UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

BSI Sistemas de Informação UFRPE

QA Metrics

Integração das métricas de qualidade de software, em ambiente Docker, para exibição de dashboards Grafana alimentado pelo banco de dados temporal InfluxDB via Newman

---

Relatório Técnico relativo ao Trabalho de Conclusão de Curso do Bacharelado em Sistemas de Informação na modalidade Empresa

---

Aluno
Lucas Ferreira da Silva

Orientador
Silvana Bocanegra
DEINFO/UFRPE

Co-orientador
Rodrigo Elia Assad
DEINFO/UFRPE

15 de julho de 2022

Lucas Ferreira da Silva

QA Metrics: Integração das métricas de qualidade de software, em ambiente Docker, para exibição de dashboards Grafana alimentado pelo banco de dados temporal InfluxDB via Newman

Relatório Técnico apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientadores: Silvana Bocanegra e Rodrigo Elia Assad

Recife

15 de julho de 2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

S586q Silva, Lucas Ferreira da
QA Metrics: integração das métricas de qualidade de software, em ambiente Docker, para exibição de dashboards Grafana alimentado pelo banco de dados temporal InfluxDB via Newman / Lucas Ferreira da Silva. - 2022.
53 f. : il.

Orientadora: Silvana Bocanegra.
Coorientador: Rodrigo Elia Assad.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Sistemas da Informação, Recife, 2022.

1. docker. 2. grafana. 3. influxdb. 4. newman. 5. qualidade. I. Bocanegra, Silvana, orient. II. Assad, Rodrigo Elia, coorient. III. Título

CDD 004

Lucas Ferreira da Silva

QA Metrics: Integração das métricas de qualidade de software, em ambiente Docker, para exibição de dashboards Grafana alimentado pelo banco de dados temporal InfluxDB via Newman

Relatório Técnico apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado em: 03 de Junho de 2022.

BANCA EXAMINADORA

Dra. Silvana Bocanegra (Orientador)
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Dr. Rodrigo Elia Assad (Coorientador)
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Dr. Guilherme Vilar
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Dedico este trabalho a minha saudosa mãe que acreditou no meu potencial, incentivou-me a aceitar novos desafios e ajudou-me a ser quem sou hoje: uma pessoa que preza pelo bem-estar da família.

Agradecimentos

Gostaria, em primeiro lugar, de agradecer ao meu bom Deus pela Sua grande misericórdia e pelas bênçãos derramadas na minha vida e na vida dos meus amigos e familiares tão queridos.

Agradeço a minha família (saudosa mãe, avó, pai, irmãs Maria e Edna) pelo suporte, paciência, confiança e por sempre acreditarem em mim e no meu desejo incansável de construir algo que beneficie a todos. Obrigado a todos e, principalmente, a minha linda noiva Yane Merik que será minha futura e única esposa no dia 03/12/22. Sou muito grato por ter você ao meu lado nesta jornada.

E, por último e não menos importante, agradeço aos meus orientadores Silvana Bocanegra e Rodrigo Assad por me auxiliarem nesse trabalho. Agradeço a todos os professores e servidores do curso Bacharelado em Sistemas de Informação por todo auxílio e conhecimento transmitidos durante toda a minha trajetória nesta graduação que me proporcionou um intercâmbio incrível no Canadá.

Muitíssimo Obrigado!

“Viver é como andar de bicicleta. É preciso estar em constante movimento para manter o equilíbrio.”
(Albert Einstein)

Resumo

As métricas e indicadores de qualidade de software são capazes de auxiliar um testador de software, conhecido como Quality Assurance (QA), a avaliar o que precisa ser realizado para melhorar o desempenho de um projeto de desenvolvimento de software. Além disso, possibilita monitorar o progresso de um projeto a fim de sugerir iniciativas baseadas nos dados coletados. Entretanto, unificar as métricas obtidas de fontes distintas para apresentá-las em tempo real não é uma tarefa fácil para algumas empresas. Apesar disso, com o uso de APIs, é possível coletar os dados, analisar e apresentá-los em dashboards, além de reduzir o tempo de retrabalho dos QAs. Neste trabalho, foi desenvolvido um sistema capaz de coletar dados de três serviços com o intuito de apresentar, em dashboards, as métricas que são essenciais na área de Qualidade de Software. Ademais, a coleta e o armazenamento destas métricas são realizados de forma automatizada e orquestrada através de uma aplicação compacta, rápida, eficiente, segura, portátil e isolada.

Palavras-chave: docker, grafana, influxdb, jira, newman, qualidade, sonarqube.

Abstract

Software Quality metrics and indicators are able to help a software tester, commonly known as QA, to assess what needs to be done to improve performance of a software development project. In addition, it makes it possible to monitor the progress of a project in order to suggest initiatives based on the collected data. However, gathering the metrics obtained from different data sources to present them in real time is not an easy task for some companies. Nevertheless, with the use of APIs, it is possible to collect the data, analyze and present them in dashboards and reduce QA rework. In this paper, it was developed a system capable of collecting data from three services in order to present, in dashboards, the metrics that are essential in the area of Software Quality. Furthermore, the collection and storage of these metrics are performed in an automated and orchestrated manner through a compact, fast, efficient, safe, portable and isolated application.

Keywords: docker, grafana, influxdb, jira, newman, quality, sonarqube.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API Application Programming Interface.

CAPES Coordenação de Aperfeiçoamento de Pessoal de Nível Superior.

CEO Chief Executive Officer.

CI Continuous Integration.

CLI Command-Line Interface.

CONAB Companhia Nacional de Abastecimento.

CsF Ciência sem Fronteiras.

cURL Client URL.

GCP Google Cloud Platform.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

JQL Jira Query Language.

JS JavaScript.

JSON JavaScript Object Notation.

JWT JSON Web Tokens.

MVP Minimum Viable Product.

NPS Net Promoter Score.

NYSE New York Stock Exchange.

QA Quality Assurance.

RegEx Regular Expression.

REST Representational State Transfer.

SENAI Serviço Nacional de Aprendizagem Industrial.

SO Sistema Operacional.

SQL Structured Query Language.

TC Test Case.

TCP Transmission Control Protocol.

TSDB Time Series Database.

UDP User Datagram Protocol.

UFRPE Universidade Federal Rural de Pernambuco.

UI User Interface.

URL Uniform Resource Locator.

UX User Experience.

VM Virtual Machine.

XML Extensible Markup Language.

YAML YAML Ain’t Markup Language.

Sumário

1 Introdução

O processo de desenvolvimento de software é composto por diversas etapas que variam entre as empresas conforme seu modelo de trabalho. As etapas mais comuns são: levantamento e análise dos requisitos do sistema; definição do projeto; implementação do software; etapa de testes; implantação e manutenção [1].

A etapa de levantamento e análise dos requisitos do sistema é responsável pela identificação das necessidades do cliente a fim de definir o escopo do trabalho e, assim, poder precificar o projeto.

Na etapa de definição do projeto, diversos aspectos podem ser avaliados a começar pela parte técnica, isto é, o stack tecnológico para desenvolvimento do projeto. Aqui a linguagem de programação, a arquitetura de software e hardware, User Experience (UX), User Interface (UI), etc., serão mapeados, discutidos e definidos.

Quanto à implementação do software, é onde o programa ganhará vida. Muitas empresas criam Minimum Viable Products (MVPs) e fazem uso da metodologia de desenvolvimento Scrum para incrementar o código em um curto espaço de tempo. A etapa de implementação ou, simplesmente, desenvolvimento é a que mais demanda tempo e é a mais longa dentro do processo.

A etapa de testes, na maioria das vezes, é executada logo após a implementação do código (ou parte dele). No entanto, esta etapa pode e deve ser executada durante todo o ciclo de desenvolvimento, isto é, antes, durante e depois do código implementado. Vale ressaltar que o foco desta etapa é garantir que a qualidade esteja dentro dos padrões pré-estabelecidos pelo cliente, ou seja, atendendo os critérios de aceite e não apresentando bugs. Existem diversas fases nas quais o código é submetido à testes, podendo ser: unitário, integração, componente, end-to-end, entre outros.

1.1 Contexto

Este trabalho está associado ao campo da qualidade de software e visa auxiliar os QAs quanto à coleta de métricas e indicadores para tomada de decisões mediante os resultados exibidos no dashboard. Além disso, a ideia é reduzir o tempo de coleta de dados e possibilitar que qualquer membro do time possa executá-lo independentemente do QA, inclusive em sua ausência.

1.2 Problema e solução proposta

Os Engenheiros de Qualidade (QAs) da squad, no contexto do cliente da empresa CI&T, coletam métricas de qualidade da pod e armazenam-nas em planilhas distintas. Antes da cerimônia de retrospectiva da sprint, estas planilhas são acessadas online e os gráficos previamente criados e associados aos dados das métricas são copiados e colados em uma apresentação do Google Slides para, logo depois, exibir as métricas da pod. No entanto, os desenvolvedores, arquitetos de software e os Scrum Masters não conhecem todo o processo sobre como são as planilhas compartilhadas entre os QAs e muito menos como elas são utilizadas. Por este motivo, na ausência de um QA de uma das pods da squad, o time fica impossibilitado de analisar as métricas e tomar as devidas ações.

Existem diversas formas para solucionar este problema. Porém, é crucial que as necessidades dos stakeholders sejam levantadas a fim de atingir um consenso. Neste trabalho, uma alternativa que soluciona o problema supracitado é o desenvolvimento de um sistema integrado, automatizado e disponível de maneira portátil em um ambiente isolado capaz de ser executado em qualquer Sistema Operacional (SO). Assim como, reduzindo o tempo despendido para coleta e exibição destes dados.

Desta forma, as métricas do Jira referentes aos bugs, os dados de cobertura de código mensurados pelo SonarQube e a distribuição de testes criados nos níveis de testes (unidade, integração, componente e end-to-end) da Pirâmide de Testes serão centralizados e disponíveis rapidamente. Como resultado, a produtividade dos QAs aumentará e todos poderão apresentar e analisar as métricas de qualidade durante a retrospectiva e, consequentemente, levantar os Action Items da sprint.

1.3 Justificativa

Na tentativa de se reinventar nos negócios e em busca da redução de custos e perdas, muitas empresas recorrem a processos que as auxiliem quanto ao aumento da competitividade e do seu desempenho em seus projetos. Trazendo essa perspectiva de melhoria contínua para a área de tecnologia, mais que programar linhas de código, o desenvolvimento de um software requer uma gestão de projeto diretamente integrada à análise sobre os dados gerados e coletados em seu processo.

Um bom gerenciamento pode diminuir e evitar riscos, mitigar impactos não desejados, manter prazos e custos dentro do estipulado [2]. É importante que o gerente de projetos implante métricas de qualidade a fim de avaliar como o tempo dispendido impacta no custo e escopo como um todo. Estas métricas podem mensurar diversos fatores a depender do objetivo delas. No contexto de desenvolvimento de software, as métricas de qualidade são criadas a partir de um histórico estatístico.

Segundo Sommerville, “A qualidade de software não é diretamente comparável à qualidade da manufatura. A ideia de tolerâncias não é aplicável aos sistemas digitais” [3]. Em outras palavras, as métricas de qualidade utilizadas para gerenciar um projeto são um instrumento crucial no processo de calcular e definir prazos, estimar o esforço requerido para o cumprimento das atividades desenvolvidas pela equipe e, não menos importante, estipular e controlar os recursos inerentes ao projeto. Os benefícios oriundos pelo uso de métricas não são apenas para a empresa, mas também para o cliente dela, pois o acompanhamento e a análise das métricas de qualidade auxiliam-na a atender as necessidades do cliente em tempo hábil, além de garantir as correções dentro do prazo estipulado.

Os QAs da empresa **CI&T**[1], a cada sprint, realizam um levantamento quanto aos bugs reportados pelos membros dos times da squad no Jira. Assim como, verificam se existem code smells remanescentes e/ou novos, analisam a cobertura de código através do SonarQube, além de coletar os testes das camadas da Pirâmide de Testes através de um script que varre cada repositório dos serviços.

Normalmente, as atividades realizadas pelos QAs demandam muito tempo e, por esta razão e a fim de otimizar a criação de métricas para retrospectiva dos times da squad, o **QA Metrics** (nome dado ao sistema desenvolvido neste trabalho) foi idealizado. Nele, todas as métricas da sprint atual são agregadas de forma que todos os membros tenham em mãos um sistema para análise e monitoramento de bugs, cobertura de código e acompanhamento da evolução da Pirâmide de Testes.

[1] https://ciandt.com

2 A empresa e sua atuação

2.1 CI&T

A **CI&T** é uma empresa multinacional brasileira fundada em 1995 na cidade de Campinas, interior do estado de São Paulo, pelo Chief Executive Officer (CEO) **César Gon**, pelo Presidente **Bruno Guiçardi** e pelo Chairman **Fernando Matt**. À época, os três fundadores eram estudantes de Engenharia da Computação na Universidade Estadual de Campinas (UNICAMP)[2] e decidiram fundar a empresa em um quarto alugado, em uma modesta casa que abrigava apenas três computadores usados [4].

Atualmente, a empresa conta com diversos escritórios no Brasil e no exterior (EUA, Canadá, Reino Unido, Portugal, Japão, China, Austrália e Colômbia) com cerca de 6.000 funcionários atuando de forma presencial, remota ou híbrida. Vale ressaltar que a CI&T fundou outras empresas tal como a **Sensedia**[3], realizou algumas aquisições nos últimos anos (**Dextra**[4] e **Somo**[5]) e planeja outras.

Por 15 anos consecutivos, a CI&T é certificada como "Great Place to Work" pelo **GPTW Institute**[6] e, em 2021, subiu do décimo primeiro para o quinto lugar no ranking. No mesmo ano, passou a ter seu mercado de capitais aberto na **New York Stock Exchange (NYSE)**[7] sob o ticker **CINT**[8].

2.1.1 Setor

Usando práticas Lean, Agile e DevOps em seu processo de desenvolvimento, a CI&T atende inúmeras indústrias com suas soluções lineares. Os principais setores de atuação são: Varejo, Bens de Consumo Embalados, Mídia & Entretenimento, Serviços Financeiros, Healthcare e Tecnologia.

2.1.2 Produtos

Por atuar no ramo de consultoria, a CI&T possui um portfólio de escopo aberto visando poder personalizar os serviços conforme a necessidade de seus clientes. Por isso, é esperado que diversos projetos sejam desenvolvidos, adaptados aos modelos de negócio e otimizados tanto em desempenho como em experiência da jornada do usuário, impulsionando o Net Promoter Score (NPS) [5].

Basicamente, a empresa atua com integração de sistemas de desenvolvimento de software, engenharia de software e Nearshore (o que acaba gerando um custo menor).

[2] https://www.unicamp.br
[3] https://www.sensedia.com
[4] https://www.dextra.com.br
[5] https://www.somoglobal.com
[6] https://gptw.com.br
[7] https://www.nyse.com
[8] https://www.nyse.com/quote/XNYS:CINT

2.1.3 Clientes

Atuando globalmente, a CI&T possui uma lista de clientes extensa, acima dos 150. No entanto, 98 clientes são responsáveis por gerar mais de R$ 1 milhão por ano à empresa (cada), dentre estes estão: Cielo, Google, HP, Itaú, Johnson & Johnson, Motorola, Nestlé, Porto Seguro, SulAmérica, Vivo, Bradesco e AB-Inbev [6].

2.1.4 Faturamento

O faturamento da CI&T em 2021 foi R$ 1,4 bilhão, o que equivale a um aumento de 51% na comparação com 2020. O lucro líquido sofreu um aumento de 43%, ou seja, bateu R$ 508 milhões. Este crescimento foi devido à desconcentração do faturamento e ao aumento no número de clientes geradores de receita acima de R$ 1 milhão por ano que subiu de 58 para 94 clientes em um ano [7].

A expectativa da empresa é de que o faturamento em 2022 seja aproximadamente 56% maior em relação a 2021, ou seja, R$ 2,25 bilhões com taxa cambial média do Dólar Americano a R$ 5,20.

2.1.5 Estrutura Organizacional

A CI&T passou a adotar o modelo mais horizontal do que vertical, tendo em vista que não existe mais a sala do CEO ou a necessidade de uma pessoa que precise intermediar uma comunicação com ele. ”O modelo tradicional de gestão hierárquica foi rompido porque numa empresa muito burocrática os setores não conversam entre si”, disse o então diretor de marketing, **Marcelo Trevisani**.

2.2 Atuando como Quality Assurance

A área de qualidade da empresa é fundamental para a garantia de que os projetos sejam entregues com baixa quantidade de defeitos, falhas e/ou bugs. O objetivo seria entregar um sistema 100% funcional, porém isso nem sempre é possível na área da tecnologia. Por isso, a CI&T compõe times não apenas com desenvolvedores de software, mas também com QAs responsáveis por:

- Disseminar a cultura de qualidade para todo o time;
- Identificar e reportar bugs não detectados durante a fase de desenvolvimento do software;
- Documentar os planos e os casos de testes.

Alocado durante um ano e seis meses na CI&T como QA, foi possível ser promovido do cargo nível pleno para nível sênior numa squad com cerca de 40 pessoas, sendo um QA para cada um dos quatro times (ou pod). As tarefas designadas para o papel de QA nas pods da squad foram: (1) participar de todos os rituais Scrum; (2) criar e manter atualizados os planos, casos e ciclos de testes; (3) validar os casos de testes das estórias de usuário com o critério de aceite; (4) reportar e/ou (re)testar bugs; (5) acompanhar as métricas de qualidade do SonarCloud (seção 3.2.6), bugs reportados no Jira (seção 3.2.5) e Pirâmide de Testes (seção 3.2.8) e apresentá-las durante a retrospectiva da sprint.

3 Desenvolvimento realizado na empresa

Este capítulo tem como objetivo abordar os conceitos teóricos fundamentais para a realização deste trabalho. Primeiramente demonstrando a problemática e a solução proposta e, depois, as tecnologias utilizadas, a arquitetura e a contribuição do QA Metrics para as pods da squad da CI&T.

3.1 A problemática e a solução proposta

Quinzenalmente, cada QA precisa coletar as métricas de qualidade para serem apresentadas durante a retrospectiva da sprint do time (pod). Esse processo demanda muito tempo dos QAs (cerca de uma a três horas para cada um deles) por ser necessário realizar as seguintes atividades:

1. Acessar os cinco projetos da squad no SonarCloud (descrito mais adiante na seção 3.2.6) e transferir os valores atuais (*coverage*, *new coverage*, *code smells* e *new code smells*) para a planilha correspondente de forma manual;
2. Filtrar os bugs e sub-bugs criados durante a sprint e exibir as métricas no dashboard do Jira (descrito mais adiante na seção 3.2.5). Apesar da possibilidade de exibir tais métricas nos dashboards criados no Jira, eles possuem limitações quanto ao compartilhamento e, muitas vezes, os dados não ficam visíveis aos demais usuários ao tentar acessá-los;
3. Executar o script do Test Reader (descrito mais adiante na seção 3.2.8) a fim de exportar os testes mapeados no código-fonte dos projetos para a planilha da Pirâmide de Testes correspondente de forma manual e exibir no slide da retrospectiva um gráfico não muito intuitivo.

Outro problema identificado diz respeito às ausências dos QAs devido a questões pessoais, férias, saúde ou de força maior. Pois, sempre que um deles estiver ausente na data da retrospectiva, o time precisa buscar ajuda de um QA de outra pod da squad para realizar as atividades supracitadas. Na maioria das vezes, o time com o QA ausente não tem o apoio de outro QA por diversos motivos. Por consequência, as métricas de qualidade não são apresentadas durante a retrospectiva da sprint causada exatamente pela dificuldade e/ou pelo desconhecimento dos desenvolvedores.

A fim de solucionar esse problema e semi-automatizar as atividades enumeradas acima, foi desenvolvido um sistema responsável por agregar as métricas de qualidade e exibi-las em dashboards. Nestes dashboards, as métricas de qualidade incluem: detalhes sobre os bugs criados no Jira durante a sprint; o nível de cobertura de código e os code smells; e a Pirâmide de Testes dos projetos.

Por conseguinte, em vez de despender entre uma a três horas, cada QA poderá coletar as métricas em questão de segundos, e no máximo 5 minutos. Além disso, qualquer membro do time poderá coletar e apresentar as métricas sem necessidade de um QA de outra pod, pois basta instanciar os serviços do Docker Compose (descrito mais adiante na seção 3.2.1) e exibir as métricas de qualidade da sprint em qualquer navegador.

3.2 Tecnologias utilizadas

Nesta seção, o Tech Stack utilizado para a realização deste projeto de software é apresentado. O desenvolvimento do sistema baseia-se, em parte, na linguagem de programação **JavaScript (JS)**. As ferramentas e plataformas utilizadas neste trabalho serão melhor descritas nas próximas subseções.

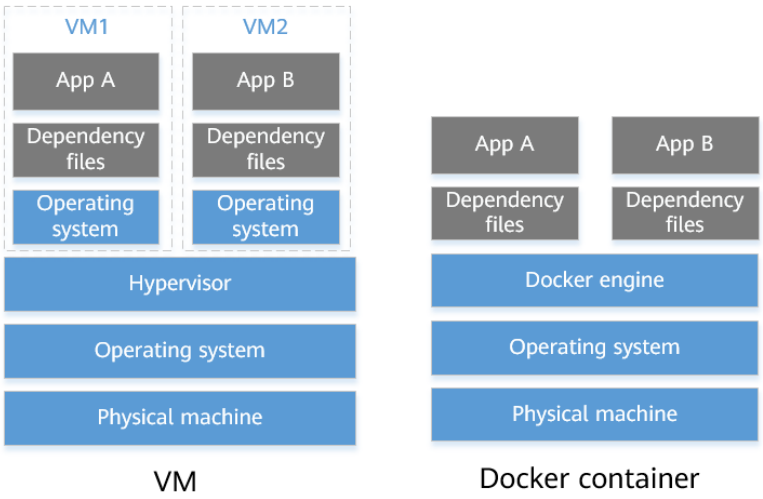
3.2.1 Docker

O **Docker** é uma plataforma *open source*, desenvolvida na linguagem de programação **GO** pela empresa **Docker, Inc**[9] (antiga **dotCloud, Inc**) em 2013, com o intuito de facilitar a criação e o gerenciamento de ambientes isolados [8]. Estes ambientes, por sua vez, são conhecidos como contêineres (*containers*), os quais são responsáveis por empacotar uma aplicação e torná-la portátil, isto é, podendo ser executada em qualquer ambiente de software que possua o Docker instalado. Com isso, é possível criar, implantar, copiar e migrar aplicações Docker de um ambiente para outro com total flexibilidade, mantendo a aplicação intacta e pronta para ser executada independentemente do SO.

Uma pesquisa realizada pela empresa **Stack Overflow**[10] em 2022 [9], mostra que o Docker ficou em primeiro lugar como a ferramenta mais amada e desejada entre os desenvolvedores. Cerca de 27.000 participantes (76,93%) afirmaram que amam utilizar o Docker e cerca de 22.000 (37,08%) expressaram interesse em utilizá-lo, um aumento aproximado de 25% em relação ao ano anterior. Segundo o site **HG Insights** quase 120.000 empresas utilizam o Docker, dentre elas destacam-se: Fisher Investments, Ad Hoc Team, Blue Origin, Wind River, Travelers e Dun & Bradstreet [10].

A arquitetura do Docker difere em muito em relação a uma Virtual Machine (VM) (Máquina Virtual). O primeiro ponto a destacar, observando a Figura 1, é que o Docker não precisa rodar um SO para que sua aplicação seja inicializada. Em comparação com a VM, o Docker tem a vantagem de poder instanciar contêineres compartilhando o mesmo Kernel e não ser dependente de uma infraestrutura específica, pois com o Docker Engine qualquer aplicação encapsulada via Docker é executada [11].

Figura 1: Composição Lógica de uma Máquina Virtual e de um contêiner Docker



Fonte: Huawei [11]

[9] https://www.docker.com
[10] https://stackoverflow.com

Ainda à título de comparação, considerando a Tabela 1 é possível concluir que o contêiner Docker possui um nível de isolamento baixo, conseguindo inicializar uma aplicação em questão de segundos. As imagens são portáteis e possuem tamanho de até alguns Megabytes. Além disso, possui uma perda de desempenho abaixo de 2% e pode executar até 1.000 aplicações em um único servidor.

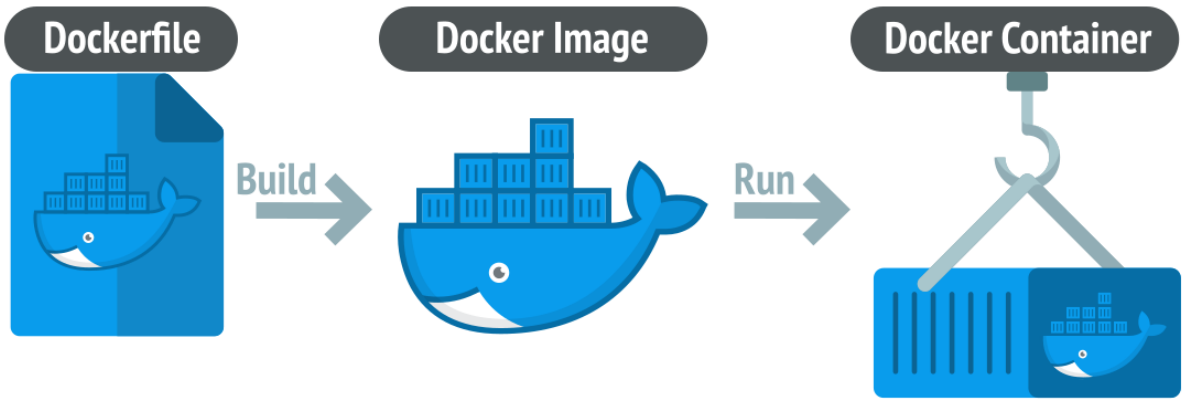
Tabela 1: Comparação entre um Contêiner Docker e uma Máquina Virtual

Item	**Contêiner Docker**	**Máquina Virtual**
Nível de Isolamento	Baixo	Alto
Tempo Necessário para Inicialização	Segundos	Minutos
Tamanho da Imagem	Alguns megabytes	Centenas de megabytes a vários gigabytes
Perda no Desempenho de Execução	Abaixo de 2%	Cerca de 15%
Portabilidade da Imagem	Não relacionado à plataforma	Relacionado à plataforma
Densidade (aplicações em um único servidor)	100 a 1.000	10 a 100
Segurança	1. Quando o privilégio de um usuário em um contêiner é escalado de um usuário comum para o usuário **root**, o usuário ganha permissões de **root** da máquina hospedeira. 2. O isolamento de hardware não está implementado, por isso os contêineres são vulneráveis a ataques.	1. As permissões **root** de um hóspede na Máquina Virtual são isoladas das permissões da máquina hospedeira. 2. O isolamento de hardware está implementado para prevenir o escape das Máquinas Virtuais e troca de dados.

Fonte: Test Reader [12] (adaptado)

O fluxo para criação de um contêiner Docker é simples. Conforme ilustrado na Figura 2, um arquivo **Dockerfile** com instruções de execução é processado (*build*) para gerar uma imagem Docker. Uma imagem do Docker é um sistema de arquivos que fornece os arquivos de programa, registro, recurso e configuração necessária para a execução de contêineres. Uma imagem do Docker também contém alguns parâmetros de configuração necessários para execução, mas não contém dados dinâmicos. Portanto, o conteúdo dela não será alterado após a criação. Uma imagem do Docker pode ser usada para instanciar (run) contêineres do Docker. Uma única imagem Docker existente em um dispositivo pode ser utilizada para instalar vários contêineres do Docker.

Figura 2: Processo para criação de um contêiner Docker



Fonte: LinuxIAC [13]

Um contêiner Docker é uma instância executável de uma imagem Docker. O Docker usa contêineres para executar aplicativos e cada contêiner é uma plataforma isolada e segura, podendo ser considerado como um ambiente leve de execução do Linux.

Docker Compose

O **Docker Compose**[11] é uma ferramenta para definição e execução de múltiplos contêineres Docker, também conhecido como orquestrador. Com esta ferramenta, é possível definir/configurar todos os parâmetros necessários para executar cada contêiner a partir de um arquivo de definição [14]. Neste arquivo, conhecido como **docker-compose.yml** e estruturado em formato **YAML Ain't Markup Language (YAML)**, cada contêiner é definido como um serviço. Cada serviço, por sua vez, é composto por especificações, tais como: imagem, volume, rede, porta, variável de ambiente, dependência entre serviços, comandos, modo de reinicialização, entre outros [15].

Conforme ilustrado na Figura 3, o arquivo *docker-compose.yml*[12] está definido para usar a estrutura da versão 3.9 (**version: '3.9'**) e instanciará um serviço nomeado como **db** (**services: db:**).

Figura 3: Docker Compose - Arquivo docker-compose.yml

Fonte: Prisma [16]

As especificações do serviço **db** definidas no arquivo *docker-compose.yml* da Figura 3 incluem:

- A imagem mais recente do **PostgreSQL** (`image: postgres:latest`);
- O modo de reinicialização em caso de falha configurado como *always* (`restart: always`);
- A definição do nome do contêiner (`container_name: integration-test-prisma`);
- A exposição da porta **5432** na porta **5433** (`ports: - '5433:5432'`);
- As variáveis de ambiente do PostgreSQL (`environment: POSTGRES_USER: prisma...`).

Com o arquivo *docker-compose.yml* corretamente definido, um único comando (`docker-compose up`) é utilizado para instanciar o serviço **db**. Caso este arquivo tivesse 10 serviços configurados, todos eles seriam instanciados sem a necessidade de fazer isso via Command-Line Interface (CLI).

Diante dos benefícios que o Docker e o Docker Compose oferecem, em termos de infraestrutura, rápida inicialização e fácil integração dos contêineres (através das imagens **grafana**, **newman** e **influxdb**), foi decidida a sua aplicação neste projeto. Pois, com eles, qualquer membro do time pode executá-lo sem a necessidade de configurações personalizadas, ou seja, para cada SO.

[11] Disponível em: https://github.com/docker/compose
[12] Consultar especificações em: https://docs.docker.com/compose/compose-file

3.2.2 InfluxDB

O **InfluxDB**[13] é um banco de dados temporal de código aberto, desenvolvido na linguagem de programação **GO** pela empresa **InfluxData**[14] em 2013, voltado para um alto volume de consultas e escritas por segundo. O InfluxDB provê uma sintaxe de consulta parecida com a **Structured Query Language (SQL)**, possibilitando seu uso sem dificuldades por aqueles que já utilizam o SQL [17].

Apesar de ser um banco de dados há pouco tempo no mercado, o InfluxDB tem ganhado espaço em diversas empresas que visam manipular dados estatísticos em tempo real com baixo custo computacional. Segundo a HG Insights, cerca de 6.800 empresas já estão usando o InfluxDB, dentre elas destacam-se: Target, Cisco, Walmart, Power Home Remodeling, Wells Fargo e Amgen [18].

Diferentemente dos demais bancos relacionais e apesar de sua sintaxe ser um subgrupo do SQL, o InfluxDB possui uma linha de protocolo (o mesmo que modelagem de banco) composta por medições (*measurements*), séries (*series*), e pontos (*points*). Uma única linha de texto no formato de protocolo de linha representa um ponto de dados no InfluxDB. A linha de protocolo informa ao InfluxDB a medição do ponto, conjunto de tags, conjunto de campos e a data/hora. Basicamente é uma linha de texto composta por uma medição, uma ou mais tags e pelo menos um campo, sendo estes dois últimos no formato **chave=valor** e, opcionalmente, o timestamp em nanosegundos.

As linhas de protocolo podem ser enviadas via requisições **Hypertext Transfer Protocol (HTTP)**, **Transmission Control Protocol (TCP)** e **User Datagram Protocol (UDP)**. A Figura 4 ilustra a referência da linha de protocolo utilizada para inserção de dados no InfluxDB.

Figura 4: InfluxDB - Linha de Protocolo

```
measurement(,tag_key=tag_val)* field_key=field_val(,field_key_n=field_value_n)* (nanoseconds-timestamp)?
```

Fonte: InfluxData [19]

O bloco de código da Figura 5 mostra um exemplo de protocolo de linha e o divide em seus componentes individuais. A primeira parte é a **medição** (*weather*), isto é, onde os dados serão salvos. A segunda parte é o conjunto de **tags** (*location=us-midwest*) o qual é opcional e, caso exista, deve ser separado por uma vírgula após o nome da medição. Após definido o nome da medição e opcionalmente as tags, resta informar o **campo** (*temperature=82*) e o timestamp (1465839830100400200).

Figura 5: InfluxDB - Sintaxe

```
weather,location=us-midwest temperature=82 1465839830100400200
  |    -------------------- --------------  |
  |             |                |           |
+-----------+--------+-+---------+-+---------+
|measurement|,tag_set| |field_set| |timestamp|
```

Fonte: InfluxData [20]

[13] Disponível em: https://github.com/influxdata/influxdb
[14] https://www.influxdata.com

3.2.3 Grafana

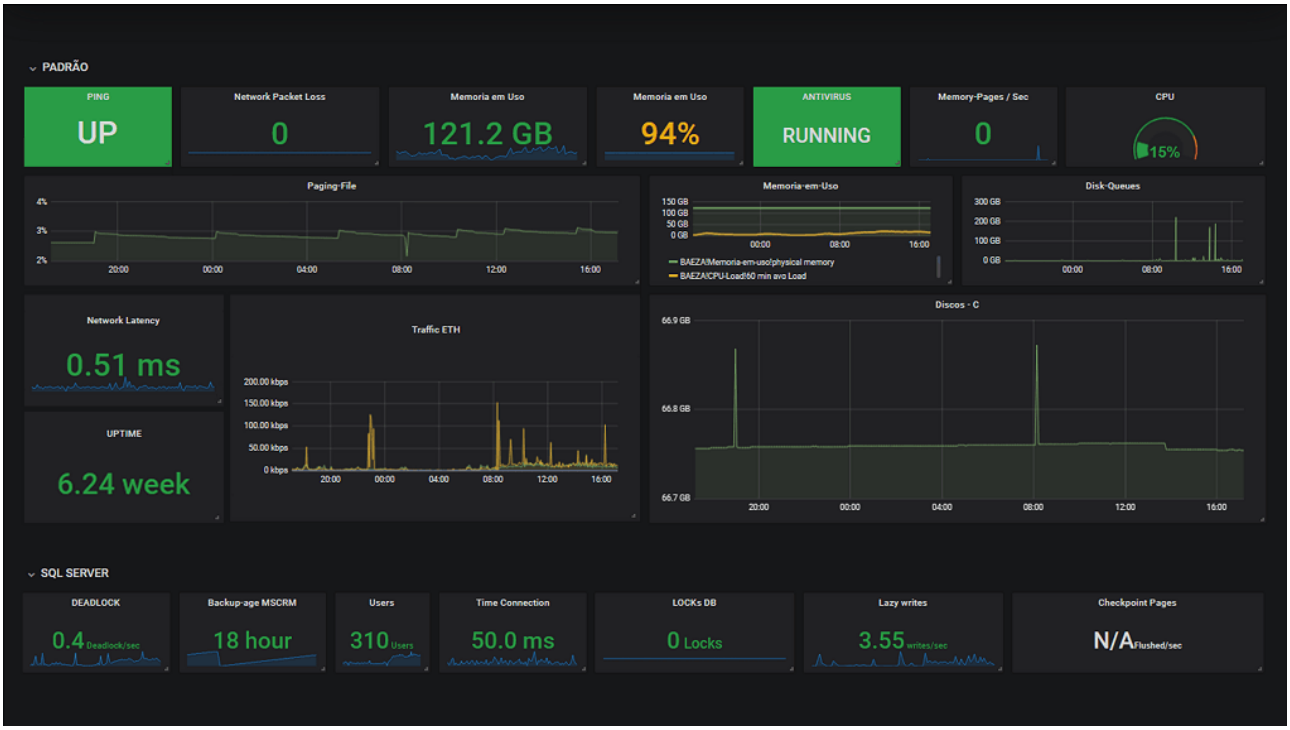
O **Grafana**[15] é uma plataforma *open source*, desenvolvida na linguagem de programação **GO** e **TypeScript** por **Torkel Ödegaard**[16] em 2014, com o intuito de visualizar e analisar métricas por meio de gráficos. O Grafana pode ser instalado em qualquer SO e possui suporte a diversos tipos de bancos de dados, principalmente TSDBs como: **InfluxDB**, **Prometheus**[17] e **Graphite**[18] [21].

A visualização de gráficos pelo Grafana por meio de **dashboards** é dinâmica e pode ser compartilhada com e por toda a equipe. É possível configurar alertas com base nas métricas, as quais são analisadas de forma contínua a fim de notificar o usuário conforme as regras por ele definidas. O Grafana é muito utilizado por sistemas de monitoramento para gerar gráficos em tempo real.

Segundo a HG Insights, cerca de 24.000 empresas já estão usando o Grafana, dentre elas destacam-se: Verizon, American Express, JPMorgan Chase, Capital One, Boeing e myGwork [22]. Atualmente administrada pela empresa **Grafana Labs**[19] e mesmo sendo *open source*, o Grafana está disponível em três versões: Grafana, Grafana Cloud (*Free*, *Pro* e *Advanced*) e Grafana Enterprise.

Esta ferramenta possui uma estrutura leve, flexível, de fácil instalação, com uma comunidade rica e colaborativa. Além disso, não é limitada a bancos de dados temporais, pois pode se integrar a outras ferramentas e a outras fontes de dados, tais como: **MySQL**, **PostgreSQL** e **Microsoft SQL Server**. A Figura 6 ilustra métricas obtidas do serviço nomeado OpMon via Microsoft SQL Server.

Figura 6: Dashboard Grafana



Fonte: OpServices [21]

[15] Disponível em: https://github.com/grafana
[16] https://grafana.com/author/torkel
[17] Disponível em: https://github.com/prometheus/prometheus
[18] Disponível em: https://github.com/graphite-project/graphite-web
[19] https://grafana.com

3.2.4 Newman

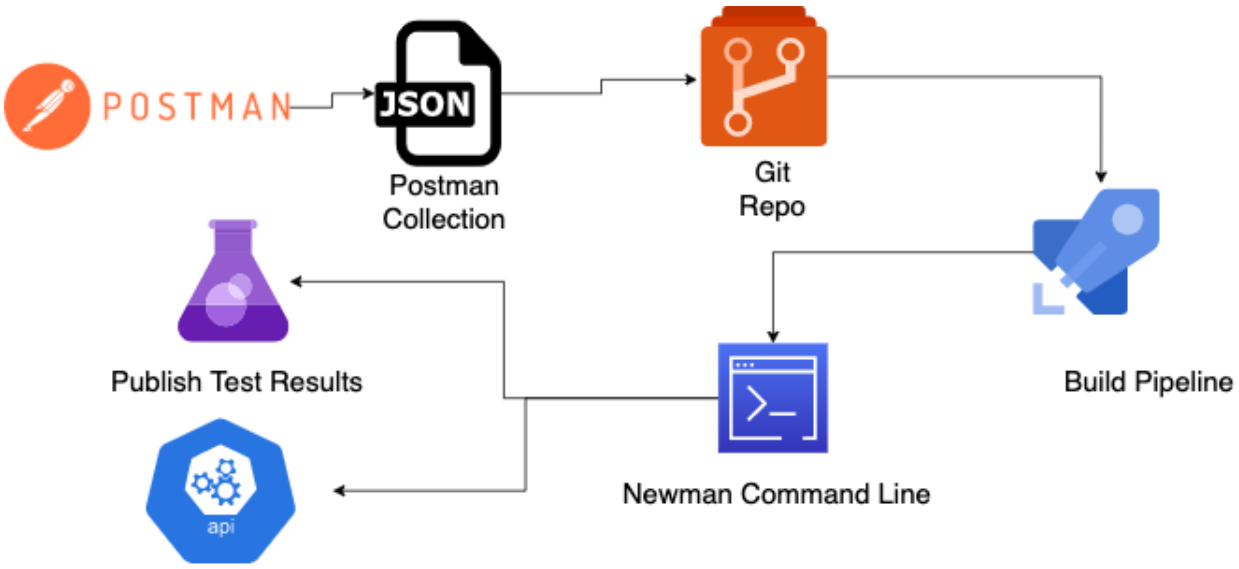
O **Newman**[20] é uma ferramenta *open source*, desenvolvida na linguagem de programação **JS** pela empresa **Postman, Inc**[21] em 2018, com o intuito de executar coleções de requisições HTTP para testar uma aplicação REST API (Representational State Transfer - Application Programming Interface) através do CLI, ou seja, via linha de comando no terminal. É possível integrar o Newman com sistemas de Integração Contínua, ou Continuous Integration (CI), que é o caso do **Jenkins** [23].

Com o Newman a execução do conjunto de requisições e testes ficou menos exaustiva e possibilitou seu uso diretamente através da linha de comando do SO. Além disso, é possível gerar relatórios nos formatos HyperText Markup Language (HTML), Extensible Markup Language (XML) e JavaScript Object Notation (JSON), podendo este último ser utilizado em ambientes de integração contínua, conforme supracitado. Também é possível exibir a execução e o resultado dos testes no próprio CLI.

O Newman está integrado ao **Postman**, porém ele pode ser instalado separadamente ou executado via Docker. Segundo a HG Insights, cerca de 33.000 empresas já estão usando o Postman, dentre elas destacam-se: U.S. Bank, AT&T, Caterpillar Inc, Pearson, Fiserv, myGwork [24].

Quanto ao CI, a Figura 7 ilustra uma automação de testes de API usando o Newman. O arquivo a ser executado no Newman CLI é exportado do Postman no formato JSON, versionado em um repositório Git para uso durante a execução (*build*) da pipeline. Esta, por sua vez, executará o arquivo JSON no Newman CLI, o qual realizará as requisições e os testes da API para, em seguida, publicar o resultado dos testes. Este processo pode durar apenas alguns segundos, a depender da quantidade de requisições realizadas, do número de testes criados e do tempo de resposta da aplicação sob teste.

Figura 7: Newman - Execução Integrada à Pipeline



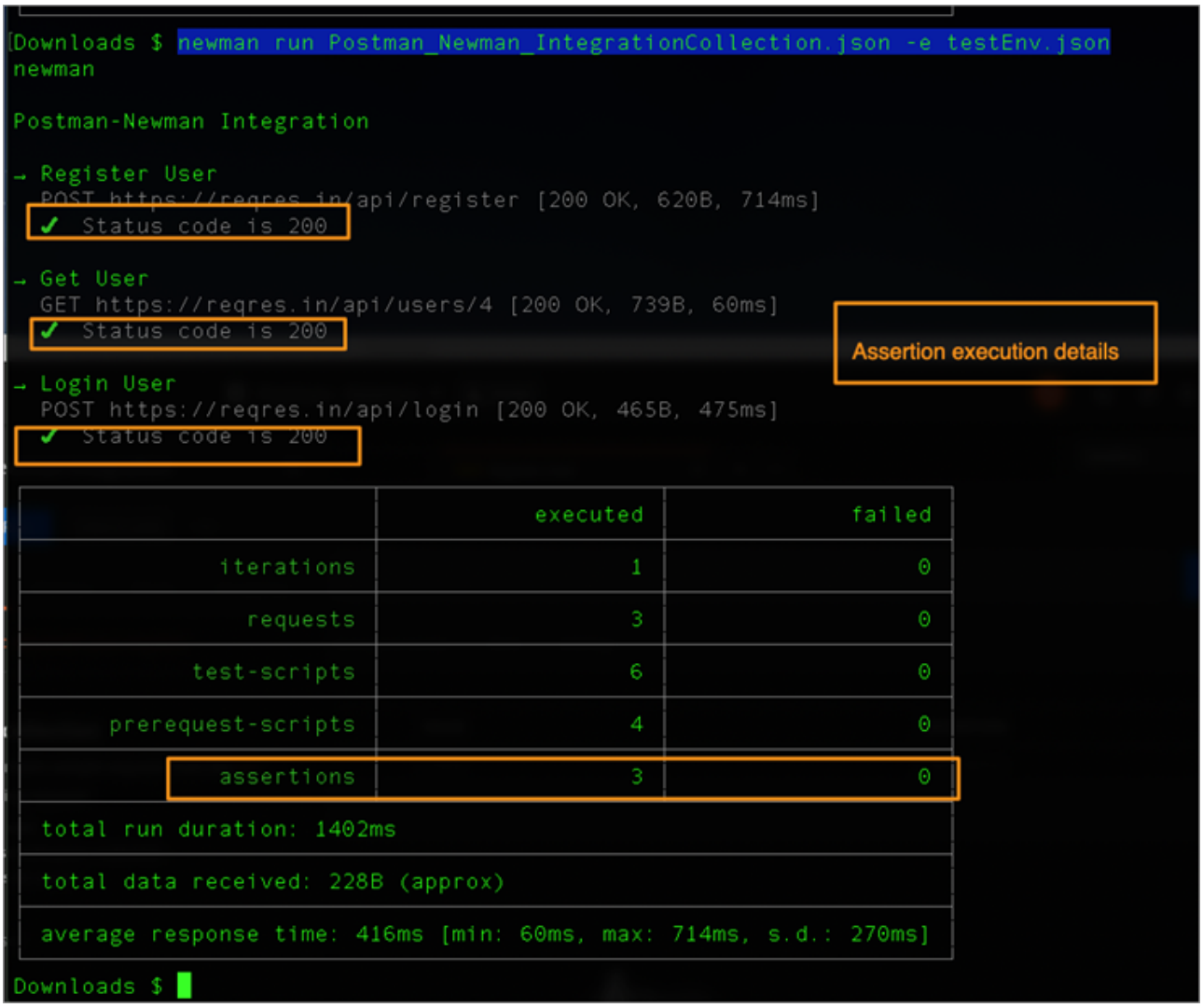
Fonte: Better Programming [25]

[20] Disponível em: https://github.com/postmanlabs/newman
[21] https://www.postman.com

A Figura 8 ilustra a execução de uma instrução, via CLI, na qual dois arquivos JSON são utilizados. O primeiro arquivo (`Postman_Newman_IntegrationCollection.json`) é uma coleção (*collection*) onde estão definidas as requisições (*requests*). O segundo arquivo (`-e testEnv.json`) é onde as variáveis de ambiente, utilizadas pelas requisições da coleção, estão definidas. Ambos os arquivos são utilizados em conjunto durante a execução do comando `newman run` a fim de validar os testes declarados (*assertions*) nas 3 requisições da coleção **Postman-Newman Integration**, a saber:

- **Register User** com código de status HTTP 200;
- **Get User** com código de status HTTP 200;
- **Login User** com código de status HTTP 200.

Figura 8: Newman - Execução via Terminal



Fonte: Software Testing Help [26]

Ao executar `newman run Postman_Newman_IntegrationCollection.json -e testEnv.json` as três requisições e os três testes, ambos definidos na coleção, são executados com sucesso.

3.2.5 Jira

O **Jira**[22] é uma ferramenta comercial desenvolvida pela empresa **Atlassian Corporation Plc** em 2004, voltada para gerenciar atividades de projetos de desenvolvimento de software baseados na Metodologia Ágil[23]. Além disso, permite que os usuários planejem, rastreiem e gerenciem backlogs, sprints, estórias, bugs e distribuição de tarefas entre as equipes de software. Alguns dos recursos ofertados são: quadros Kanban e Scrum, roadmaps, relatórios, integrações de ferramentas de desenvolvedor, fluxos de trabalhos e filtros, este dois últimos customizáveis.

Segundo a HG Insights, aproximadamente 160.000 empresas utilizam o Jira, dentre elas se destacam: PGIM Global Short Duration High Yield Fund Inc, Mercury Systems, DTCC, NBCUniversal Media LLC, Carvana e Fidelity Investments [27].

O Jira fornece uma Application Programming Interface (API), através da qual é possível realizar requisições HTTP, tal como pesquisar as atividades criadas (estórias e bugs) com o uso do endpoint `/rest/api/3/search` através do método **POST**. Conforme ilustrado na Figura 9, a Uniform Resource Locator (URL) da requisição (`https://your-domain.atlassian.com/rest/api/3/search`) é definida em conjunto com a credencial do usuário (`email@example.com:<api_token>`) e com os cabeçalhos da requisição (`--header`).

Neste caso, o endpoint é requisitado por um método **POST** contendo o corpo com uma consulta **Jira Query Language (JQL)**[24]. Outra vantagem é a possibilidade de usar a API integrando-a à linguagens como **Forge**, **NodeJS**, **Java**, **Python**, **PHP** e pela biblioteca **Client URL (cURL)**.

Figura 9: Jira - Exemplo de Requisição via cURL

```
curl --request POST \
  --url 'https://your-domain.atlassian.net/rest/api/3/search' \
  --user 'email@example.com:<api_token>' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{
  "jql": "project = HSP",
  "fields": [
    "summary",
    "status",
    "assignee"
  ]
}'
```

Fonte: Atlassian Developer [28]

Atualmente o Jira está disponível nas versões: Jira Cloud com quatro planos (*Free*, *Standard*, *Premium* e *Enterprise*) e o Jira Data Center, para equipes que exigem uma solução autogerenciada.

[22] https://www.atlassian.com/br/software/jira
[23] Metodologia Ágil: https://www.lucidchart.com/blog/pt/o-que-e-metodologia-agil-em-projetos
[24] Disponível em: https://support.atlassian.com/jira-software-cloud/docs/what-is-advanced-searching-in-jira-cloud

3.2.6 SonarCloud

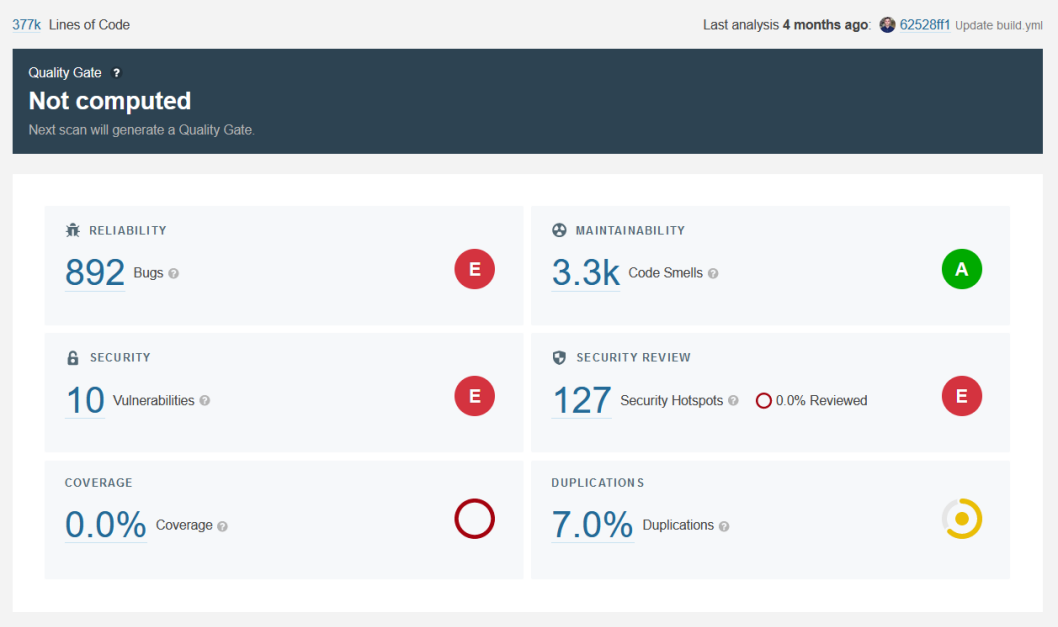
O **SonarCloud**[25] é uma ferramenta open source, desenvolvida pela empresa **SonarSource** em 2018, disponível online com o intuito de inspecionar continuamente o código-fonte de uma aplicação. Para isso, utiliza o **SonarQube** para realizar a varredura do código e analisar possíveis vulnerabilidades, erros e regras específicas para cada linguagem de programação, onde o code smell é um desse erros identificados pelo SonarQube [29]. Devido a sua versatilidade, o SonarCloud pode ser integrado ao pipeline para ser executado em conjunto com a execução (*build*) do código.

Com a análise realizada pelo SonarCloud, durante o processo de compilação da aplicação, é possível detectar: quais os trechos do código-fonte que podem causar bugs; quais linhas de código estão duplicadas; e qual o nível de segurança do código. Outro ponto importante é que o SonarCloud visa incentivar os desenvolvedores a escreverem um código limpo, seguro e de qualidade [30].

Segundo a HG Insights, cerca de 15.000 empresas utilizam o SonarQube, dentre elas se destacam: Peraton, General Motors, Uline, Boeing, Northrop Grumman e a Verizon [31].

Conforme as métricas sumarizadas do SonarCloud ilustradas na Figura 10, é possível observar que existem 10 vulnerabilidades (*Vulnerabilities*) em termos de segurança (*Security*), 127 áreas que podem gerar problemas de segurança (*Security Hotspots*) em termos de Revisão de Segurança (*Security Review*), duplicação de linhas de código em 7,0% (*Duplications*), cobertura de teste em 0% (*Coverage*), 892 bugs (*Reliability*) e 3.300 code smells quanto à manutenibilidade (*Maintainability*). Além destas métricas, pode-se também saber que neste projeto já existem 377 mil linhas de código (*Lines of Code*), que a última análise ocorreu há 4 meses e que o indicador que define imediatamente se o projeto está pronto para produção (*Quality Gate*) ainda não foi computado (*Not computed*).

Figura 10: SonarCloud - Métricas Sumarizadas



Fonte: Milan Milanović [32]

Vale ressaltar que a configuração entre o SonarCloud e o código-fonte é feita de forma simples e rápida através de uma interface Web e, internamente, convertida para o formato YAML.

[25] https://sonarcloud.io

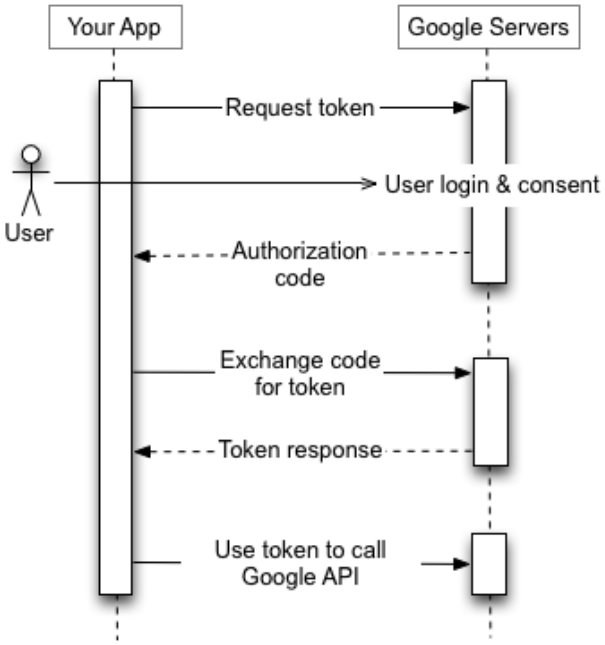
3.2.7 Google Cloud Platform

O **Google Cloud Platform (GCP)**[26] é uma plataforma comercial, desenvolvida em 2008 e mantida pela empresa **Google**, voltada para serviços de Computação em Nuvem (*Cloud Computing*) os quais são executados na mesma infraestrutura utilizada para produtos e serviços da própria Google. É o mais completo software de armazenamento de dados existente hoje no mercado, podendo se adaptar à necessidade de qualquer empresa, transmitindo segurança, praticidade e tranquilidade para a realização de negócios, além de ser muito útil para empresa de qualquer segmento/porte [33].

Segundo a HG Insights, cerca de 40.000 empresas utilizam o GCP, dentre elas se destacam: CloudFlare, Mythical Games, Wayfair, LiveRamp, Caesars Entertainment e Shopify [34] com uma receita em torno de 257 bilhões de dólares em 2021 (sofrendo uma perda de US$ 3,1 bilhões) [35].

Dentro do GCP existe uma gama de serviços e funcionalidades tal como o **Google Sheets API**, a qual é responsável por criar um canal de comunicação entre os usuários e as planilhas da Google. As requisições via API necessitam de um canal seguro conforme ilustra o processo da Figura 11. A sequência de autorização começa quando o aplicativo do usuário redireciona um navegador para um URL da Google; a URL inclui parâmetros de consulta que indicam o tipo de acesso que está sendo solicitado, no caso, um OAuth Access Token. A Google lida com a autenticação do usuário, a seleção da sessão e o consentimento do usuário. O resultado é um código de autorização, que o aplicativo pode trocar por um **Token de Acesso** (*Access Token*) e um **Token de Atualização** (*Refresh Token*). O aplicativo deve armazenar o Token de Atualização para uso futuro e usar o Token de Acesso para acessar uma API da Google. Depois que o Token de Acesso expira, o aplicativo usa o Token de Atualização para obter um novo.

Figura 11: Google Sheets API - Sequência de Autorização



Fonte: Google Identity [36]

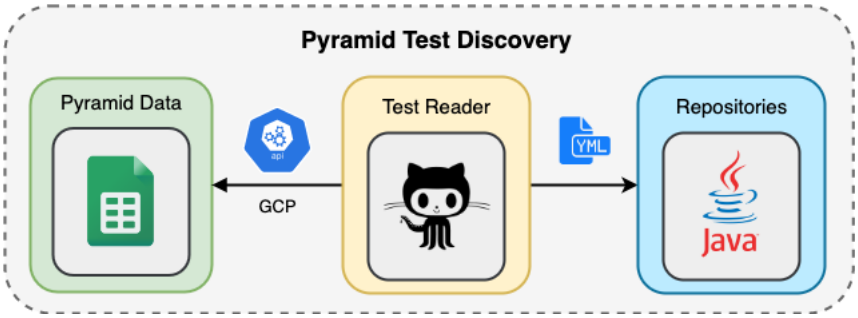
[26] https://cloud.google.com

3.2.8 Test Reader

O **Test Reader**[27] é uma ferramenta open source, desenvolvida na linguagem de programação Python pela **Amanda Quaglio**[28] em 2020, criada para identificar *Test Cases (TCs)* automatizados, isto é, Casos de Teste. Estes, por sua vez, são identificados por meio de Expressões Regulares, conhecidas como Regular Expression (RegEx) e exportados para um repositório online.

A Figura 12 ilustra o fluxo para identificar os TCs e seus respectivos níveis na Pirâmide de Testes. A identificação é realizada a partir de uma varredura no código-fonte do projeto definido no arquivo de configuração que é estruturado no formato YAML, o qual contém as regras de mapeamento para extrair e exportar os TCs para uma planilha via API da Google Sheets disponibilizada pelo GCP.

Figura 12: Arquitetura do Pyramid Test Discovery



Fonte: Elaborado pelo autor

O arquivo de configuração, conforme exemplo ilustrado na Figura 13, utiliza um conjunto de atributos para extrair os TCs de nível Unitário (*Unit*) de um código-fonte. Estes atributos possuem funções específicas durante o processo de identificação e extração dos Casos de Teste. Neste arquivo de configuração estão presentes 6 (vide Tabela 2) dos 11 atributos disponíveis na versão 1.0.

Figura 13: Test Reader - Arquivo de Configuração (exemplo)

```
tests:
  -
    file_name_regex: .*Test\.kt$
    name: Unit
    path: ~/projects/my-android-app/app/src/test/java
    test_rules:
      test_description_regex: "`(.+?)`"
      test_description_strategy: NEXT_LINE
      test_notation: "^@Test+$"
```

Fonte: Test Reader [12]

O Test Reader executa a classe **TestConfigReader** para ler o arquivo de configuração e as variáveis de ambiente (APP_NAME, YAML_CONFIG_PATH, SPREADSHEET_ID, CREDENTIALS_PATH e ROOT_FILE_PATH). Em seguida, a classe **TestDiscovery** faz uma varredura no código-fonte para identificar e extrair os TCs (com seus respectivos níveis e descrições). Por último, a classe **SheetsPublisher** exporta os TCs para uma planilha da Google através da plataforma GCP.

[27] Disponível em: https://github.com/amandaquaglio/test-reader
[28] https://www.linkedin.com/in/amanda-castro-quaglio-mariote-0a538820

Tabela 2: Test Reader - Atributos

Atributo	Descrição	Valor
`file_name_regex`	Define como o TC será nomeado conforme o RegEx	.*Test\.kt$
`name`	Faz o mapeamento do tipo de teste	Unit
`path`	Local para varrer e identificar os TCs	~/projects/my-android-app/app/src/test/java/SumTest.java
`test_description_regex`	Extrai a descrição da linha do TC conforme o RegEx	”‘(.+?)‘”
`test_description_strategy`	Identifica a descrição do TC	NEXT_LINE
`test_notation`	Identifica a anotação de um teste conforme o RegEx	“^@Test+$”

Fonte: Test Reader [12]

A Figura 14 é um exemplo de uma classe desenvolvida na linguagem de programação **Java**, a qual pode ser utilizada pelo Test Reader a fim de identificar se o TC atende aos critérios (ou regras) definidas nos atributos do arquivo de configuração ilustrado na Figura 13. É possível observar que o nome da classe atende ao RegEx definido no atributo `file_name_regex` (Sum**Test**), assim como a anotação atende ao RegEx definido no atributo `test_notation` (**@Test**).

Figura 14: Test Reader - Classe Java (exemplo)

```
class SumTest {
  @Test
  fun `should sum two numbers`() {

  }
}
```

Fonte: Test Reader [12]

Desta forma, é possível extrair a descrição do TC, definida após a anotação, e utilizá-la como nome do arquivo (**~/projects/my-android-app/app/src/test/java/SumTest.java**). Esta descrição foi encontrada a partir da definição do atributo `test_description_strategy` (NEXT_LINE) que extrai o dado da linha logo após a anotação. De igual modo, o nome do TC pode ser extraído de acordo com o RegEx definido no atributo `test_description_regex` (**should sum two numbers**). Por último, os dados extraídos estão prontos para serem exportados para a planilha (vide Tabela 3).

Tabela 3: Test Reader - Resultado da Extração

FileName	TestCaseName	TestType
~/projects/my-android-app/app/src/test/java/SumTest.java	*should sum two numbers*	*Unit*

Fonte: Test Reader [12]

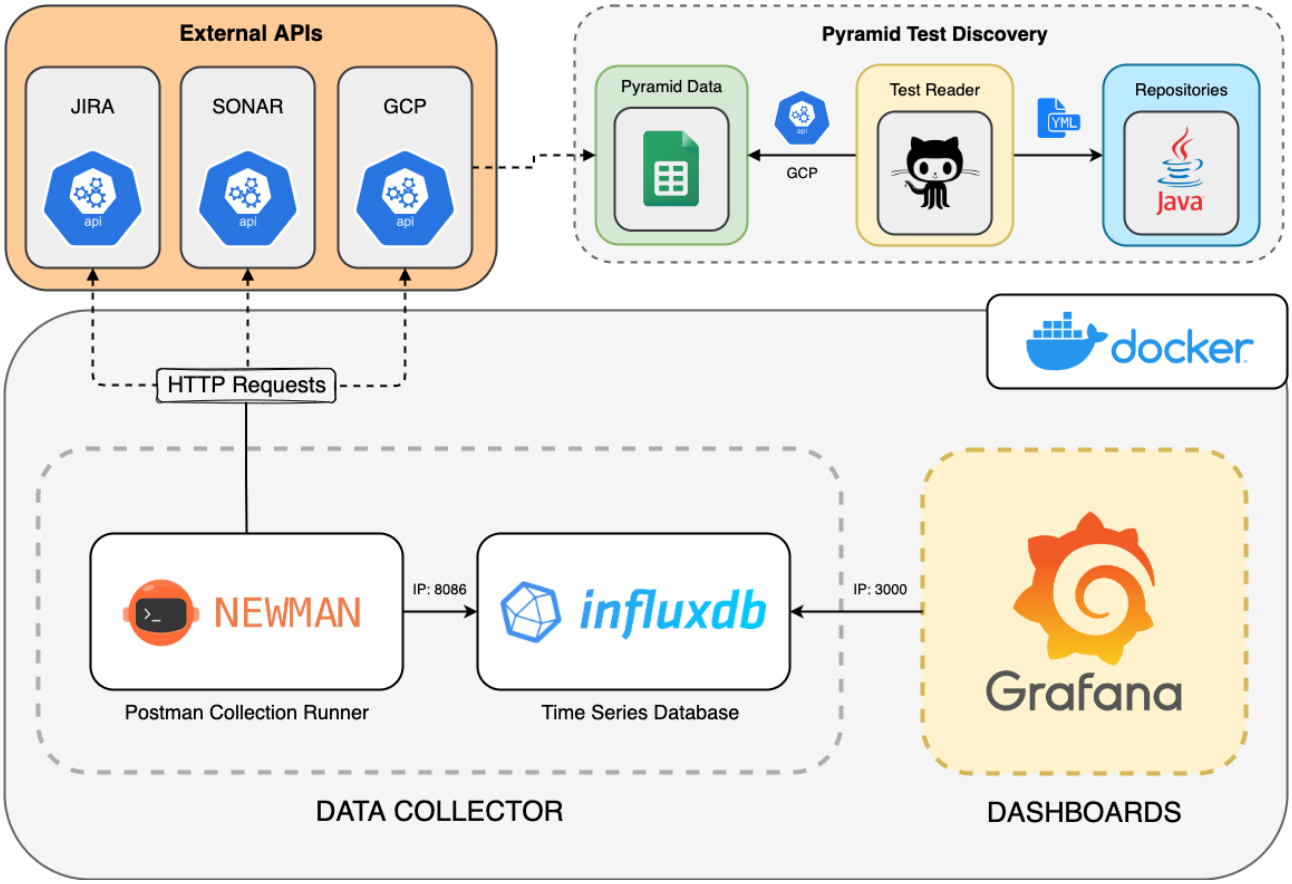
3.3 QA Metrics

Esta seção tem o objetivo de demonstrar como as tecnologias (descritas na seção 3.2) foram, de fato, utilizadas neste projeto. Por motivos de segurança e privacidade, a empresa solicitou que todos os dados e referências ao cliente fossem substituídos ou ocultados. Por esta razão, algumas imagens e dados contidos aqui, e no repositório, do **QA Metrics** estarão com tarjas ou nomes aleatórios.

3.3.1 Arquitetura

Antes de demonstrar como o projeto foi implementado, é importante conhecer a visão macro da arquitetura conceitual do **QA Metrics**. A Figura 15 ilustra a interoperabilidade do **Pyramid Test Discovery** com as **External APIs** (Jira, Sonar, GCP) e do ambiente Docker com as **External APIs**.

Figura 15: Arquitetura do QA Metrics



Fonte: Elaborado pelo autor

O ambiente Docker criado é composto por três serviços, a saber: **newman**, **influxdb** e **grafana**. Os serviços **newman** e o **influxdb** estão agrupados na arquitetura como **Data Collector** (Coletor de Dados) e o serviço **grafana** está representando a plataforma onde são exibidos os dashboards. Por motivos técnicos, a extração de TCs realizada no **Pyramid Test Discovery** é executada manualmente por um dos QAs mediante instruções definidas na configuração do Test Reader. Ao instanciar o QA Metrics com o comando `docker-compose up`, as imagens dos serviços são baixadas, associadas e instanciadas aos respectivos contêineres. Depois, as requisições às APIs das **External APIs** são realizadas e os dados são populados no influxDB para serem exibidos nos dashboards.

3.3.2 Versões das Tecnologias Utilizadas

Na Tabela 4 estão listadas as versões de cada tecnologia utilizada no desenvolvimento deste projeto. É importante que estas versões sejam respeitadas a fim de garantir a perfeita execução do sistema. Exceto, obviamente, quando for necessário atualizá-las certo de que o sistema não sofrerá prejuízos. Por questões de compatibilidade, o influxDB precisa estar limitado à versão 1.8 porque as versões posteriores utilizam o Flux[29], a nova sintaxe funcional para consulta de dados temporais.

Tabela 4: Versões das Tecnologias Utilizadas no QA Metrics

Ferramenta	**Versão**	**Tipo**
Postman Desktop	v9.24.0	Aplicação
Docker Desktop	4.10.1	Aplicação
Docker Compose	2.6.1	Aplicação
Docker Engine	20.10.17	Aplicação
Docker containerd	1.6.6	Runtime
Docker runc	1.1.2	Runtime
Docker init	0.19.0	Runtime
Docker Compose File	3.4	Arquivo de Configuração YAML
Grafana	8.5.0	Imagem Docker hospedada no Docker Hub
Newman	5.3-alpine	Imagem Docker hospedada no Docker Hub
InfluxDB	1.8.10	Imagem Docker hospedada no Docker Hub
InfluxQL (Influx Query Language)	1.8	SQL
InfluxDB API	1.8	REST API
SonarCloud	v1	REST API
Jira Cloud Platform	v3	REST API
Google Cloud Platform - Google Sheets	v4	REST API
Test Reader	1.0	Projeto Python hospedado no Github

Fonte: Test Reader [12]

De igual modo, por questões de compatibilidade, a versão do arquivo `docker-compose.yml`, que atualmente está definida como 3.4, não deve ser modificada para versões superiores porque o formato da estrutura entre elas pode diferir. Também não é garantido quais tecnologias e/ou versões definidas na Tabela 4 podem ser incompatíveis quanto ao upgrade, pois em algum momento alguma funcionalidade de uma das ferramentas utilizadas pode apresentar falhas causadas pela atualização.

Em relação às plataformas **SonarCloud**, **Jira Cloud Platform** e **Google Cloud Platform**, por serem APIs, espera-se que não sofram alterações. Pois, quando isso acontece, uma nova versão é publicada a fim de não impactar projetos desenvolvidos numa determinada versão anterior. No entanto, caso as requisições não sejam realizadas com sucesso, vale verificar se a versão de uma das plataformas ainda está disponível para uso, ou seja, não está como **descontinuada** (*deprecated*).

[29] Disponível em: https://docs.influxdata.com/influxdb/v2.3/query-data/get-started

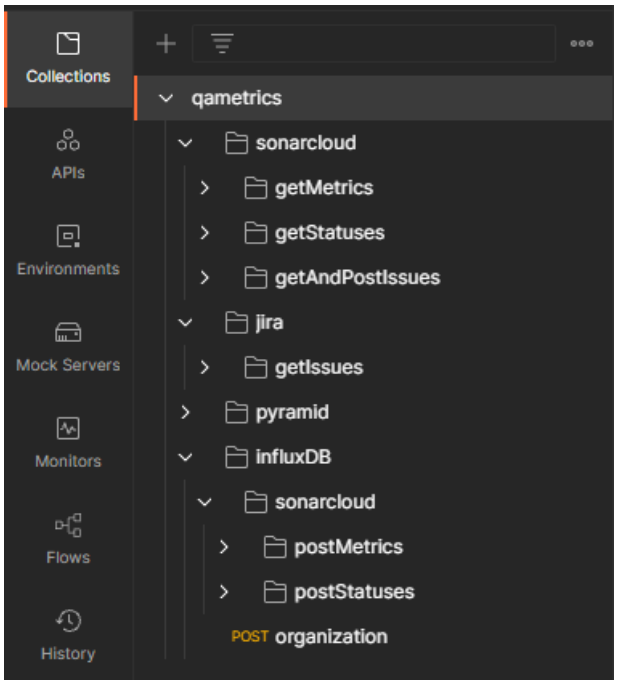
3.3.3 Integração das Tecnologias Utilizadas

Uma vez entendida a arquitetura e as versões utilizadas, resta a integração entre as tecnologias. Primeiramente, após o estudo de viabilidade, ficou definido que a extração dos TCs feita pelo Test Reader seria semiautomatizada, ou seja, um membro qualquer da squad precisará rodar o comando `test-reader-start` manualmente. Desta forma, os TCs e seus níveis correspondentes (extraídos dos repositórios) serão coletados e armazenados na planilha de métricas da Pirâmide de Testes.

Postman/Newman

A primeira atividade realizada foi criar uma coleção (*collection*) no Postman chamada **qametrics**. Dentro desta coleção, foram criadas as requisições (*requests*) organizadas em pastas e subpastas conforme ilustra a Figura 16. As pastas **sonarcloud** e **jira** estão configuradas para incluir o tipo de autorização **Basic Authentication**[30], sendo que o SonarCloud utiliza apenas o campo ***Username*** com a API Key. Quanto ao Jira, os campos ***Username*** e ***Password*** são utilizados com o email e a API Key do usuário, respectivamente. As pastas **pyramid** e **influxDB** não têm autorização definida.

Figura 16: Estrutura das Requisições do Postman



Fonte: Elaborado pelo autor

As requisições possuem variáveis de ambiente (*environments*) que foram definidas dentro do Postman e exportadas juntamente com a coleção (*collection*) **qametrics**. Ambos os arquivos são exportados no formato JSON e utilizados pelo Newman durante a execução do Docker Compose. No arquivo `docker-compose.yml`, no serviço newman, fica o comando responsável por realizar as requisições (`run qametrics.postman_collection.json -e qametrics.postman_environment.json -r cli --bail`), o qual será executado mais uma vez caso ocorra alguma falha durante a requisição. O Newman, conforme configurado no `docker-compose.yml`, depende dos serviços **influxdb** e **grafana** para poder ser instanciado como serviço **newman** via Docker Compose para o QA Metrics.

[30] https://www.ibm.com/docs/en/cics-ts/5.4?topic=concepts-http-basic-authentication

InfluxDB

A segunda atividade realizada foi configurar o serviço **influxdb** no arquivo `docker-compose.yml`. Em seguida, foi definida a linha de protocolo utilizada durante as requisições de escrita no banco de dados chamado **metrics**. Não foi preciso criar nenhuma modelagem porque o InfluxDB não a requer. Após coleta das métricas do Jira, SonarCloud e Pirâmide de Testes por meio das requisições às APIs, faz-se necessário o armazenamento delas no banco de dados. Para isso, a própria coleção **qametrics** do Postman é utilizada para fazer as requisições de escrita no banco de dados **metrics**.

A pasta **influxDB**, da coleção **qametrics**, contém as requisições que inserem as métricas do **sonarcloud** e o nome da organização no banco de dados. Porém, por motivo de desempenho e uso da estrutura de repetição **forEach** do JS, também existem requisições de escrita nas seguintes subpastas: **getAndPostIssues** da pasta sonarcloud; **getIssues** da pasta jira; e na pasta **pyramid**. Acontece que, para cada requisição (*request*), o Newman já realiza a inserção dos dados coletados diretamente no banco **metrics** do InfluxDB através da linha de protocolo correspondente à requisição.

Vale ressaltar que, para quase todas as requisições, os dados foram tratados, formatados e adaptados a fim de permitir seu armazenamento no InfluxDB, uma vez que este modelo de banco de dados não permite salvar os dados de qualquer maneira. Apesar disso, não houve nenhum prejuízo em relação à utilização da ferramenta. O InfluxDB apresentou um desempenho muito satisfatório.

A Figura 17 ilustra a linha de protocolo (vide Figura 4 como referência) utilizada para inserir os bugs encontrados pelo SonarCloud. Como é possível observar, a medida (*measurement*) definida é chamada **metrics**; as tags são *author*, *component*, *file*, *id*, *metric*, *name*, *rule*, *status*, *service*, *severity* e *type*; e os campos são *creationDate*, *debt*, *effort*, *line*, *message*, *tags* e *value*; e o *timestamp*.

Figura 17: InfluxDB - Linha de Protocolo do SonarCloud para Bugs

```
metrics,author,component,file,id,metric,name,rule,status,service,severity,type
creationDate,debt,effort,line,message,tags,value timestamp
```

Fonte: Elaborado pelo autor

A Figura 18 ilustra a linha de protocolo utilizada para inserir as métricas[31] de cada projeto no SonarCloud. Igualmente à linha de protocolo da Figura 17, a medida é **metrics**; as tags são *component*, *id*, *key*, *metric*, *name*, *qualifier*, *type* e *service*; e os campos são *bestValue* e *value*; e o *timestamp*.

Figura 18: InfluxDB - Linha de Protocolo do SonarCloud para Métricas

```
metrics,component,id,key,metric,name,qualifier,type,service bestValue,value timestamp
```

Fonte: Elaborado pelo autor

A Figura 19 ilustra a linha de protocolo utilizada para inserir os *statuses* de cada projeto no SonarCloud. Igualmente à linha de protocolo da Figura 17, a medida é **metrics**; as tags são *branch*, *commit*, *component*, *key*, *metric*, *name*, *type* e *service*; e o campo é apenas o *value*; e o *timestamp*.

[31] coverage (cobertura), new coverage (cobertura de novas linhas), code smell, new code smell (novos)

Figura 19: InfluxDB - Linha de Protocolo do SonarCloud para Status

```
metrics,branch,commit,component,key,metric,name,type,service value timestamp
```

Fonte: Elaborado pelo autor

A Figura 20 ilustra a linha de protocolo utilizada para inserir os bugs reportados pelos membros da squad no Jira durante o período da sprint. A medida definida continua sendo **metrics**; as tags são *component*, *issueSubtask*, *metric*, *service* e *type*; e os campos são *assignee*, *issueType*, *key*, *priority*, *rootCause*, *rootCauseAnalysis*, *bugType*, *businessComponent*, *created*, *creator*, *developmentPhase*, *environment*, *frequency*, *id*, *aggregateProgressPercent*, *projectId*, *projectKey*, *projectName*, *reporter*, *status*, *sprints*, *summary*, *teamName*, *timeEstimate*, *timeSpent* e *value*; e o *timestamp*.

Figura 20: InfluxDB - Linha de Protocolo do Jira

```
metrics,component,issueSubtask,metric,service,type
assignee,issueType,key,priority,rootCause,rootCauseAnalysis,bugType,businessCom
ponent,created,creator,developmentPhase,environment,frequency,id,aggregateProgr
essPercent,projectId,projectKey,projectName,reporter,status,sprints,summary,tea
mName,timeEstimate,timeSpent,value timestamp
```

Fonte: Elaborado pelo autor

A Figura 21 ilustra a linha de protocolo utilizada para inserir os números de testes realizados em cada nível da Pirâmide de Testes. A medida definida continua sendo **metrics**; as tags são *component*, *metric*, *service* e *type*; e os campos são *previous*, *value* e *v0*; e o *timestamp*.

Figura 21: InfluxDB - Linha de Protocolo da Pirâmide de Testes

```
metrics,component,metric,service,type previous,value,vo timestamp
```

Fonte: Elaborado pelo autor

Por último, a Figura 22 ilustra a linha de protocolo utilizada para inserir o nome da organização (cliente) definido na URL do Jira Cloud. A medida definida continua sendo **metrics**; as tags são *component*, *type* e *service*; e o campo é apenas o *organization*; e o *timestamp* definido automaticamente.

Figura 22: InfluxDB - Linha de Protocolo da Organização

```
metrics,component,type,service organization
```

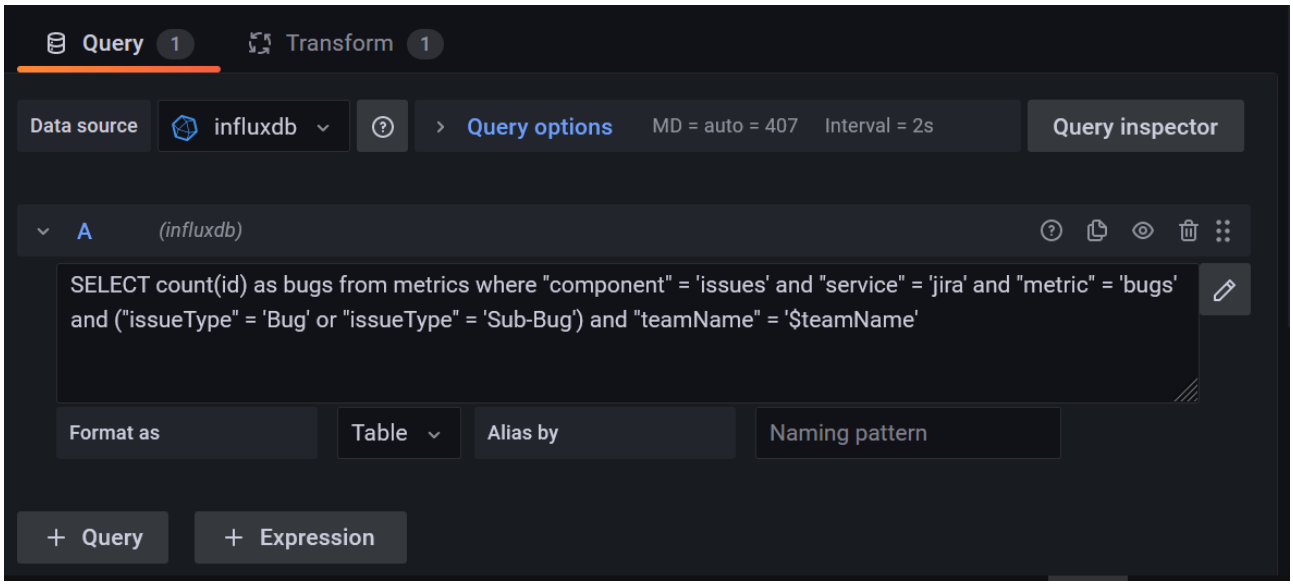
Fonte: Elaborado pelo autor

Grafana

A terceira atividade realizada foi a configuração do serviço **grafana** no arquivo `docker-compose.yml`. O serviço **grafana** possui dependência com os outros dois serviços: **newman** e **influxdb**. Apesar disso, não é garantia que as métricas sejam salvas no banco de dados **metrics** do InfluxDB a tempo de todas elas serem coletadas. Por isso, caso as métricas ainda não estejam sendo exibidas nos dashboards criados, basta aguardar a finalização das requisições às APIs por alguns segundos.

Dentro do Grafana foram criados três dashboards, a saber: **Jira**, **Pyramid** e **Sonar**. Todos eles possuem consultas que são realizadas ao banco de dados **metrics** do serviço **influxdb**. A Figura 23 ilustra uma consulta InfluxQL (Influx Query Language) que seleciona o número de bugs e sub-bugs (bugs associados a uma estória) do Jira e que correspondam ao nome do time (*teamName*). Os dados consultados podem ser formatados como tabela, time series (séries de tempo) ou logs.

Figura 23: Grafana - Consulta ao TSDB via InfluxQL

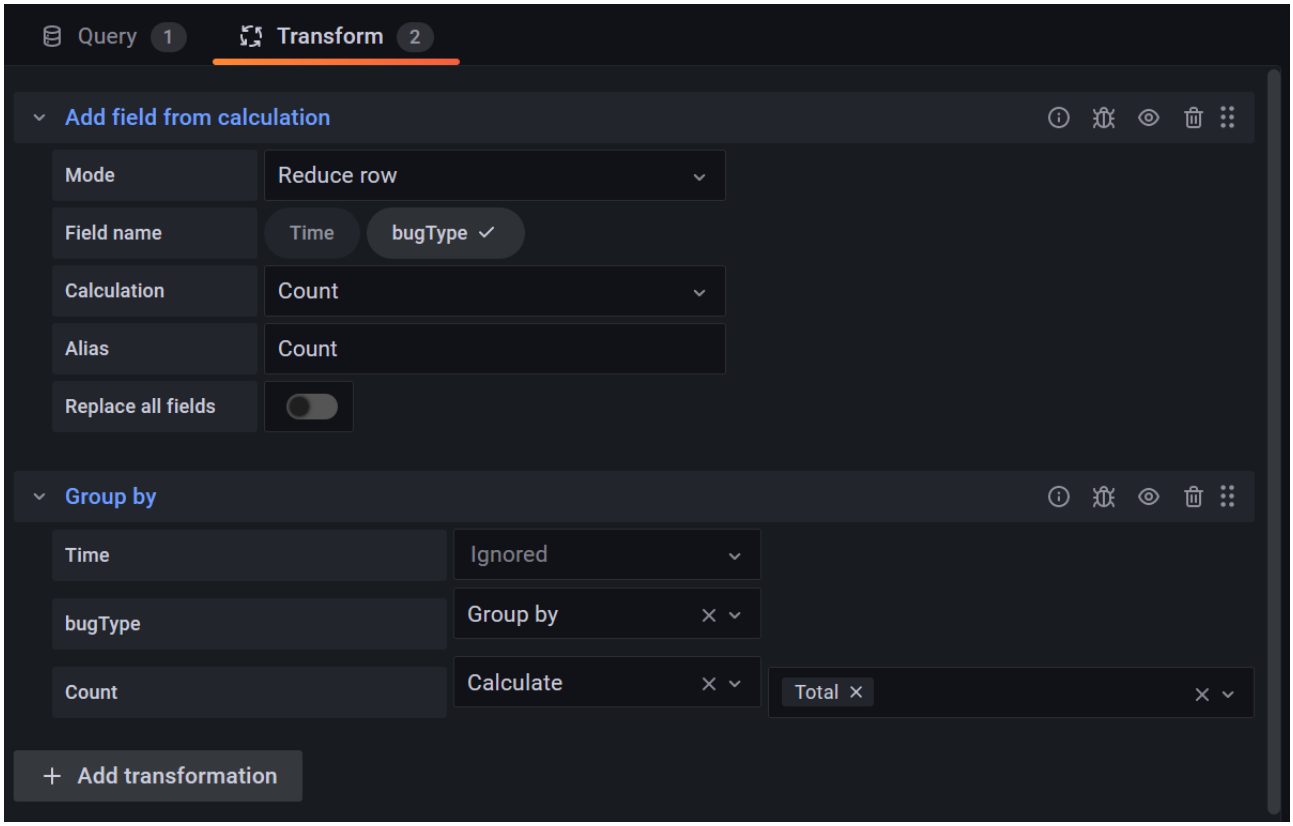


Fonte: Elaborado pelo autor

Para cada um dos painéis (os quais representam gráficos, tabelas, estatísticas, etc.), uma consulta InfluxQL é utilizada. Dentro da edição do painel existem inúmeras possibilidades de transformar a exibição dos dados selecionados, alguns deles são: **Filter by Name** (Filtrar por Nome), **Sort by** (Ordenar por uma coluna específica), **Organize fields** (Organizar Campos), **Convert field type** (Converter tipo de campo), **Add field from calculation** (Adicionar campo para cálculo), **Group by** (Agrupar por um determinado campo), entre tantos outros.

A Figura 24 ilustra a transformação realizada em um gráfico do tipo pizza, no qual um campo para cálculo foi adicionado. Em seguida, foi selecionado o campo **bugType** e aplicada a contagem (*Count*) da quantidade de dados do campo **bugType**, resultando em uma massa de dados contendo uma coluna a mais chamada **Count**, conforme está definido no campo **Alias** da transformação. Logo depois, uma nova transformação é realizada sobre a massa de dados da transformação anterior. Desta vez, os dados foram agrupados pelo campo **bugType** e o campo **Count** passou a calcular o **Total** de vezes em que o tipo do bug (**bugType**) apareceu na consulta. Caso fossem necessárias, novas transformações poderiam ser aplicadas, porém, vale lembrar que elas sobrepõem e utilizam a massa de dados da transformação anterior.

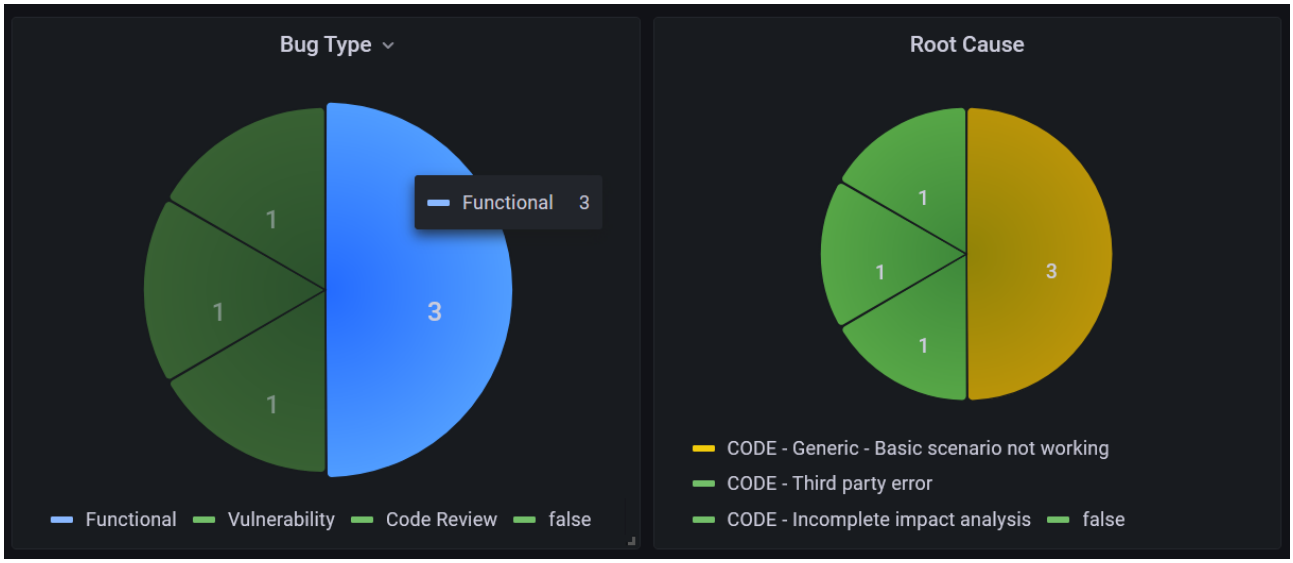
Figura 24: Grafana - Transformação da massa de dados consultada



Fonte: Elaborado pelo autor

Após aplicadas algumas customizações, o Grafana renderiza os elementos visuais (painéis) de forma dinâmica e interativa. O usuário pode passar o mouse sobre os gráficos para destacar o que for de interesse, conforme ilustra a Figura 25, além de poder incluir legendas ou hyperlinks.

Figura 25: Grafana - Gráfico Pizza do Bug Type



Fonte: Elaborado pelo autor

Os dashboards criados foram exportados e adicionados ao código do repositório do QA Metrics. Quando o Docker Compose instancia o Grafana, os arquivos **jira.json**, **pyramid.json** e **sonar.json** (localizados no diretório: `grafana > dashboards`) são provisionados com o serviço **grafana**.

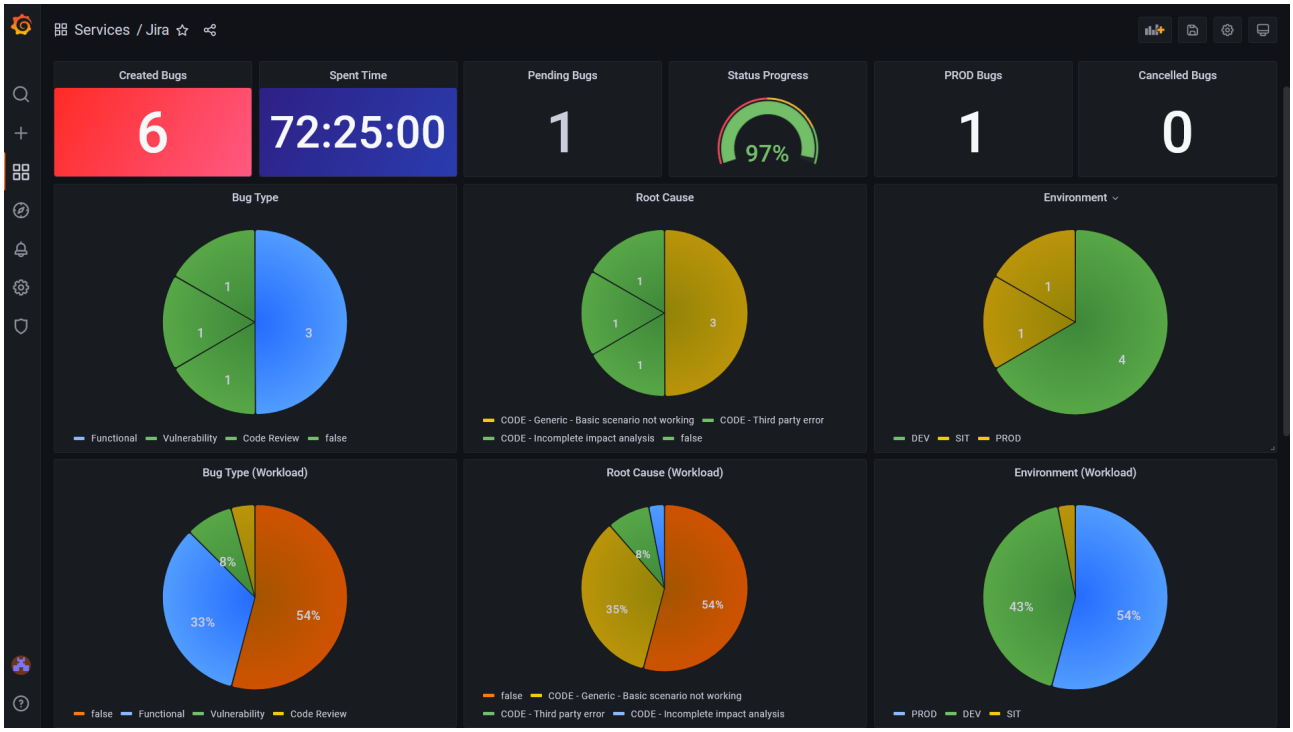
3.3.4 Dashboards

O desenvolvimento do **QA Metrics** resultou em 3 dashboards interativos: **Jira**, **Pyramid** e **Sonar**. A interface de cada um dos dashboards implementados é exibida e descrita em detalhes logo abaixo. Os dashboards ficam acessíveis através de qualquer navegador web no endereço localhost:3000.

Jira Dashboard

No dashboard Jira (vide Figura 26) são exibidas as métricas da sprint para cada time através do filtro **Team Name** localizado no canto superior esquerdo do dashboard (padrão: *Wäls Impetus*).

Figura 26: Jira Dashboard



Fonte: Elaborado pelo autor

- Cartões (*Cards*)
  - **Created Bugs**: Quantidade de bugs/sub-bugs criados;
  - **Spent Time**: Quantidade de horas de retrabalho para corrigir os bugs/sub-bugs;
  - **Pending Bugs**: Quantidade de bugs/sub-bugs que ainda estão pendentes;
  - **Status Progress**: Progresso para correção de bugs/sub-bugs;
  - **PROD Bugs**: Quantidade de bugs reportados no ambiente de produção;
  - **Cancelled Bugs**: Quantidade de bugs/sub-bugs cancelados.
- Gráficos (*Graphs*)
  - **Bug Type**: Os tipos de bugs/sub-bugs reportados;
  - **Bug Type (Workload)**: Percentual de retrabalho em cada tipo de bug/sub-bug;

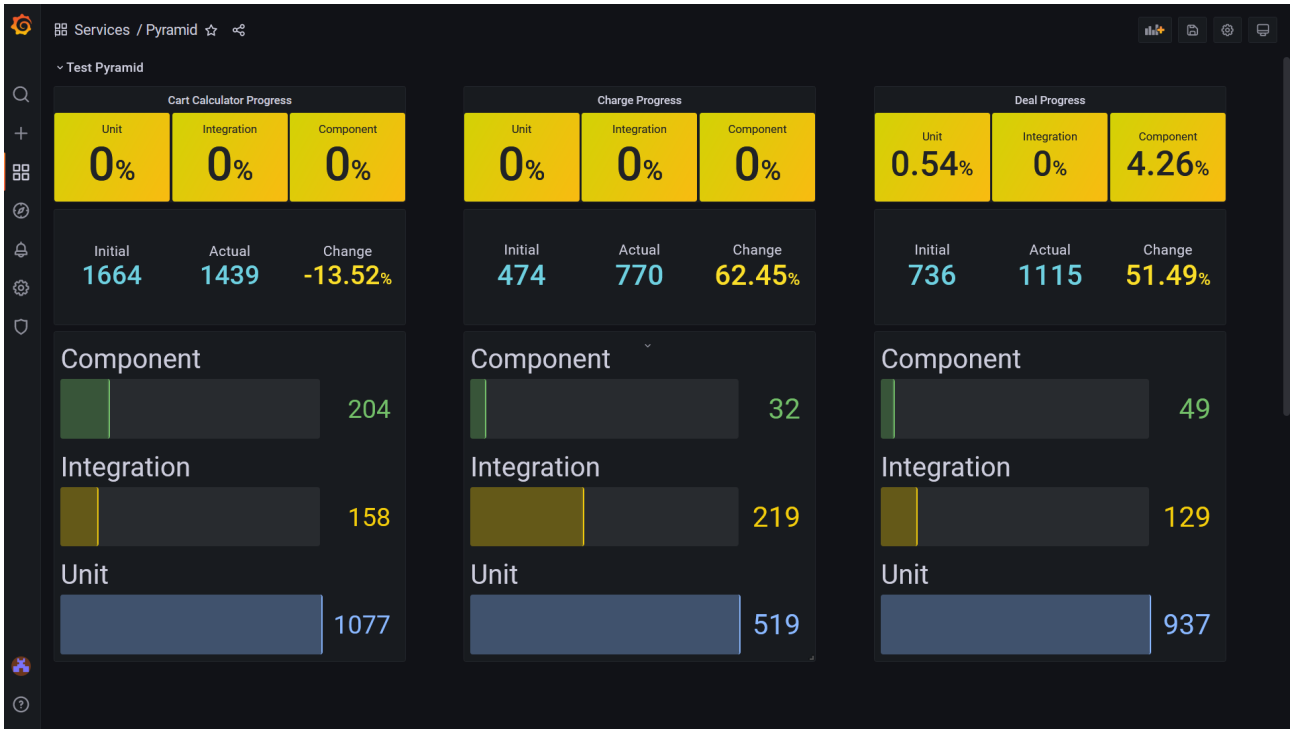
  - **Root Cause**: As causas raízes dos bugs/sub-bugs reportados;
  - **Root Cause (Workload)**: Percentual de retrabalho em cada causa raiz de bug/sub-bug;
  - **Environment**: Os ambientes onde os bugs/sub-bugs foram encontrados;
  - **Environment (Workload)**: Percentual de retrabalho dos bug/sub-bug em cada ambiente.

- Tabelas (*Tables*)
  - **Created Bugs**: Tabela listando todos os bugs/sub-bugs reportados;
  - **Required Fields Missing**: Tabela listando todos os bugs/sub-bugs sem os campos obrigatórios preenchidos. (Nota: todos eles precisam estar preenchidos para fechar a sprint)

Pyramid Dashboard

No dashboard Pyramid (vide Figura 27) são exibidas as métricas da sprint para cada projeto:

Figura 27: Pyramid Dashboard



Fonte: Elaborado pelo autor

- Cartões (*Cards*): Níveis da Pirâmide de Teste em relação à sprint anterior.
  - **Unit**: Progresso do percentual de testes unitários;
  - **Integration**: Progresso do percentual de testes de integração;
  - **Component**: Progresso do percentual de testes de componente;
  - **E2E**: Progresso do percentual de testes end-to-end.

- Cartões (*Cards*): Estatística comparativa entre o estágio inicial e o atual.
  - **Initial**: Número total de testes criados no início (v0), isto é, quando implementado;

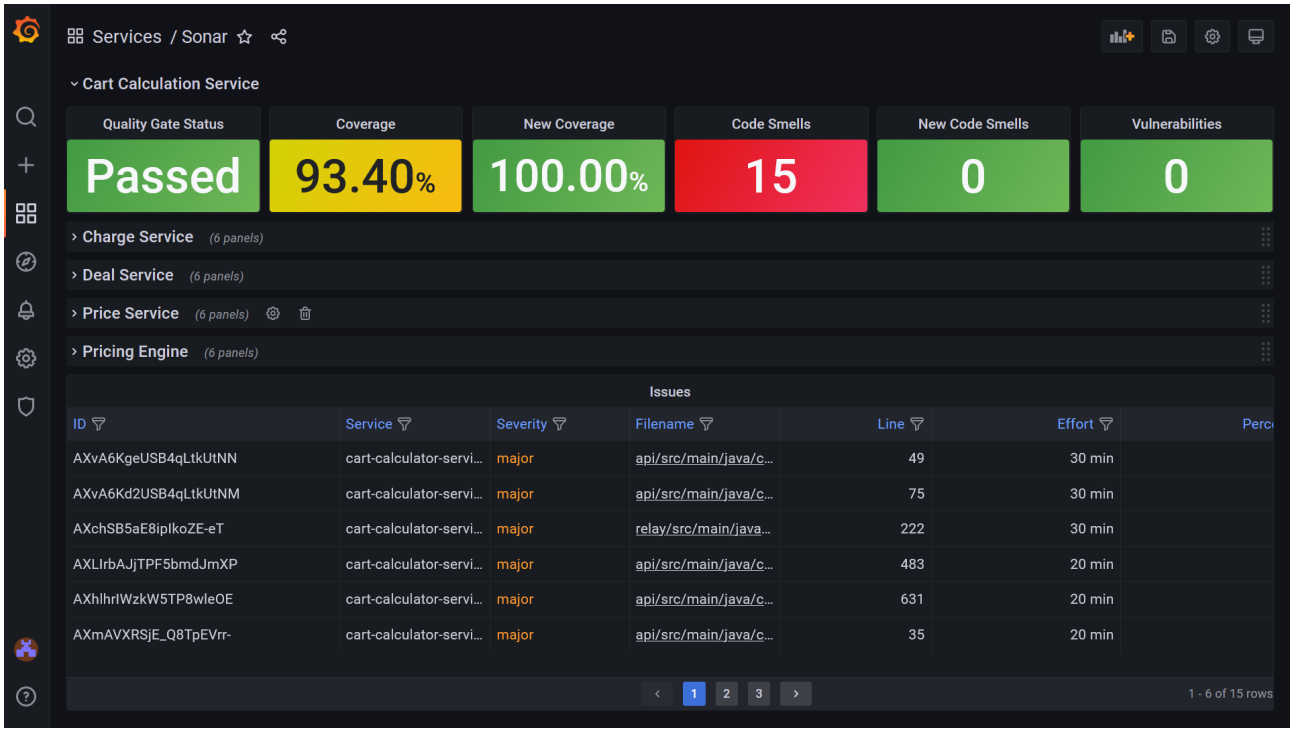
  - **Actual**: Número total de testes criados até o presente momento (v1);
  - **Change**: Progresso do percentual entre o estágio inicial e o atual.

- Gráficos (*Graphs*)
  - **Unit**: Número total de testes unitários;
  - **Integration**: Número total de testes de integração;
  - **Component**: Número total de testes de componente;
  - **E2E**: Número total de testes end-to-end.

Sonar Dashboard

No dashboard Sonar (vide Figura 28) são exibidas as métricas da sprint com relação à cobertura de testes, bugs, vulnerabilidades, status e code smells para cada projeto:

Figura 28: Sonar Dashboard



Fonte: Elaborado pelo autor

- Cartões (*Cards*)
  - **Quality Gate Status**: Informação sobre o status do projeto, se passou ou falhou;
  - **Coverage**: Percentual geral da cobertura de testes do projeto;
  - **New Coverage**: Percentual de cobertura de testes do projeto (novas linhas de código);
  - **Code Smells**: Quantidade geral de código que não atende aos critérios definidos;
  - **New Code Smells**: Quantidade de código novo que não atende aos critérios de qualidade.

- Tabela (*Table*)
  - **Issues**: Tabela listando todos os code smells dos projetos a fim de que o time os corrija;

3.4 Contribuição

Com o uso do **QA Metrics**, todos os membros da squad passaram a ter acesso às métricas de qualidade e a realizar as análises dos indicadores (gráficos, tabelas e estatísticas) por conta própria. Isso só foi possível com a automação do processo de coleta e exibição de métricas em dashboards. Uma das contribuições deste projeto foi a redução desse processo de quase 10 horas para 4 minutos.

Como a squad, no escopo do cliente da empresa CI&T, está dividida em 4 pods e cada uma delas possui até 1 QA, cada um deles precisava de aproximadamente 180 minutos por sprint para criar a apresentação da retrospectiva. A Tabela 5 mostra a comparação da aplicação de recursos (tempo) antes e depois da implementação do sistema QA Metrics dentro da squad do cliente da empresa.

Tabela 5: Comparação de uso do QA Metrics

QA / Pod	**Antes do QA Metrics**	**Depois do QA Metrics**	**Resultado** [(1 - v1/v0) * 100]
QA Pod 1	2,5 horas (180 minutos)	1 minuto	99,45%
QA Pod 2	2,5 horas (180 minutos)	1 minuto	99,45%
QA Pod 3	2,5 horas (180 minutos)	1 minuto	99,45%
QA Pod 4	2,5 horas (180 minutos)	1 minuto	99,45%
Totais	**10 horas (720 minutos)**	**4 minutos**	**Economia de 716 minutos (99,86%)**

Fonte: Elaborado pelo autor

Com a economia de aproximadamente 10 horas, o uso deste recurso (tempo) tão precioso pode e deve ser destinado para outras atividades mais importantes que ainda precisem de ações manuais e repetitivas, como não era o caso do processo de coleta, integração e análise de métricas de qualidade no contexto desta squad para uma retrospectiva de sprint. A transferência de alguns gráficos contidos nas planilhas (jira, sonar, pirâmide) não será mais necessária, tendo em vista que os dashboards por si só já permitem uma demonstração satisfatória, interativa e em tempo real.

Com isso, foram eliminados muitos processos recorrentes, tais como: acessar os 5 projetos no SonarCloud e, em cada projeto, coletar as métricas de cobertura e code smells para, em seguida, preencher a respectiva planilha. Depois era preciso atualizar um filtro dentro do Jira para exportar para a respectiva planilha os bugs e sub-bugs criados durante a sprint. E, por último, criar uma apresentação no Google Slides para a retrospectiva e copiar os gráficos de cada planilha para ela.

Os demais QAs ficaram responsáveis por solicitar à equipe de arquitetura a implantação deste projeto no ambiente de produção, uma vez que não pertenço mais ao quadro de funcionários da empresa. Desta forma, todos os membros da squad poderão fazer uso do QA Metrics sem a necessidade de ficar instanciando os serviços Docker localmente. No entanto, esse tipo de solicitação é complexa porque o cliente é global e impõe muitas regras de segurança. Com isso, o projeto precisará passar por análise de outros setores. No mais, não houve nenhum impedimento quanto ao desenvolvimento deste projeto que, inclusive, foi acolhido por todos os membros da squad durante os meses de estudo, pesquisa, prototipação, implementação e implantação do QA Metrics.

4 Dificuldades encontradas

A realização deste trabalho demandou uma série de atividades que envolveram duas frentes: pesquisa e desenvolvimento. Devido ao prévio conhecimento em algumas das tecnologias utilizadas, a curva de aprendizagem e o grau de complexidade foram minimizados apesar de toda dificuldade. A seguir, estão as maiores dificuldades encontradas durante o desenvolvimento deste trabalho.

Privacidade

Devido a questão de privacidade do cliente da CI&T, foi necessária a remoção de termos e palavras que o identificassem. Como resultado, os dados mencionando o nome do projeto, do cliente e dos membros da squad foram substituídos a fim de preservar a confidencialidade requerida pela empresa.

Docker/Docker Compose

O planejamento do fluxo de coleta e armazenamento de dados através do Docker Compose foi um dos desafios superados. Decidir entre qual TSDB utilizar e qual ferramenta de comunicação entre as APIs e o Grafana demandou tempo e análise de desempenho, uma vez que a velocidade para disponibilidade do sistema era, também, um dos objetivos deste trabalho.

Arquitetar o arquivo **'docker-compose.yml'** para isolar e criar os serviços com uma determinada dependência; configurar as variáveis de ambientes; publicar (externalizar) portas de comunicação; mapear os volumes onde os dados dos contêineres seriam contidos; definir política de reinicialização; criar comandos para execução interna do contêiner; e associar todos os três serviços (influxdb, grafana, newman) em uma rede nomeada **qametrics** foram também alguns dos desafios superados.

Postman/Newman

O arquivo **qametrics.postman_collection.json** da coleção (*collection*) com as requisições às APIs e a configuração das variáveis de ambiente no arquivo **qametrics.postman_environment.json**, foram criados e elaborados dentro do Postman. Por questão de organização e priorização, as requisições da coleção foram agrupadas na seguinte ordem: **sonarcloud**, **jira**, **pyramid** e **influxDB**.

Para realizar requisições no **SonarCloud**, foi necessário apenas criar uma API Key e defini-la como Basic Authentication informando apenas o campo `Username` com a API Key. Quanto ao Jira, foi utilizado um JSON Web Tokens (JWT) gerado através de uma autenticação Basic Authentication informando o email no campo `Username` e a senha no campo `Password` das requisições.

Em relação à busca dos dados na planilha da Pirâmide de Teste, foi necessária a criação de um OAuth Access Token gerado através de uma requisição feita manualmente e unicamente antes de executar o **QA Metrics**. Esta foi uma das maiores dificuldades, tendo em vista que foi preciso entender a documentação da Google Cloud Platform sobre este tipo de autenticação no Postman.

Por último, as requisições realizadas ao TSDB **InfluxDB** foram feitas sem a necessidade de credenciais. Apesar dessa facilidade, o que causou um pouco de dificuldade foi entender como um

banco de dados temporal funciona e como manipular as métricas nele, tais como: inserir, selecionar e excluir. Não só isso, mas também solucionar os inúmeros erros encontrados durante o desenvolvimento deste projeto, a saber: valores nulos que não são aceitos e a preferência pelo uso de aspas simples nos valores e aspas duplas nas chaves e, também, questões de codificação de caracteres.

Grafana x InfluxDB

Finalmente, após instanciar o Docker Compose e subir os três serviços (influxdb, grafana, newman), foi possível carregar o Grafana no navegador, associar as consultas (*queries*) ao banco de dados e exibir as métricas nos gráficos dos dashboards. Essa foi uma das atividades mais complexas devido ao desconhecimento sobre a instalação, configuração e uso das ferramentas disponíveis no Grafana. Além disso, algumas limitações gráficas e de interface impediram que determinados tipos de gráficos fossem construídos dinamicamente tal como o da Pirâmide de Testes, por exemplo.

Test Reader

A configuração do Test Reader foi uma atividade sem muita dificuldade devido ao auxílio recebido do arquiteto de software. Infelizmente esta ferramenta não pôde ser integrada, de fato, ao QA Metrics. A integração facilitaria a verificação automática dos repositórios e inclusão do QA Metrics na pipeline. Entender como funciona o RegEx e utilizá-lo no Test Reader também foi uma árdua experiência durante a configuração dos atributos.

Para facilitar o trabalho dos QAs, criei um atalho (**alias**) no terminal para rodar as instruções do Test Reader (*script*) com apenas um comando, a saber: `pyramid`. Esse comando força a atualização da branch `master` de cada repositório, executa o Test Reader Discovery e exporta o resultado para a planilha que está definida nas variáveis de ambiente do computador local.

Google Cloud Platform

Desvendar o GCP foi uma ótima atividade porque agregou, em muito, o aprendizado sobre como a Google disponibiliza suas APIs e suas diferentes formas de autenticação. Ter de criar um projeto na Google Cloud Platform, gerar uma Service Account (Conta de Serviço), baixar as credenciais e utilizá-las no Test Reader e como parte de autenticação das requisições no Postman, foi de grande valia e conhecimento. Vale considerar também a geração do OAuth Access Token e das API Keys.

Jira

Ler a documentação e tentar executar o endpoint `/rest/api/3/search` foi um tanto frustante devido a mudanças realizadas na API, por exemplo: o endpoint supracitado passou a limitar a quantidade de bugs retornados na resposta da requisição até 50. No entanto, esta informação não consta na documentação e, caso o número de bugs seja além desse limite, é preciso utilizar paginação.

5 Impactos da sua formação no seu trabalho

Desde 2008, quando aprovado no processo seletivo para o curso Técnico em Informática disponibilizado gratuitamente pelo Serviço Nacional de Aprendizagem Industrial (SENAI) no estado de Pernambuco, notei que a área de tecnologia poderia me proporcionar uma gama de oportunidades no mercado de trabalho. Por isso, concluí o curso técnico que me possibilitou estagiar na Companhia Nacional de Abastecimento (CONAB) e ampliar minhas capacidades intelectual e profissional.

Três anos depois, em 2011, fui aprovado no processo seletivo para o curso Bacharelado em Sistemas de Informação disponibilizado gratuitamente pela Universidade Federal Rural de Pernambuco (UFRPE). O resultado ao cursar este curso não foi diferente, pois ele me capacitou muito mais para o mercado de trabalho. Ainda durante o curso, fui aprovado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) para realizar um intercâmbio via Ciência sem Fronteiras (CsF) para cursar 3 períodos acadêmicos na **Dalhousie University**[32] no Canadá durante 16 meses. Neste intercâmbio, pude aprimorar minha fluência na Língua Inglesa, ampliar o domínio de novas tecnologias e conhecer novas culturas em virtude da multiculturalidade presente nas províncias do Canadá.

No campo profissional, durante vínculo acadêmico com a UFRPE, inúmeras oportunidades de estágio e emprego surgiram em decorrência da experiência obtida na teoria e na prática sobre linguagens de programação, desenvolvimento de software, infraestrutura, bancos de dados e qualidade de software, além da relevância e do reconhecimento que a universidade tem diante do mercado de trabalho nacional. A Tabela 6 lista a linha do tempo da minha carreira enquanto vinculado à UFRPE.

Tabela 6: Histórico de carreira durante vínculo acadêmico

Empresa	**Função**	**Tipo de Contrato**	**Período**	**Local**
Grupo Líder Seguros	Analista de Sistemas	Estágio	Abr/2013 a Ago/2013	Brasil
SolutionInc	Analista de Testes (QA)	Estágio	Mai/2014 a Set/2014	Canadá
Qualinfo Tecnologia	Analista de Suporte	Estágio	Set/2015 a Mai/2016	Brasil
Qualinfo Tecnologia	Analista de Testes (QA)	Estágio	Jun/2016 a Ago/2017	Brasil
Qualinfo Tecnologia	Analista de Testes Júnior (QA)	CLT	Set/2017 a Mai/2019	Brasil
Stefanini	Engenheiro de Testes Júnior (QA)	CLT	Mai/2019 a Out/2020	Brasil
CI&T	Engenheiro de Testes Pleno (QA)	CLT	Out/2020 a Dez/2020	Brasil
CI&T	Engenheiro de Testes Sênior (QA)	CLT	Jan/2021 a Mai/2022	Brasil
Questrade	Engenheiro de Testes Sênior (QA)	CLT	Desde Mai/2022	Brasil

Fonte: Elaborado pelo autor

Acredito que, com todo esse *background* em **bancos de dados** (SQL & NoSQL), **automação de testes de sistemas** (Web, Mobile e APIs), **criação de relatórios de qualidade**, **fluência na Língua Inglesa**, entre outros, possa ter sido crucial e determinante para o desenvolvimento do QA Metrics. Pois, parte das tecnologias utilizadas neste projeto já faziam parte do meu *know-how*, o que facilitou ainda mais sobre quais tecnologias poderiam ser adequadas e de fácil suporte perante as comunidades *open source* existentes para cada uma das ferramentas utilizadas neste trabalho.

[32] https://www.dal.ca

6 Conclusão

O QA Metrics auxilia a integração de métricas de qualidade de software, permitindo assim reduzir tarefas repetitivas realizadas pelos QAs, além de possibilitar que qualquer membro da squad do cliente da empresa CI&T tenha acesso, mesmo na ausência de um QA, a todas estas métricas disponibilizadas na forma de gráficos, tabelas e estatísticas as quais são exibidas em dashboards.

O projeto de desenvolvimento de um sistema integrador de métricas e indicadores de qualidade, como é o caso do QA Metrics, requer um profundo conhecimento de ferramentas, tais como as apresentadas neste trabalho, que possibilitem a implementação de um sistema portátil e executável em qualquer sistema operacional no qual os dados são exibidos interativamente e acessível via web.

A partir desse projeto foi possível garantir que todos os membros da squad pudessem coletar as métricas em questão de segundos. O tempo para execução e disponibilização do QA Metrics (incluindo coleta e armazenamento) varia, normalmente, entre 30 e 60 segundos. A economia de tempo total no processo de criação da retrospectiva foi reduzida drasticamente, como visto, de 10h (aproximadamente) para 4 minutos. Ou seja, para cada QA a média para execução foi de até 1 minuto, equivalente a 99,45% de economia do recurso mais importante de um colaborador: **tempo**.

Em linhas gerais, os QAs foram os mais beneficiados com esse projeto, uma vez que não será mais necessário realizar a coleta de métricas manualmente nem ter que armazená-las nas planilhas de qualidade. Contudo, a empresa, o cliente e todos os membros da squad, por tabela, foram favorecidos pelo fato de não dependerem unicamente de um QA quando este estiver ausente no ambiente de trabalho por quaisquer motivos.

Referências Bibliográficas

[1] Thiago Coutinho, “Saiba como funciona o desenvolvimento de software e suas etapas,” https://www.voitto.com.br/blog/artigo/o-que-e-desenvolvimento-de-software, 2020, (Acesso em: 25/05/2022).

[2] Espresso Labs, “Como usar métricas para garantir a qualidade de software,” https://espressolabs.com.br/2021/04/07/como-usar-metricas-para-garantir-a-qualidade-de-software, 2021, (Acesso em: 08/07/2022).

[3] I. Sommerville, *Engenharia de Software*. Editora Pearson Education, 2011, no. 9.

[4] Katia Simões, “Como a CI&T, criada por três amigos, tornou-se a única brasileira no ranking da revista Fortune de empresas de tecnologia,” http://revistaepoca.globo.com/Revista/Epoca/0,,EDR78613-8056,00.html, 2007, (Acesso em: 26/05/2022).

[5] Tomás Duarte, “Net Promoter Score: entenda o que é o NPS e como implementar esta métrica na sua empresa!” https://track.co/blog/net-promoter-score, 2021, (Acesso em: 26/05/2022).

[6] Juliana Américo, “Em pleno crescimento: veja como é trabalhar na CI&T,” https://vocesa.abril.com.br/carreira/em-pleno-crescimento-veja-como-e-trabalhar-na-cit, 2021, (Acesso em: 26/05/2022).

[7] Maurício Renner, “CI&T fatura R$ 1,4 bilhão, alta de 51%,” https://www.baguete.com.br/noticias/14/03/2022/cit-fatura-r-14-bilhao-alta-de-51, 2022, (Acesso em: 26/05/2022).

[8] Marylene Guedes, “No final das contas: o que é o Docker e como ele funciona?” https://www.treinaweb.com.br/blog/no-final-das-contas-o-que-e-o-docker-e-como-ele-funciona, 2018, (Acesso em: 15/06/2022).

[9] StackOverflow, “Developer Survey 2022,” https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-tools-tech-want, 2022, (Acesso em: 06/07/2022).

[10] HG Insights, “Companies Currently Using Docker,” https://discovery.hgdata.com/product/docker, (Acesso em: 15/06/2022).

[11] Cui Yunlong, “What Is Docker Container?” https://info.support.huawei.com/info-finder/encyclopedia/en/Docker+Container.html, 2021, (Acesso em: 16/06/2022).

[12] Amanda Quaglio, “test-reader (Github Repository),” https://github.com/amandaquaglio/test-reader, (Acesso em: 30/06/2022).

[13] Linux IAC, “What is Docker Container,” https://linuxiac.com/what-is-docker-container, (Acesso em: 16/06/2022).

[14] Rafael Gomes, “Docker para Desenvolvedores,” https://stack.desenvolvedor.expert/appendix/docker/compose.html, 2016, (Acesso em: 22/06/2022).

[15] Cristian Trucco, “Docker Compose: O que é? Para que serve? O que come?” https://imasters.com.br/banco-de-dados/docker-compose-o-que-e-para-que-serve-o-que-come, 2018, (Acesso em: 22/06/2022).

[16] Prisma, “Integration testing,” https://www.prisma.io/docs/guides/testing/integration-testing, (Acesso em: 22/06/2022).

[17] João Pedro Bitencourt, “InfluxDB: Opção de banco de dados para um alto volume de consultas e escritas,” https://serverdo.in/influxdb, 2020, (Acesso em: 15/06/2022).

[18] HG Insights, “Companies Currently Using InfluxDB,” https://discovery.hgdata.com/product/influxdb, (Acesso em: 15/06/2022).

[19] Josh Powers, “Getting Started with Telegraf,” https://www.influxdata.com/blog/getting-started-with-telegraf, (Acesso em: 30/06/2022).

[20] InfluxData, “InfluxDB Line Protocol Tutorial,” https://docs.influxdata.com/influxdb/v1.8/write_protocols/line_protocol_tutorial, (Acesso em: 30/06/2022).

[21] Pedro César Tebaldi, “O que é e como funciona o grafana? Entenda aqui!” https://www.opservices.com.br/grafana, 2019, (Acesso em: 23/06/2022).

[22] HG Insights, “Companies Currently Using Grafana,” https://discovery.hgdata.com/product/grafana, (Acesso em: 23/06/2022).

[23] Rajat Gupta, “API Testing Using Postman and Newman,” https://www.velotio.com/engineering-blog/api-testing-postman-newman, 2022, (Acesso em: 23/06/2022).

[24] HG Insights, “Companies Currently Using Postman,” https://discovery.hgdata.com/product/postman, (Acesso em: 23/06/2022).

[25] Tanmay Deshpande, “Automate Your API Testing With CI Pipelines,” https://betterprogramming.pub/automate-your-api-testing-with-ci-pipelines-ee6b8d133114, (Acesso em: 23/06/2022).

[26] Software Testing Help, “How To Use Command Line Integration With Newman In Postman?” https://www.softwaretestinghelp.com/postman-newman, (Acesso em: 23/06/2022).

[27] HG Insights, “Companies Currently Using JIRA,” https://discovery.hgdata.com/product/atlassian-jira-software, (Acesso em: 30/06/2022).

[28] Atlassian, “Atlassian Developer - API Reference,” https://developer.atlassian.com/cloud/jira/platform/rest/v3/api-group-issue-search/#api-rest-api-3-search-post, (Acesso em: 30/06/2022).

[29] Jhony Walker, “Azure DevOps e SonarCloud - em nosso pipeline,” https://dev.to/jhonywalkeer/azure-devops-e-sonarcloud-em-nosso-pipeline-2nmc, 2021, (Acesso em: 30/06/2022).

[30] Israel Lucania, “O que é o SonarQube?” https://konia.com.br/o-que-e-o-sonarqube, 2020, (Acesso em: 30/06/2022).

[31] HG Insights, “Companies Currently Using SonarQube,” https://discovery.hgdata.com/product/sonarqube, (Acesso em: 30/06/2022).

[32] Milan Milanović, “Enabling High-Quality Code in .NET,” https://milan.milanovic.org/post/enabling-high-code-quality-in-net, (Acesso em: 30/06/2022).

[33] Thays Sanchez, “Google Cloud Platform: saiba o que é e quais seus benefícios!” https://santodigital.com.br/google-cloud-platform-saiba-o-que-e-e-quais-seus-beneficios, 2018, (Acesso em: 30/06/2022).

[34] HG Insights, “Companies Currently Using Google Cloud Platform,” https://discovery.hgdata.com/product/google-cloud-platform, (Acesso em: 30/06/2022).

[35] Bobby Hellard, “Google Cloud lost $3.1 billion in 2021,” https://www.itpro.co.uk/cloud/cloud-computing/362121/google-cloud-reports-31bn-annual-losses, 2022, (Acesso em: 30/06/2022).

[36] Google Identity, “Using OAuth 2.0 to Access Google APIs,” https://developers.google.com/identity/protocols/oauth2, (Acesso em: 30/06/2022).

7 Apêndice

QA Metrics

O código do QA Metrics encontra-se disponível publicamente no repositório do Github sob o endereço: github.com/lflucasferreira/qametrics. No local citado existe um arquivo **README.md** com todas as informações necessárias para configuração e uso do projeto desenvolvido. Espera-se que a pessoa interessada saiba utilizar o Git ou baixar manualmente a última versão do repositório.

Glossário

Action Item é um evento, tarefa, atividade ou ação documentada que precisa ocorrer. Os action items (itens de ação) são unidades discretas que podem ser manipuladas por uma única pessoa.

Agile é um abordagem de desenvolvimento de software centrada nas necessidades do cliente. Seu foco é atender bem, usando como pilares fases curtas e dinâmicas de desenvolvimento e, principalmente, a flexibilidade para lidar com mudanças no projeto.

API é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem se envolver em detalhes da implementação do software, mas apenas usar seus serviços.

API Key é uma chave criptografada que dá acesso total a todas as operações que uma API pode realizar, incluindo escrever novos dados ou excluir dados existentes.

Background é a experiência, ou habilidades, obtida nos campos intelectual, profissional e educacional, necessárias em uma determinada área ou segmento.

Backlog é uma pilha de pedidos em espera, ou melhor, corresponde a um registro ou histórico de requisições que são demandas do próprio cliente, embora também possam ser internas.

Basic Authentication é o sistema de autenticação mais comum do protocolo HTTP que é incluído no cabeçalho da requisição HTTP.

Branch é um ramo, ou cópia, das linhas de código de um software gerenciada por um sistema de controle de versão (VCS) tal como o Git.

Bug é um defeito, uma falha ou um erro no código de um programa que provoca seu mau funcionamento.

Casos de Teste é um conjunto de condições usadas para teste de software, no qual se deve especificar os valores de entrada e os resultados esperados do processamento.

Chairman é o Presidente do Conselho de Administração de uma empresa, ou seja, o mais alto representante de um grupo empresarial ou empresa individual.

Cloud Computing é uma tecnologia que permite o uso remoto de recursos da computação por meio da conectividade da Internet..

Code Smell é qualquer característica no código-fonte de um programa que possivelmente indica um problema mais profundo.

Dashboard é um painel visual que apresenta, de maneira centralizada, um conjunto informações: indicadores e suas métricas. Essas informações podem ser tanto indicadores da área de TI como de gestão empresarial.

DevOps (junção das palavras “desenvolvimento” e “operação”) é um modelo que combina filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma empresa distribuir seus serviços em alta velocidade.

Docker Engine é uma tecnologia de conteinerização de código aberto para construir e conteinerizar seus aplicativos.

Endpoint é a URL onde um serviço de um servidor web pode ser acessado por uma aplicação cliente, ou seja, é a localização de onde APIs podem acessar um recurso.

Framework é uma espécie de modelo que, quando utilizado, oferece certos artifícios e elementos estruturais básicos para a criação de alguma aplicação ou software.

Git é um sistema para controle de versão capaz de criar todo o histórico de alterações em um código de software e facilmente voltar para qualquer ponto dele para saber como o código estava numa determinada versão.

Grafana é uma plataforma para visualizar e analisar métricas por meio de gráficos em dashboards.

Hyperlink é uma referência dentro de um documento em hipertexto a outras partes desse documento ou a outro documento.

Java é uma linguagem de programação orientada a objetos desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems.

Jenkins é um servidor de automação de código aberto que gerencia e controla os processos de entrega de software em todo o ciclo de vida, incluindo desenvolvimento, documentação, teste, implantação e análise de código estático.

Jira é uma ferramenta que permite o monitoramento de tarefas e acompanhamento de projetos garantindo o gerenciamento de todas as suas atividades em único lugar.

Kanban é um sistema de gestão visual para controle de tarefas e fluxos de trabalho através da utilização de colunas e cartões, facilitando a gestão de atividades.

Kernel é o componente central do sistema operacional da maioria dos computadores. Ele serve de ponte entre aplicativos e o processamento real de dados feito à nível de hardware. As responsabilidades do núcleo incluem gerenciar os recursos do sistema.

Know-How é uma habilidade adquirida pela experiência podendo ser prática ou teórica.

Lean é uma filosofia de gestão inspirada em práticas e resultados do Sistema Toyota que utiliza alguns princípios e técnicas operacionais buscando sempre reduzir o desperdício de recursos.

Microsoft SQL Server é um sistema gerenciador de banco de dados relacional desenvolvido pela empresa Microsoft.

MySQL é um sistema gerenciador de banco de dados relacional desenvolvido como projeto de código aberto.

Máquina Virtual é um software de ambiente computacional, que executa programas como um computador real, também chamado de processo de virtualização.

Nearshore é um tipo de subcontratação ou terceirização de uma atividade com salários mais baixos que no próprio país, que se encontra relativamente perto na distância ou a zona horária (ou ambos).

OAuth Access Token é um protocolo de autorização que permite que uma aplicação se autentique em outra. contém as credenciais de segurança para uma sessão de login e identifica o usuário, os grupos do usuário, os privilégios do usuário e, em alguns casos, um aplicativo específico.

Open Source é um software de código aberto, ou seja, com o seu código fonte disponibilizado e licenciado com uma licença de código aberto no qual o direito autoral fornece o direito de estudar, modificar e distribuir o software de graça para qualquer um e para qualquer finalidade.

Pipeline é uma cadeia de etapas que são realizadas até que a versão final de um software seja disponibilizado.

Pirâmide de Teste é uma forma gráfica de demonstrar de forma simples os tipos de testes, seus níveis, velocidade de implementação e complexidade dos testes realizados.

Pod é um pequeno time que é responsável por uma tarefa, requerimento ou parte do backlog. Uma Pod é customizável e pode mudar dependendo do andamento dos requisitos.

PostgreSQL é um sistema gerenciador de banco de dados relacional desenvolvido como projeto de código aberto.

Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.

Qualidade de Software é uma área de conhecimento em engenharia de software e seu objetivo é garantir a qualidade do software através da definição e normatização de processos de desenvolvimento.

Ranking é uma posição/classificação que algo ou alguém ocupa numa escala ordenada que destaca seu mérito em relação aos demais.

Retrospectiva é um rito de avaliação do sprint que acabou de se encerrar. Nessa reunião, o time Scrum avalia o que foi bom e o que deve ser melhorado a fim de traçar planos de ações em busca da melhoria contínua do processo.

Roadmap é uma espécie de mapa, uma poderosa ferramenta visual e descritiva que apontará como será o produto ou projeto a cada período de sua evolução.

Runtime é um termo geral que se refere a qualquer biblioteca, estrutura ou plataforma na qual seu código é executado.

Scrum é um framework de gerenciamento de projetos (conjunto de técnicas/processos de gerenciamento não linear de projetos em equipe), da etapa da organização ao desenvolvimento ágil de produtos complexos e adaptativos buscando o valor máximo, criado na década de 1990.

Scrum Master é o responsável por garantir que o Time Scrum se oriente pelos valores e práticas do Scrum, bem como proteger o time, certificando-se de que os membros não se comprometam com compromissos além dos que eles conseguem cumprir dentro de uma Sprint.

Software é uma coleção de dados ou instruções que informam a um mecanismo como trabalhar. Em síntese, é um programa que você acessa no celular, tablet, computador ou qualquer outro dispositivo eletrônico.

SonarQube é uma ferramenta responsável por garantir a qualidade de software através do processo de inspeção continua do código de um projeto realizando uma “varredura” em seu código-fonte e analisando possíveis vulnerabilidades, erros e regras específicas da linguagem (code smells).

Sprint é o nome de um ciclo de desenvolvimento do Scrum. Os sprints podem ter a duração de 2 a 4 semanas, sendo esse o time box (duração) do ciclo e todos os sprints de um projeto devem ter a mesma duração.

Squad é um time de trabalho composto por pessoas de diferentes áreas e que, ao mesmo tempo, possui ações mais focadas, colaborativas e ágeis.

Stack é o conjunto de sistemas necessários para executar um único aplicativo sem outro software adicional.

Stakeholder é a pessoa e a organização que pode ser afetada por um projeto ou empresa, de forma direta ou indireta, positiva ou negativamente. O stakeholder faz parte da base da gestão de comunicação e é importante para o planejamento e execução de um projeto.

Tech Stack é um conjunto de tecnologias usado para criar e executar um aplicativo ou projeto. Normalmente, consiste em linguagens de programação, frameworks, banco de dados, ferramentas de front-end e back-end e aplicativos conectados por meio de APIs.

Testador de Software é o profissional responsável por disseminar a cultura de qualidade, encontrar erros, falhas, bugs e outros tipos de problemas que não foram detectados durante o desenvolvimento de um software.

Timestamp é uma cadeia de caracteres denotando a hora ou data que certo evento ocorreu.

Upgrade é a atualização dos componentes do hardware ou do software de um computador/servidor.