* distribuídos podem ser usados. Essa resiliência local pode ser parte integral da arquitetura.

Para a prova de conceito, será utilizado o provedor *AWS*. Duas zonas de disponibilidade serão configuradas para garantir uma alta disponibilidade dos sistemas. É importante que se note que as decisões por não usar alguns serviços gerenciados gera um custo extra se comparado com as ferramentas fornecidas pelo próprio provedor. Por isso, um estudo dos custos para desenvolvimento e produção em carga mínima foi feito, de forma a auxiliar o direcionamento do projeto e garantir que não existirão surpresas no uso da nuvem. O levantamento de custos pode ser visto no Anexo C deste documento. Os principais (e mais relevantes) custos foram levados em consideração, mas alguns valores foram ignorados. Para uma definição do custo, será necessário antes uma avaliação da quantidade de dados gerados e transferidos para dentro e para fora do provedor. Os custos apresentados pela *AWS* demonstraram ser muito mais competitivos que os concorrentes.

A escolha da *AWS* como provedor também é justificada pelo seu ecossistema, pois já inclui diversos serviços para análise e inteligência de dados, serviços de *IoT* e conexão com equipamentos industriais, entre outros. Do ponto de vista de inteligência de dados,

as ferramentas da *AWS* podem não ser a melhor escolha e será necessário que um estudo mais minucioso seja feito nesta frente, sugerido como uma terceira etapa, após a validação inicial da ideia. Conforme visto nas aulas, boas opções são o *DataBricks* e o *Snowflake*. É por esse motivo que a evolução arquitetural tem um papel bastante importante: Algumas variáveis ainda precisam ser mais bem detalhadas. Essa intervenção está dividida em três etapas, apresentadas na Tabela 1:

ETAPA 1	Análise dos requisitos e definições iniciais.
	Testes exploratórios, descoberta e escolha de ferramentas, *frameworks*, etc.
	Modelagem dos principais fluxos: Desenvolvimento e implementação.
	Levantamento de métricas importantes + *fitness functions (FF).*
ETAPA 2	Modelagem de diagramas por componentes/sistemas e modelagem de ameaças.
	Desenvolvimento: *FF*; *IaC*; métricas importantes para o negócio; aplicação.
	Período de prova de conceito – internamente / DAQSYS.
	Apresentação dos resultados. Avaliação de viabilidade / continuidade.
ETAPA 3	Testes exploratórios, descoberta e escolha de ferramentas e *frameworks* (dados).
	Modelagem dos fluxos de dados.
	Modelagem de diagramas, análises de governança e segurança dos dados.
	Apresentação de perspectivas e possibilidades. Avaliação de continuidade.

Tabela 1 - Etapas do Projeto de Intervenção.

Os elementos e componentes considerados para essa primeira etapa do projeto podem ser vistos pelo diagrama da Figura 2 e estão melhor detalhados no Anexo A, enquanto uma sugestão de evolução é vista no Anexo B.

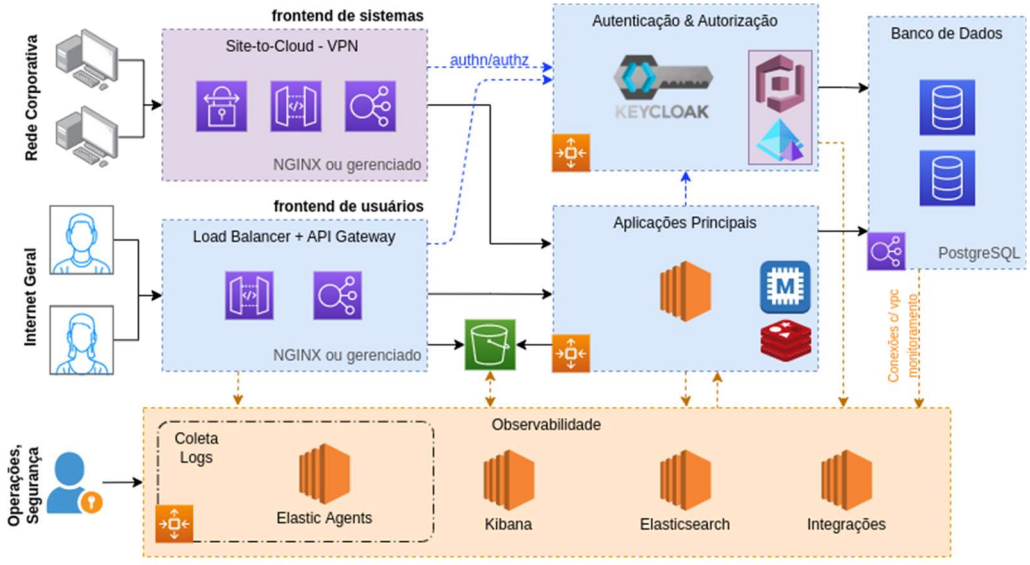


Figura 2- Componentes principais - daqhub etapa 1 e 2

Para um melhor entendimento e uma melhor apresentação, a esteira de implementação será apresentada de dois pontos de vista distintos.

A esteira de implementação de desenvolvimento, apresentada na Figura 3, será responsável por toda a verificação do código (qualidade, estilo, segurança), construção e verificação das imagens de contêiner, e disponibilização de novas versões (sempre em *releases* do tipo canário).

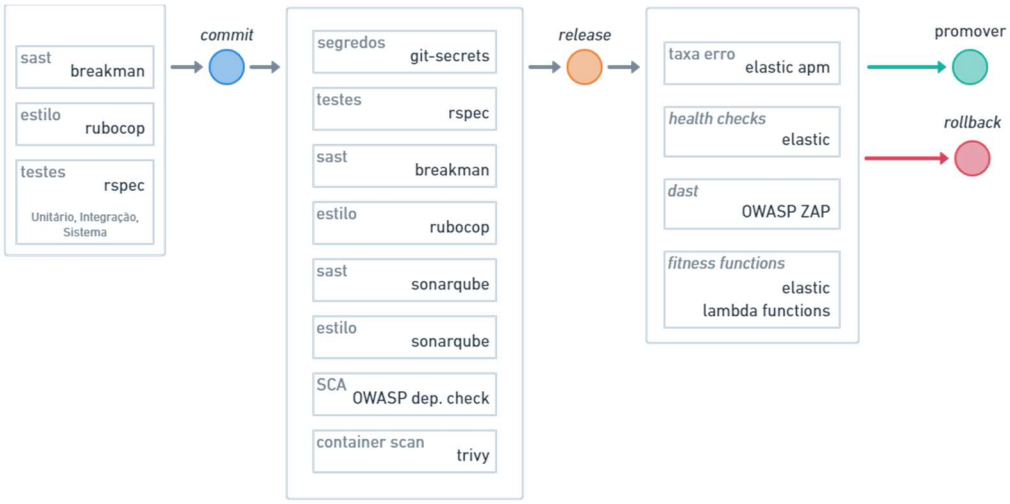


Figura 3- Esteira de implementação - Desenvolvimento

Já a esteira de implementação de infraestrutura, como é vista na Figura 4, será responsável pelas verificações relativas à evolução da arquitetura e da segurança da infraestrutura, além de garantir que exista todo um histórico de quanto essas alterações impactam financeiramente o projeto.

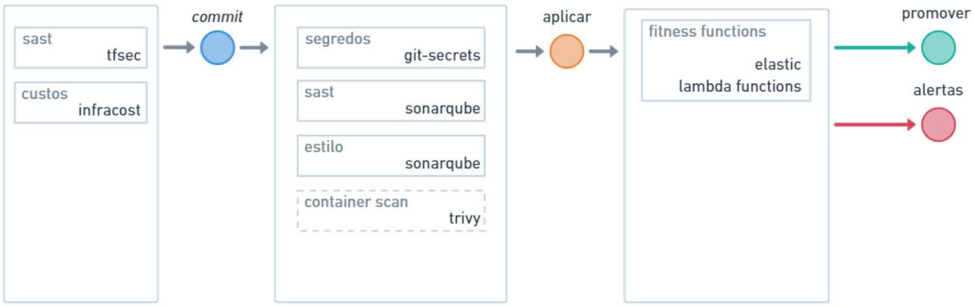


Figura 4- Esteira de implementação - Infraestrutura

A ferramenta escolhida para orquestrar toda a cadeia de implementação é o *GitLab*, não apenas por já ser de uso da empresa, mas por ser a ferramenta mais completa que permite cobrir todo o ciclo de *DevOps*. Todo o ambiente de *CI/CD* deve ser monitorado pela pilha de observabilidade.

Lambda Functions, embora não apareçam nos desenhos arquiteturais, serão bastante utilizadas como apoio às metodologias de *DevSecOps* e da Arquitetura Evolutiva. Por exemplo, a técnica de r*elease* canário pode ser automatizada com essas funções sem servidor, bem como permite a execução das *fitness functions.*

Finalmente, a definição de *fitness functions* só é possível pelos passos já elencados na esteira de implementação. Foram definidas várias métricas de verificação abrangendo desde os conceitos mais básicos e difundidos (como por exemplo, cobertura de testes acima de 90%) até os mais complexos de se identificar. Para garantir a manutenibilidade e extensibilidade, por exemplo, métricas que analisam coesão, acoplamento e modularidade do código podem ser desenvolvidas. O Anexo D relaciona todas as *fitness functions* propostas e a respectiva característica arquitetural que ela monitora.

RESULTADOS ESPERADOS

Muito trabalho ainda precisa ser feito. Os resultados apresentados nesse documento correspondem à uma primeira etapa das definições arquiteturais levando em conta os requisitos e motivadores para um sistema agregador de informações como proposto. A arquitetura apresentada é vista como o mínimo necessário para garantir que os aspectos de agilidade, segurança e operabilidade estejam presentes desde o início do ciclo de vida do produto. A entrega deste trabalho marca o fim da primeira de três etapas.

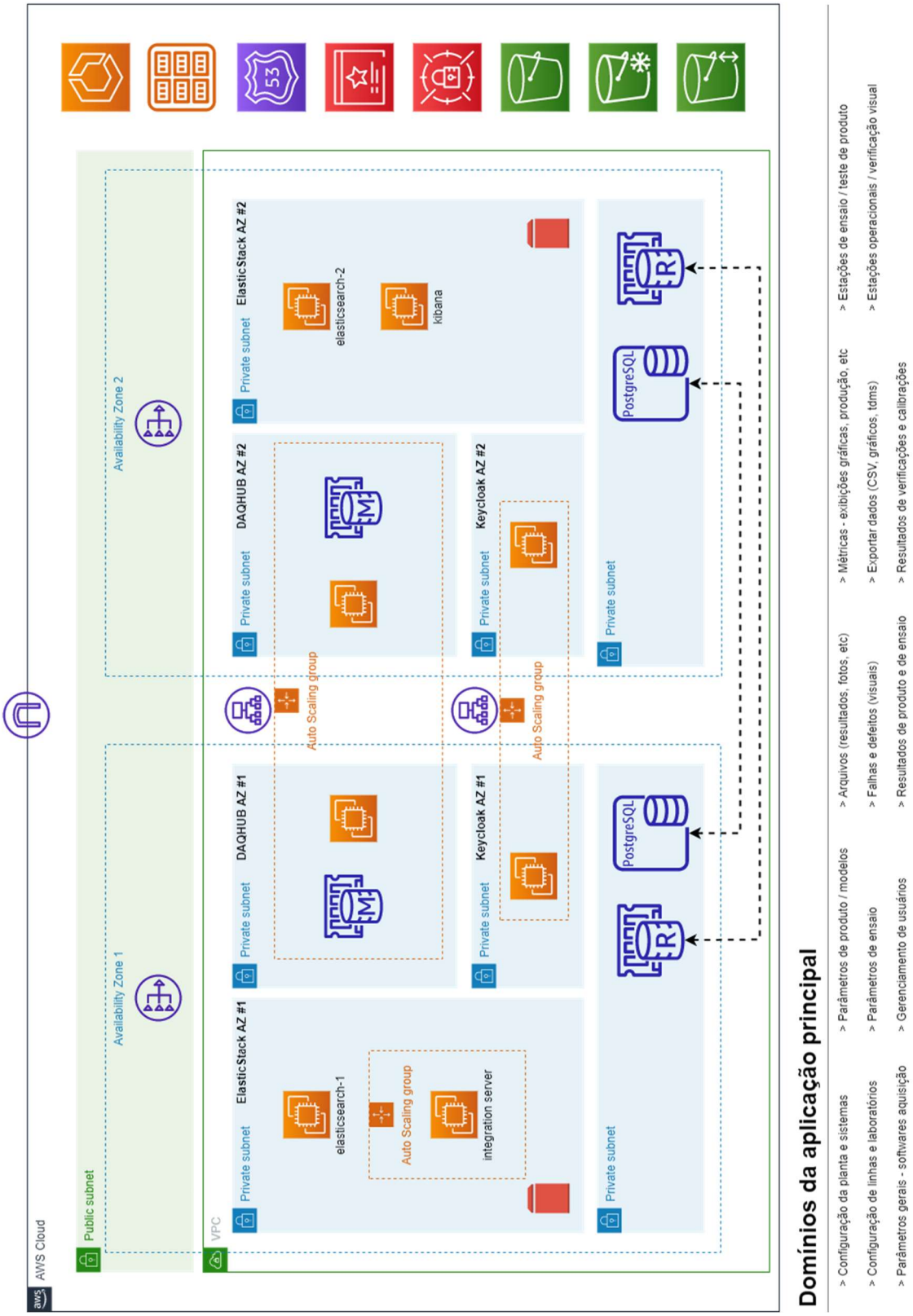
O conjunto como um tudo permitirá que as melhores práticas possíveis sejam empregadas e viabilizará uma análise preliminar da viabilidade do sistema. Há muito campo para otimização, principalmente do ponto de vista de custos. É possível até iniciar a validação da ideia coletando dados dos laboratórios, que não requer alta disponibilidade.

Foram especificadas *fitness functions* para verificação de resiliência, observabilidade, segurança, disponibilidade, desempenho e agilidade (manutenibilidade e extensibilidade). Com isso, todas as características elencadas no início do capítulo serão verificadas de forma a guiar a evolução arquitetural e garantirão que os requisitos inicialmente definidos serão respeitados. Espera-se que essas características sejam constantemente verificadas, principalmente para garantir que elas ainda façam sentido no decorrer da evolução do projeto. Como visto, a arquitetura deverá ser provada pelo seu uso, sua evolução e sua medição. Esse é só o início do trabalho de arquitetura do sistema.

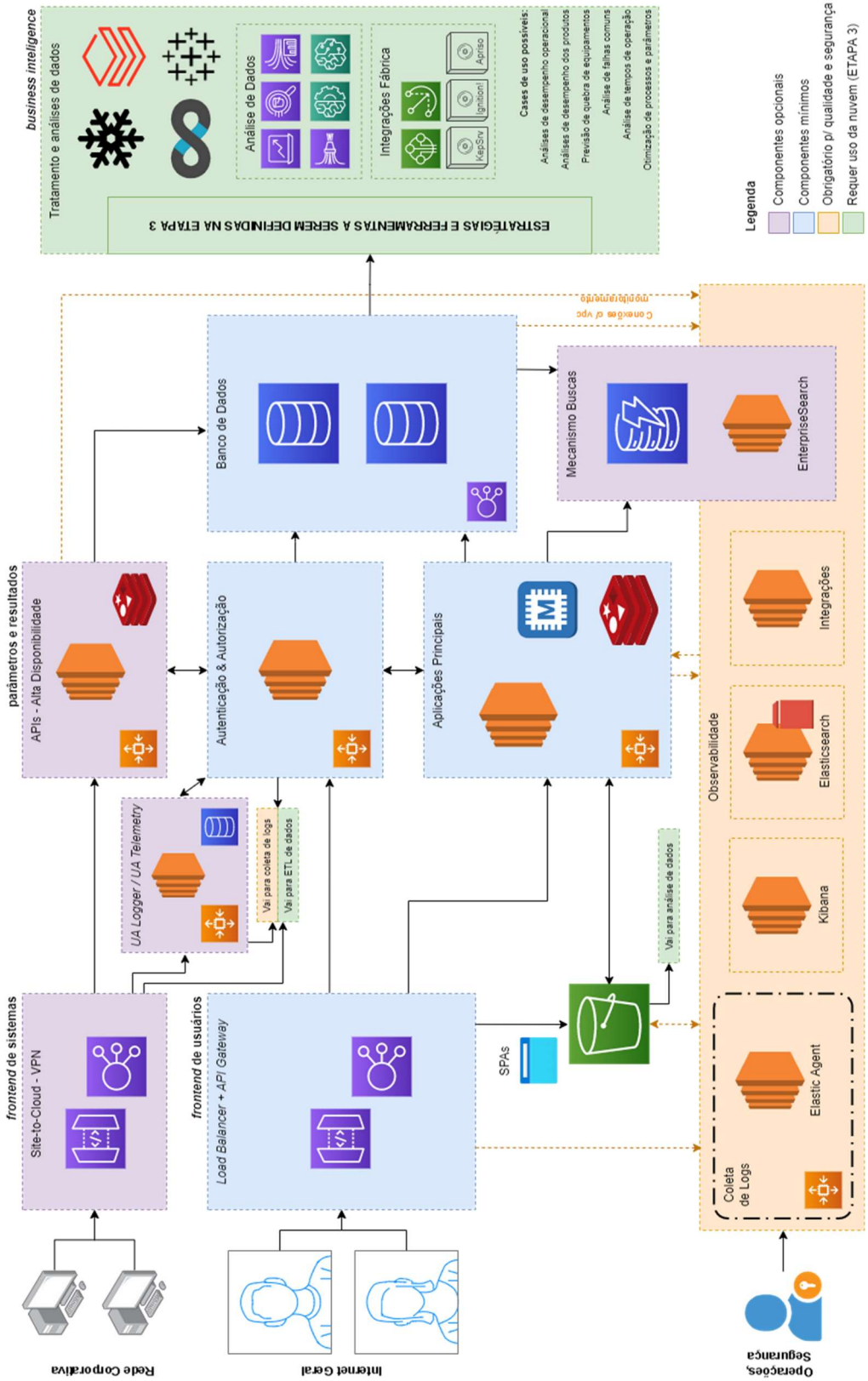
REFERENCIAL TEÓRICO

1. N. Ford, R. Parsons, e P. Kua, **Building evolutionary architectures:** Support constant change. Beijing: O'Reilly, 2017.
2. R. C. Martin, **Clean architecture:** A craftsman's guide to software structure and Design. Boston: Prentice Hall, 2018.
3. Netflix, **Chaos monkey,** [Online]. Disponível em: https://netflix.github.io/chaosmonkey/. Acessado em 10 abr. 2022.
4. M. T. Nygard, **Release it!:** *Design and deploy production-ready software*. Raleigh: The Pragmatic Bookshelf, 2018.
5. A. Hunt e D. Thomas, **The pragmatic programmer.** Massachusetts: Addison-Wesley, 2000.
6. F. R. Silva et al. **Cloud Computing**. Porto Alegre, 2020
7. E. Freeman, **DevOps Para Leigos.** Rio de Janeiro: Alta Books, 2021.
8. A. Muniz et al. **Jornada DevOps.** Rio de Janeiro: Brasport, 2020
9. RedHat, **O que é DevSecOps** [Online]. Disponível em: https://www.redhat.com/pt-br/topics/devops/what-is-devsecops/. Acessado em 07 abr. 2022.
10. Signal vs. Noise, **The majestic monolith,** [Online]. Disponível em: https://m.signalvnoise.com/the-majestic-monolith/. Acessado em 07 abr. 2022.
11. N. Ford e M. Richards, **Fundamentals of Software Architecture:** An Engineering Approach. Beijing: O'Reilly, 2020.

ANEXO A – ARQUITETURA INICIAL DO SISTEMA



ANEXO B – EVOLUÇÃO ARQUITETURAL PROPOSTA



ANEXO C – LEVANTAMENTO DE CUSTOS (EM DÓLARES)

SERVIÇOS / CAPACIDADES CONTRATADAS		SOB DEMANDA	CAPACIDADE RESERVADA		
			1 ANO	3 ANOS	
		MENSAL	MENSAL	MENSAL	ANTECIP.
AUTENTICAÇÃO	Balanceador de carga (authn/authz)	0,0025 por GB por hora			
	Keycloak #1 (t4g.micro)	6,25	3,87	2,63	0,00
	EBS HDD 8GB	0,24	0,12	0,12	0,00
	Keycloak #1 (t4g.micro)	6,25	3,87	2,63	0,00
	EBS HDD 8GB	0,24	0,12	0,12	0,00
APLICAÇÃO DAQHUB	Balanceador de carga (app principal)	0,0025 por GB por hora			
	Servidor Aplicação #1 (t4g.small)	12,26	7,67	5,33	0,00
	EBS SSD 8GB	0,64	0,64	0,64	0,00
	MemCached (ElasticCache) – cache*	23,36	16,06	12,41	0,00
	Redis (ElasticCache) - session/jobs	11,68	8,03	5,84	0,00
	Servidor Aplicação #2 (t4g.small)	12,26	7,67	5,33	0,00
	EBS SSD 8GB	0,64	0,64	0,64	0,00
BANCO DADOS	Amazon RDS PostgreSQL (t4g.small)	47,45	33,95	11,31	407,00
	Amazon RDS PostgreSQL (t4g.medium)	94,17	67,89	22,63	801,00
	Amazon RDS PostgreSQL (m6g.large)	232,14	148,70	47,60	1713,00
	Amazon RDS - Storage	0,23 por GB (por instância)			
	Amazon RDS - Backup	0,095 por GB (por instância)			
	Transferência de dados *	0,00 por GB / 0,02 por GB (outra região)			
ARMAZENAMENTO	S3 Inteligent Tier (resultados, log cliente)	0,023 por GB (FA) / 0,0125 por GB (IA)			
	S3 Glacier (logs daqhub, backups, etc)	0,004 a 0,0036 por GB			
	S3 Deep Archive (arquivamento hisórico)	0,00099 por GB			
	Recuperação de dados – Intel. Tier (IA)	0,01 por GB			
	Recuperação de dados – Glacier	0,03 por GB (instant) / 0,00 por GB (bulk)			
	Recuperação de dados – Deep Archive	0,02 por GB (std) / 0,0025 por GB (bulk)			
	Transferência de dados – externa	0,09 por GB			
	Transferência de dados – entre regiões	0,01 até 0,02 por GB			
MONITORAMENTO	Balanceador de carga (monitoramento)	0,0136986 por GB por hora			
	Elasticsearch #1 – dados (t4g.medium)	24,53	15,40	10,59	0,00
	Elasticsearch #2 – dados (t4g.medium)	24,53	15,40	10,59	0,00
	Elasticsearch #3 – tiebreaker (t4g.micro)	6,25	3,87	2,63	0,00
	Integration Server (t4g.micro)	6,25	3,87	2,63	0,00
	Kibana Server (t4g.micro)	6,25	3,87	2,63	0,00
	EBS SSD 180GB (por instancia de dados)	18,00 (0,10 por GB)			0,00
REDE	Internet Gateway	0,09 por GB – transferências externas			
	NAT Gateway	0,045 por GB + 0,045 por hora + transf.			
	AWS Secrets Manager	0,40 por segredo por mês			
	AWS Certificate Manager	0,75 por certificado por mês			

Tabela 2 - Levantamento de custos para primeira iteração - daqhub

ANEXO D – FITNESS FUNCTIONS

MÉTRICA A SER VERIFICADA (*FITNESS FUNCTION*)		FONTE DO DADO
AGILIDADE	Cobertura de testes acima 90 %	simplecov / deep-cover
	Índice de manutenibilidade acima de 0.5	calculado (rubocop)
	Baixa complexidade ciclomática (valores serão estatísticos)	rubocop
	Baixa complexidade, atendimento ao princípio SRP	rubocop
	Não existência de código duplicado	rubocop / sonarqube
	Modularidade, acoplamento e coesão	packwerk / rubocop
	Respeito dos módulos aos limites arquiteturais	packwerk / rubocop
	Nome de métodos/variáveis, atendimento às convenções	rubocop
RESILIÊNCIA	Taxa de erro menor que 1% para novas implementações	elastic apm
	Taxa de erro menor que 5% na ocorrência de latência	elastic apm
	Transações são completas em 10 min. mesmo em erros	elastic apm
	Replicação de dados críticos (como parâmetros de produto e ensaio) ocorre em menos de 10 minutos	calculado (elastic apm)
MONITORAM.	Aplicação exporta logs	calculado (elastic apm)
	Aplicação possui *endpoint* para *check* de saúde	A definir
	Endpoints de *check* de saúde devem verificar erros dos componentes e/ou serviços externos requeridos	A definir
	Logs devem apresentar informações úteis para rastreio	A definir
	Alterações são corretamente logadas para auditoria?	A definir
SEGURANÇA	Nenhum problema OWASP TOP 10 identificado	breakman, tfsec, ZAP
	Dependências não devem ter nenhum registo CVE aberto.	OWASP dependency check
	Imagem de container não apresenta vulnerabilidade.	trivy
	Nenhuma chave ou segredo deve ser encontrado.	git-secrets
	Rotação de credenciais deve ser realizada a cada mês.	calculado (elastic)
	Rotação de certificados deve ser realizada a cada 3 meses.	calculado (elastic)
DESEMPENHO	Transações são completas em menos de 1 segundo.	elastic apm
	Salvamento de resultado não deve demorar mais que 1 seg.	elastic apm
	Recuperação de parâmetros não deve demorar mais que 500ms (caso não esteja em cache)	calculado (elastic apm)
	Recuperação de parâmetros não deve demorar mais que 200ms quando estiver em cache.	calculado (aplicação)
	Parâmetros buscados devem ficar em cache por 24 horas.	calculado (aplicação)

Tabela 3- Fitness Functions definidas para o projeto