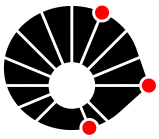

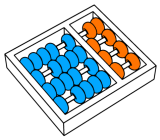

Análise de repositórios de software em projetos baseados na arquitetura Serverless

Davi Vinícius Cunha *Breno Bernard Nicolau de França*

Relatório Técnico - IC-PFG-21-37
Projeto Final de Graduação
2021 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Análise de repositórios de software em projetos baseados na arquitetura Serverless

Davi Vinícius Cunha* Breno Bernard Nicolau de França*

Resumo

Computação *Serverless* é um assunto atualmente em alta no mercado, relacionado à Computação em Nuvem, devido ao seu potencial de eliminação de preocupações com infraestrutura e escalabilidade de soluções por parte da equipe de desenvolvimento. A literatura acadêmica sobre o tema está em estágio inicial no que diz respeito às boas e más práticas de uso comuns em aplicações Serverless, sobretudo em uma perspectiva de manutenção e evolução de software. Diante desse cenário, o objetivo deste trabalho é, através da mineração de repositórios de software de código aberto, desenvolver uma ferramenta que analise e colete métricas relacionadas às práticas comumente presentes em projetos desenvolvidos para execução Serverless, de modo que questionamentos sejam levantados e sejam definidas direções que sirvam de motivação e base para estudos mais aprofundados no tema.

Para isso, foi desenvolvida uma estratégia capaz de buscar, avaliar a pertinência e coletar métricas de manutenibilidade e escala dos projetos, e por fim os resultados passaram por uma análise inicial que permitiu identificar questões, padrões e suspeitas de possíveis antipadrões e práticas comuns para que os estudos na área evoluam, além de limitações e possibilidades de evolução da ferramenta de apoio serem abordadas para desenvolvimento futuro.

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP

1 Introdução

Serverless Computing tem a proposta de ser uma abordagem eficiente e prática para se mitigar a necessidade de gestão de infraestrutura durante o processo de desenvolvimento de software. Nesse sentido, as plataformas *Serverless* oferecidas pelos provedores de serviços em nuvem atuais incorporam toda a responsabilidade de controle da escalabilidade, provisionamento de hardware e manutenção de infraestrutura, permitindo que desenvolvedores mantenham o foco estritamente no desenvolvimento das soluções.

Apesar de oferecer tais benefícios, também há algumas limitações em relação à flexibilidade das plataformas *Serverless*. É necessário que os artefatos desenvolvidos (incluindo código-fonte e configurações) sigam padrões estritos de desenvolvimento para que sejam compatíveis com as plataformas, onde cada provedor possui um padrão próprio a ser seguido, o que pode acabar acoplando o código desenvolvido ao provedor de serviços na nuvem escolhido, como observado em [1]. Com isso, há especificidades a serem consideradas quando se analisa a estrutura de projetos *Serverless* e quais práticas são comuns nesses projetos, especialmente devido ao fato de que a escolha do provedor de serviços influenciará a estruturação do projeto.

Ao nos aprofundarmos na literatura sobre *Serverless Computing*, é possível observar-se que questões relacionadas com boas e más práticas, organização de projetos, manutenibilidade e dívida técnica estão em um estágio inicial [2], com diferentes abordagens sendo propostas. Entretanto, essas propostas eventualmente acabam até mesmo conflitando entre si [3]. Ainda não há consenso e, mais que isso, ainda é difícil de compreender como *Serverless* é aplicado atualmente por desenvolvedores e se a tecnologia pode estar sendo adotada "cegamente" devido aos seus benefícios prometidos. Tal situação poderia acarretar no uso em cenários que não são pertinentes à tecnologia, gerando problemas de manutenção e desempenho no longo prazo. Já existem algumas ideias do que seriam práticas comuns nesse contextos e quais consequências algumas delas podem trazer [3] [4], logo também é de interesse a análise dessas práticas para que se observe se as mesmas são de fato comuns em aplicações reais disponíveis em repositórios de código aberto.

Na ausência de definições concretas e empiricamente avaliadas de padrões e antipadrões na literatura, não é viável que uma análise seja conduzida buscando identificar os mesmos. Porém, pensando na carência dessas informações e na possibilidade de construir uma base de análise de práticas utilizadas atualmente que seja capaz de direcionar e acelerar esse entendimento do uso de *Serverless*, este trabalho propõe o desenvolvimento de uma ferramenta de análise estática de código que seja capaz de acessar repositórios de software e extrair uma suíte de métricas relacionadas a padrões de desenvolvimento que podem ser analisadas e utilizadas para comparação entre diferentes projetos.

Considerando a grande quantidade de repositórios disponíveis, a plataforma GitHub foi escolhida para que repositórios disponíveis na mesma sejam utilizados e uma lista selecionada de repositórios foi utilizada para aplicação da solução e análise em um contexto menor. Ainda que em menor escala, o racional da análise e a ferramenta desenvolvida foram preparados para aplicação de forma genérica em outros projetos e em maior escala, servindo como uma contribuição inicial no desenvolvimento da literatura e análises no tema.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta a

fundamentação teórica e conceitos relevantes para o entendimento do trabalho. A Seção 3 descreve o método e abordagem utilizada para definição da melhor solução. Em seguida, na Seção 4, a solução é descrita conceitualmente e é detalhada sua implementação prática. Na Seção 5, os resultados da solução e análise são discutidos e, na Seção 6, detalha-se a avaliação da solução proposta, dificuldades encontradas, conclusões do trabalho e direções para iterações futuras.

2 Fundamentação Teórica

2.1 Computação em Nuvem e Provedores de Serviços na Nuvem

A computação em nuvem [7] se trata de um termo que define a disponibilização de recursos compartilhados sobre demanda de forma virtual para utilização de terceiros, em forma de serviço. Mais especificamente, trata-se do oferecimento de infraestrutura e serviços que aceleram e facilitam o desenvolvimento e a disponibilização de software, de forma que os usuários não necessitem da implantação e investimento em infraestrutura física, podendo contratar provedores que forneçam as mesmas via uma estrutura de rede personalizada. Um grande diferencial da computação em nuvem é a tendência ao modelo de cobrança sob demanda, onde usuários pagam apenas pelos recursos utilizados, sem preços fixos.

Os provedores de serviços na nuvem são então companhias que disponibilizam uma grande quantidade de recursos físicos e capacidade computacional para contratação externa de maneira compartilhada, buscando otimizar o uso e distribuição de seus recursos para que a maior capacidade possível seja utilizada. Entre eles estão Google, Amazon, Microsoft, entre outros. O grande desafio dos provedores é que este modelo de negócios os torna responsáveis pela disponibilidade, manutenção e escalabilidade dos recursos de infraestrutura, que influencia a maneira como realizam a cobrança e oferecimento de cada serviço e recurso personalizado.

Com a evolução da computação em nuvem e o aumento de sua adoção, suas capacidades foram se expandindo, de forma que atualmente é possível ter datacenters e recursos híbridos, compartilhados entre infraestrutura física e serviços contratados de provedores de serviços em nuvem. Ainda mais, os serviços disponibilizados têm se tornado cada vez mais complexos e granulares, buscando diminuir cada vez mais a necessidade de esforço de configuração e *setup*, de forma a oferecer maior velocidade de desenvolvimento e simplicidade de disponibilização de recursos. Um exemplo interessante, e que ilustra bem essa dinâmica, são plataformas de computação *Serverless*, melhor detalhadas na subseção 2.2.

2.2 Computaçao Serverless

Computação *Serverless* [6], ou computação "sem servidor" (em tradução literal), é um formato de oferecimento de recursos de infraestrutura de forma altamente abstraída, onde o foco do desenvolvimento fica totalmente no código e em configurações simples de escalabilidade e bibliotecas externas, sem qualquer trabalho adicional de disponibilização ou manutenção de infraestrutura. Esse formato de oferecimento costuma contar com controle de escalabilidade automático e dinâmico e a distribuição de carga automática, tendo também algumas limitações como obrigatoriedade do cumprimento de padrões de assinatura de função e organização de arquivos do projeto e uma tendência a segregação de código em pedaços pequenos, de execução rápida e leve. Nesse contexto, surge o termo Funções *Serverless*, que é uma nomenclatura comum utilizada nos serviços fornecidos pelos provedores de serviço na nuvem no modelo *Serverless*.

Funções *serverless* nada mais são do que a aplicação prática desse modelo de computação, de forma que os provedores direcionam o uso de seus serviços *Serverless* no formato de funções, sendo esse o padrão de desenvolvimento definido para que o serviço possa

ser utilizado. Com isso, a tendência em um primeiro momento é que os projetos desenvolvidos utilizando plataformas *serverless* sejam funções pequenas e leves, de execução rápida. As funções *serverless* costumam ter a característica de execução sobre demanda, ou seja, além de terem sua quantidade de réplicas e recursos de hardware modificados e alterados automaticamente em tempo real conforme a carga, sua execução e disponibilização de infraestrutura tende a ser feita por demanda, onde caso nenhuma função seja executada, nenhuma infraestrutura estará ativa, reduzindo o custo de computação na nuvem. Nesse sentido, os serviços *Serverless* oferecidos costumam limitar o tempo de execução de funções e invocações e gerar custos maiores conforme sua utilização se torna longa e próxima de serviços normais de execução contínua.

2.3 Padrões de Projeto e Antipadrões

Padrões de projeto são soluções gerais para problemas comuns em projetos de software, geralmente propostas como modelos ou padrões gerais que podem ser adaptados para contextos semelhantes durante o desenvolvimento [8]. De forma similar, antipadrões [9] são utilizações de designs de projeto e práticas em contextos que não são coerentes com seu propósito, podendo trazer resultados contrários e influenciar negativamente a qualidade do código e projeto desenvolvidos, dificultando a evolução e aplicação do software em questão.

2.4 Mineração de Repositórios de Software

A mineração de repositórios de software [10] se trata de um processo definido para se obter métricas, dados e evidências iniciais em maior escala a partir de repositórios de software disponíveis. Seu propósito é a obtenção de informações de forma automatizada a partir de repositórios de software para se realizar estudos preliminares.

Sua aplicação não se limita somente à análise de código, englobando até mesmo a análise de padrões de processo de desenvolvimento e commits, frequência de atividade, documentação, issues, bugs, entre outros dados de alta relevância que estão presentes em repositórios de versionamento de software online.

2.5 Análise Estática de Código e Expressões Regulares

A análise automatizada de código-fonte é um conceito chave para a solução proposta no trabalho. Trata-se de uma análise estática (sem a execução do software) realizada automaticamente sobre o código de um projeto, sendo utilizada com diversos objetivos no contexto de software. Um exemplo de suas aplicações são as ferramentas de análise do tipo *Linter*, que são utilizadas para analisar e reforçar padrões de qualidade no código escrito. No contexto deste trabalho, a análise estática será utilizada para gerar uma série de métricas a partir de padrões e elementos encontrados diretamente no código.

O formalismo utilizado para extrair as métricas do código dos projetos selecionados também utiliza de Expressões Regulares, que são formas definidas de se identificar cadeias de caracteres de acordo com um padrão esperado. As expressões regulares seguem uma especificação formal a ser seguida ao serem criadas, e que então são interpretadas para se buscar conteúdo textual.

3 Método de Trabalho

Com o objetivo do trabalho sendo o desenvolvimento de uma ferramenta para extração de métricas relacionadas aos padrões e práticas recorrentes em projetos *Serverless* de código aberto, os seguintes passos foram seguidos durante o desenvolvimento:

- Revisão bibliográfica da literatura para identificação do estado da arte relacionado às práticas e padrões utilizados em projetos Serverless;
- Proposição de um método a ser utilizado na plataforma GitHub para buscar por repositórios de interesse e excluir possíveis falsos-positivos;
- Definição de um grupo reduzido e curado de repositórios de interesse para aplicação da ferramenta e análise dos resultados, de forma a ilustrar sua utilização e facilite a aplicação generalizada em escala maior;
- Definição das métricas a serem extraídas dos artefatos dos projetos para análise;
- Desenvolvimento da ferramenta para extração das métricas a partir dos artefatos dos repositórios;
- Análise e consolidação das métricas extraídas, resultando em evidências iniciais e validação dos dados obtidos.

4 Resultados

4.1 Revisão da Literatura

4.1.1 Processo de Revisão

A revisão da literatura foi realizada através de buscas nas bibliotecas digitais *Scopus* e *IEEEXplore,* em especial por incluírem os estudos mais relevantes sobre Arquitetura de Software e Engenharia de Software. Para a realização das buscas, restringidas à área de Ciência da Computação, a seguinte estratégia de combinação de palavras-chave foi utilizada:

```
(
"serverless"
OR "function as a service"
OR "function-as-a-service"
OR "FaaS"
)
AND
(
"smell"
OR "issue"
OR "issues"
OR "problem"
OR "anti-pattern*"
OR "anti pattern*"
OR "maintenance"
OR "technical debt"
OR "design debt"
OR "architecture debt"
OR "architectural debt"
OR "pitfall*"
OR "bad practice*"
)
```

Com os resultados da busca, foram selecionados artigos que apresentaram maior relevância e discussões diretamente relacionadas com o tema de práticas em projetos serverless, e com isso foram obtidas as publicações de interesse [2] [3] [4] [5] [11] [12] [13] [14], que então seguiram para aprofundamento e revisão.

4.1.2 Conclusão da Revisão

Após a condução da revisão da literatura, foi possível identificar que a literatura em questão de *Serverless* está em um momento inicial, com algumas propostas de uso e organização de código, mas ainda sem muitos experimentos e avaliações práticas. A revisão foi fundamental para verificar a carência deste conteúdo e o valor que uma análise inicial das práticas utilizadas pode trazer no direcionamento e fomento dos estudos na questão de práticas e padrões de utilização de soluções *Serverless*.

4.2 Definição de métricas a serem extraídas dos repositórios

Para a definição das métricas a serem extraídas, foi realizada uma análise dos principais pontos de relevância que podem ser extraídos através da análise estática de código e que podem servir como ponto de partida para a evolução do estudo de práticas e padrões no cenário. Os principais aspectos de relevância a serem analisados foram Escalabilidade e Manutenibilidade de código. Foram também propostas informações gerais como a identificação do provedor Cloud baseado nos padrões de código exigidos por cada um, que podem ser úteis para a análise futura. Com isso, as métricas foram definidas avaliando valores máximos, mínimos e médios conforme a Tabela 1.

Identificador	**Característica**	**Métrica**
G1	Geral	Profundidade máxima de pastas
G2	Geral	Provedor Cloud
E1	Escala	Quantidade total de arquivos
E2	Escala	Quantidade de arquivos de código
E3	Escala	Quantidade de funções
E4	Escala	Quantidade de linhas de código
M1	Manutenibilidade	Quantidade de bibliotecas externas
M2	Manutenibilidade	Quantidade de pastas
M3	Manutenibilidade	Quantidade de comentários
M4	Manutenibilidade	Quantidade de arquivos de configuração
MXM5	Manutenibilidade	Quantidade máxima de arquivos de código por pasta
MDM5	Manutenibilidade	Quantidade média de arquivos de código por pasta
MNM5	Manutenibilidade	Quantidade mínima de arquivos de código por pasta
MXM6	Manutenibilidade	Tamanho máximo de nome de funções
MDM6	Manutenibilidade	Tamanho médio de nome de funções
MNM6	Manutenibilidade	Tamanho mínimo de nome de funções
MXM7	Manutenibilidade	Quantidade máxima de parâmetros em funções
MDM7	Manutenibilidade	Quantidade média de parâmetros em funções
MNM7	Manutenibilidade	Quantidade mínima de parâmetros em funções
MXM8	Manutenibilidade	Quantidade máxima de linhas em arquivos de código
MDM8	Manutenibilidade	Quantidade média de linhas em arquivos de código
MNM8	Manutenibilidade	Quantidade mínima de linhas em arquivos de código
MXM9	Manutenibilidade	Quantidade máxima de linhas em arquivos de configuração

Identificador	Característica	Métrica
MDM9	Manutenibilidade	Quantidade média de linhas em arquivos de configuração
MNM9	Manutenibilidade	Quantidade mínima de linhas em arquivos de configuração
MXM10	Manutenibilidade	Quantidade máxima de funções por arquivo de código
MDM10	Manutenibilidade	Quantidade média de funções por arquivo de código
MNM10	Manutenibilidade	Quantidade mínima de funções por arquivo de código
MXM11	Manutenibilidade	Quantidade máxima de comentários por arquivo
MDM11	Manutenibilidade	Quantidade média de comentários por arquivo
MNM11	Manutenibilidade	Quantidade mínima de comentários por arquivo

Tabela 1. Definição das métricas que serão extraídas dos repositórios

4.3 Desenvolvimento da ferramenta para obtenção das métricas

A escolha do método de análise estática foi uma decisão importante pois com ela não somente pode-se analisar e obter métricas da organização do código, como também podemos analisar a estrutura do repositório, no que diz respeito às pastas, arquivos de configuração, entre outros. Essas informações são também importantes para maior conhecimento da estrutura desses tipos de projetos, dado que a proposta inicial dos serviços e provedores passa a impressão de que o foco dos projetos são funções pequenas e autocontidas.

A plataforma GitHub foi escolhida para seleção dos repositórios por ser uma plataforma de versionamento de software amplamente utilizada e que possui fácil acesso aos códigos e mecanismos de filtro de forma automatizada, também com uma clara documentação desses meios de acesso. Com isso, foi desenvolvido o método a ser seguido pela ferramenta e análise, conforme ilustrado na Figura 1.

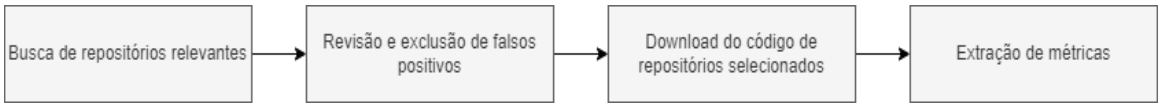

Figura1. Passos executados pela ferramenta desenvolvida.

Com a visão macro definida para a ferramenta, foi iniciado o trabalho de detalhar as etapas no que diz respeito à definição dos filtros para os repositórios, definição de regras para exclusão de falso-positivos, criação de uma estratégia para download e acesso ao código do repositório e extensão da estratégia para extração das métricas.

4.3.1 Tecnologias Adotadas

A ferramenta foi desenvolvida utilizando a linguagem de programação Python com execução via Jupyter Notebook por sua flexibilidade e praticidade de manipulação de métricas. O ambiente proposto é o da execução em um container Docker, de forma que toda a infraestrutura fique abstraída e possa ser executada virtualmente em qualquer ambiente. A definição

Docker da ferramenta, as bibliotecas necessárias e seu código-fonte estão disponíveis em: *https://github.com/inoridv/pfg*.

Para sua execução, basta que um computador com capacidade de execução de contêineres Docker construa a imagem proposta e a execute.

4.3.2 Meio de busca de repositórios

Os repositórios de interesse são projetos que foram desenvolvidos para serem executados em ambientes *serverless*, não sendo eles bibliotecas ou extensões que facilitam o desenvolvimento nesses ambientes, nem sendo projetos de exemplo.

O filtro para busca dos repositórios foi desenvolvido utilizando de algumas palavras-chave para filtro dos projetos na busca automatizada da ferramenta GitHub, considerando também um filtro de linguagem limitando as buscas às linguagens Python e Javascript, que são atualmente as mais amplamente utilizadas no contexto de projetos Serverless [15]. As combinações propostas para os filtros são apresentadas abaixo:

Palavras-chave

- serverless
- function as a service
- FaaS

Linguagens

- Python
- Javascript

Com isso, 6 buscas são executadas no total, sendo cada uma a combinação de uma palavra-chave com uma linguagem simultaneamente.

As buscas foram divididas entre as diferentes linguagens de programação pelo fato de suas características de projeto e organizacionais serem diferentes. Com isso, os projetos são separados para serem tratados separadamente conforme sua respectiva linguagem nos próximos passos.

Na prática, a estratégia seguida pela ferramenta utiliza URLs no padrão GitHub de filtro utilizando as palavras-chave para realizar os filtros e navegar entre as páginas, no modelo:

https://api.github.com/search/repositories?q={palavra-chave}+language:{linguagem} &sort=stars&order=desc&page={página}

Um exemplo, utilizando da palavra-chave *serverless* e linguagem *python*:

https://api.github.com/search/repositories?q=serverless+language:python&sort=stars linebreak&order=desc&page=0

Com o resultado da busca, são montados objetos de dados que representam as informações do repositório que serão relevantes para os próximos passos, como sua URL para download do código, conforme o exemplo na Figura 2.

```
{
    'description': 'Serverless email forwarding using AWS Lambda and SES',
    'title': 'arithmetric/aws-lambda-ses-forwarder',
    'url': 'https://github.com/arithmetric/aws-lambda-ses-forwarder',
    'language': 'JavaScript',
    'owner_url': 'https://github.com/arithmetric',
    'clone_url': 'https://github.com/arithmetric/aws-lambda-ses-forwarder.git',
    'included': True
},
```

Figura 2. Exemplo de objeto com informações de um repositório.

Onde *description* é a descrição do repositório no github, *title* é o nome do repositório, *url* é seu endereço de visualização HTTP, *language* indica a linguagem utilizada, *owner_url* aponta para o perfil do dono do repositório, *clone_url* é o endereço para se clonar o projeto, e *included* é uma variável interna de controle para caso de necessidade de exclusão do repositório do processo.

O código foi desenvolvido de forma genérica para navegar entre todos os resultados da busca, mesmo com as limitações da API do GitHub, para estar preparado para cenários genéricos e evoluções futuras. Decisões tomadas para contornar essas limitações são melhor descritas na subseção 4.3.4.

4.3.3 Processo de exclusão de falsos positivos selecionados no filtro

Com o intuito de remover possíveis projetos que não são de interesse, em especial *frameworks* e bibliotecas que auxiliam o desenvolvimento em *serverless* mas não são em si projetos práticos realizados para execução em ambiente *serverless*, foi definida uma regra para exclusão de alguns repositórios desse tipo do retorno do filtro, que são chamados de falsos positivos.

O processo definido para revisão dos repositórios retornados na busca é realizado com uma análise dos conteúdos do arquivo README do mesmo, buscando pelos termos "framework"ou "plugin". Apesar desse critério remover os repositórios que não são de interesse conforme mencionado anteriormente, ressalta-se que a exclusão de repositórios com esses termos no README pode acarretar na exclusão acidental de projetos que não são frameworks ou plugins, mas que fazem uso de algum desses e por isso possuem o termo presente no README. Atualmente essa é uma limitação conhecida da ferramenta e o processo de exclusão é parte das propostas de melhorias futuras de forma que se alcance outros métodos com menor risco de excluir repositórios de interesse.

4.3.4 Decisão final sobre repositórios analisados

Devido às limitações de busca da API do github e para que este trabalho possa servir como um ponto de referência inicial para desenvolvimento de estudos mais complexos no

tema, foi tomada a decisão de reduzir os repositórios considerados na aplicação da ferramenta e análise, buscando provar o conceito e ilustrar a aplicação que pode ser generalizada para qualquer repositório. Para isso, foi selecionada a lista de repositórios ilustrada na Tabela 2, já validados para a análise inicial neste trabalho. Os repositórios, de forma geral, contém aplicações projetadas para execução em ambiente *serverless* variadas, contendo desde funções de redirecionamento de e-mails até ferramentas de logging e de segurança compostas por diversas funções.

Identificador	**Endereço GitHub**	**Linguagem**
R1	*https://github.com/arithmetric/aws-lambda-ses-forwarder*	JavaScript
R2	*https://github.com/danilop/LambdAuth*	JavaScript
R3	*https://github.com/aws-samples/lambda-refarch-webapp*	JavaScript
R4	*https://github.com/aws-solutions/serverless-image-handler*	JavaScript
R5	*https://github.com/serverless/serverless-graphql-blog*	JavaScript
R6	*https://github.com/0x4D31/honeyLambda*	Python
R7	*https://github.com/Netflix/bless*	Python
R8	*https://github.com/DataDog/datadog-serverless-functions*	Python
R9	*https://github.com/servian/aws-auto-remediate*	Python
R10	*https://github.com/khornberg/elasticpypi*	Python
R11	*https://github.com/ellimilial/sqs-s3-logger*	Python
R12	*https://github.com/alecrubin/pytorch-serverless*	Python
R13	*https://github.com/jlhood/github-codebuild-logs*	Python

Tabela 2. Repositórios selecionados para análise

4.3.5 Processo de download do código fonte dos repositórios

Com essa lista de repositórios em mãos, foi utilizada a biblioteca GitPython para que seja realizado o download individualmente do código-fonte de cada projeto para a análise e extração das métricas conforme definidas previamente. É realizado um processo simples onde, para cada repositório, seu código é clonado, a extração das métricas é realizada e em seguida o repositório é excluído, visando evitar um uso desnecessário de disco no executor.

4.3.6 Estratégia para extração das métricas

Finalmente, a estratégia de extração de métricas foi desenvolvida com funções individuais para a extração de cada métrica, especificamente observando qual a linguagem na qual o projeto foi desenvolvido para guiar a lógica de como extrair as métricas, dadas as diferenças entre os projetos de cada linguagem.

As métricas são exibidas já num formato CSV após a execução da ferramenta e podem ser movidas diretamente para um arquivo CSV e abertas para análise em ferramentas como o Excel ou o Google sheets.

Com exceção das métricas de contagem de arquivos e tamanhos de arquivos, há métricas que avaliam práticas específicas dentro do código que são extraídas através do uso de ex-

pressões regulares para sua identificação. Além disso, também há algumas especificidades como a definição do que é considerado um arquivo de configuração ou arquivo de código para cada tipo de projeto. Todas essas decisões são detalhadas nos itens abaixo.

O que é considerado arquivo de configuração (métricas M4, M9)
Arquivos que contém algum dos termos abaixo em seu nome:

- ".json"
- "config."
- ".yml"
- ".conf"
- ".config"
- "settings."

O que é considerado arquivo de código (métricas E2, M5, M8, M10)
Arquivos que contém algum dos termos abaixo em seu nome:

- Python
  - ".py"
- JavaScript
  - ".js"

Como são identificados nomes de funções (métricas E3, M6, M10)
É utilizada uma expressão regular baseada na linguagem utilizada no projeto para encontrar o nome das funções no código, conforme abaixo:

- Python
  - *def (.*)\("*
- *JavaScript*
  - *exports\.(.*) = .**

Como são identificados parâmetros das funções (métricas M7)
Também é utilizada uma expressão regular baseada na linguagem utilizada no projeto para encontrar os parâmetros das funções, conforme abaixo:

- Python

  - *def .*\((.*)\).*:*

- *JavaScript*
  - *exports\..* = .*\((.*)\)*

Como são identificados os comentários no código (métricas M3, M11)

Também é utilizada uma expressão regular baseada na linguagem utilizada no projeto para encontrar os comentários presentes nas funções, conforme abaixo:

- Python
  a. *ˆ#.*$*
- *JavaScript*
  a. *ˆ\/*.*$*
  b. *ˆ\/\/.*$*

Tentativa de identificação do provedor de serviços na nuvem utilizado (métrica G2)

Foi criada uma proposta inicial para a tentativa de identificação do provedor na nuvem utilizado no projeto para referência baseada especialmente nas bibliotecas utilizadas nos projetos. Há chances de falsos positivos serem identificados com essa abordagem, e também é possível que após a execução da tentativa, ainda não seja possível identificar o provedor.

Essa tentativa tem a intenção de ser também um ponto de partida para a evolução futura da ferramenta, e é realizada da seguinte forma:

- Python
  - Microsoft Azure
    - Contém a biblioteca *azure.functions* ou o arquivo *function.json*
  - Google Cloud Platform
    - Contém a biblioteca *google.cloud*
  - Amazon Web Services
    - Contém a biblioteca *boto3*
- *JavaScript*
  - Microsoft Azure
    - Contém a biblioteca *azure/functions* ou o arquivo *function.json*
  - Google Cloud Platform
    - Contém a biblioteca *google-cloud*

- Amazon Web Services
  - Contém a biblioteca *aws-sdk* ou a função *exports.handler*

Caso nenhuma das condições detecte o provedor, o mesmo será marcado como nulo, pois essas checagens sozinhas não são capazes de identificar todos os cenários possíveis.

5 Avaliação

A execução da solução funcionou como esperado, sendo executada dentro de um container Docker em uma máquina com o mesmo instalado e disponível, e a saída foi movida para um arquivo CSV e analisado via Google Sheets.

5.1 5.1 Resultados da ferramenta

Após a execução da ferramenta, obtivemos os seguintes resultados:

Repo	**G1**	**G2**	**E1**	**E2**	**E3**	**E4**	**M1**	**M2**	**M3**	**M4**	**MDM5**	**MXM5**
R1	2	AWS	27	12	7	1257	9	3	43	4	4	6
R2	1	AWS	32	18	6	987	0	8	12	12	2.2	11
R3	3	AWS	55	38	6	3102	36	17	54	27	2.2	6
R4	4	AWS	38	19	2	6915	28	13	67	8	1.5	5
R5	2	AWS	15	11	1	529	20	3	4	7	3.7	4
R6	1	AWS	16	1	8	423	0	3	17	2	0.3	1
R7	3	AWS	85	54	198	3602	30	16	22	3	3.4	9
R8	6	AWS	146	25	211	6083	0	31	157	53	0.8	7
R9	4	AWS	59	23	172	3234	3	16	1	27	1.4	8
R10	2	AWS	30	15	47	933	2	5	2	5	3	8
R11	1	AWS	15	8	35	504	1	2	1	0	4	7
R12	2	AWS	26	19	165	3059	5	5	0	2	3.8	9
R13	2	AWS	28	15	66	909	0	5	9	4	3	8

Tabela 3. Métricas obtidas após execução - 1

Repo	**MNM5**	**MDM6**	**MXM6**	**MNM6**	**MDM7**	**MXM7**	**MNM7**
R1	1	11.4	19	7	1.7	4	1
R2	0	7	7	7	2	2	2
R3	0	13.3	16	11	2	2	2
R4	0	7	7	7	1.5	2	1
R5	1	7	7	7	2	2	2
R6	0	13.6	22	9	1.9	2	1
R7	1	23.8	67	3	1.3	4	1
R8	0	20.1	59	3	1.6	6	1
R9	0	17.4	64	2	1.7	3	1
R10	0	24.8	72	2	1.5	5	1
R11	0	18.8	74	4	1.6	6	1
R12	0	16.2	74	1	2.3	10	1
R13	0	18.7	51	7	1.8	6	1

Tabela 4. Métricas obtidas após execução - 2

Repo	**MDM8**	**MXM8**	**MNM8**	**MDM9**	**MXM9**	**MNM9**	**MDM10**
R1	104.8	385	3	39.8	97	9	0.6
R2	54.8	130	13	23	30	13	0.3
R3	81.6	190	10	80.8	190	10	0.2
R4	363.9	1760	0	244.6	1760	7	0.1
R5	48.1	154	1	38.1	154	1	0.1
R6	423	423	423	42.5	46	39	8
R7	66.7	558	0	162.3	244	26	3.7
R8	243.3	845	0	141.6	1189	1	8.4
R9	140.6	1044	0	73.9	329	4	7.5
R10	62.2	191	0	68.8	264	13	3.1
R11	63	190	1	0	0	0	4.4
R12	161	801	0	46	86	6	8.7
R13	60.6	149	12	66.2	194	13	4.4

Tabela 5. Métricas obtidas após execução - 3

Repo	**MXM10**	**MNM10**	**MDM11**	**MXM11**	**MNM11**
R1	6	0	3.6	37	0
R2	1	0	0.7	2	0
R3	1	0	1.4	9	0
R4	1	0	3.5	17	0
R5	1	0	0.4	4	0
R6	8	8	17	17	17
R7	33	0	0.4	9	0
R8	41	0	6.3	74	0
R9	36	0	0	1	0
R10	11	0	0.1	1	0
R11	15	0	0.1	1	0
R12	86	0	0	0	0
R13	13	0	0.6	3	0

Tabela 6. Métricas obtidas após execução - 4

Com as métricas retiradas a partir da ferramenta, pode-se ter uma ideia preliminar de possíveis práticas e padrões presentes nos projetos. As observações encontradas foram organizadas a partir das métricas que levantaram aspectos mais claros, além de algumas possibilidades que podem indicar a necessidade de estudos mais aprofundados e a evolução da ferramenta.

5.2 Análise preliminar das métricas obtidas

Tendo as métricas em mãos, foi realizada uma análise preliminar dos valores e dos possíveis significados e indícios que as mesmas fornecem sobre os projetos.

Profundidade de pastas

A maior profundidade de pastas e estruturas de arquivos nos repositórios selecionados foi de 6 níveis, e a menor foi de 1 nível, com números muito variados entre os repositórios sem apresentar diretamente um padrão. Um ponto identificado nessa métrica é que a ideia inicial de uso de Serverless que implica em funções pequenas e pouco código em um projeto pode não ser sempre uma realidade.

Se observamos o repositório mais destoante, que possui 6 níveis de profundidade, pode-se tirar a conclusão de que a profundidade maior é devido ao fato de que o projeto é desenvolvido com duas versões para serem executadas em provedores cloud diferentes, que gera alguns níveis de separação, e também pelo fato de que há pastas dentro de módulos de código incluindo testes e exemplos.

Quantidade de arquivos total e de código

A quantidade de arquivos identificada nos repositórios foi maior do que o esperado, com projetos grandes chegando até 146 arquivos totais, enquanto outros apresentam apenas 15 arquivos. Essa indicação novamente reforça a possibilidade de que projetos grandes também sejam desenvolvidos para uso em plataformas Serverless, o que pode indicar tanto que repositórios são utilizados para manter várias funções quanto a possibilidade de funções longas e complexas estarem sendo desenvolvidas. Ao observar o repositório com 146 arquivos, é possível notar-se que se trata de um projeto mais complexo, por coincidentemente ser o mesmo projeto que possuía 6 níveis de pastas mencionado na análise anterior, tendo segregação entre provedores cloud diferentes e arquivos de teste de integração e unidade.

Observando também a quantidade de arquivos de código, a presença de projetos grandes fica mais clara, dado o fato de que mesmo os repositórios com maior quantidade de arquivos no total não apresenta uma quantidade grande de arquivos de código. Isso indica que há uma quantidade maior de arquivos que podem ser *assets*, documentação e outros, indicando um projeto mais complexo.

Quantidade, tamanho e padrões de assinatura de funções

Em questão de quantidade de funções, observa-se uma variação muito evidente entre os repositórios, onde todos os repositórios JavaScript identificados na amostra possuem um número baixo, entre 1 e 8 funções, enquanto os projetos Python apresentam um número muito maior de funções, alguns possuindo mais de 100.

Também fica claro nas métricas que os projetos maiores tendem a ter funções com nome mais longo e receberem mais parâmetros no geral, além dos projetos possuírem arquivos com uma maior quantidade de funções, o que também indica que, conforme os projetos ficam maiores, estes podem ter sua complexidade de manutenção aumentada também.

Comentários e arquivos de configuração

Observa-se também uma distribuição variada da presença de arquivos de configuração no código, não tendo um padrão muito claro onde há projetos com uma quantidade grande de arquivos de configuração e outros com uma quantidade menor. Essa observação em específico é de grande interesse pelo fato de haver a necessidade de uma organização específica dos projetos para o uso de plataformas *serverless*, e se espera ao menos a necessidade de alguns arquivos de configuração para interação com tais plataformas.

No quesito de comentários, nota-se que a média de comentários por arquivo é bem baixa, o que pode indicar tanto uma baixa presença de comentários nos projetos quanto a centralização dos comentários em alguns arquivos, o que pode indicar funções e trechos de código mais complexos presentes nos projetos.

Visão geral e consolidação

De forma geral, as métricas apontam alguns indícios que indicam a existência prática de projetos grandes e complexos que fazem uso de plataformas *serverless*, o que em um primeiro momento parece diferir do esperado dado o conceito de Funções amplamente utilizado, levantando a possibilidade de se tratarem tanto de projetos de funções únicas complexas com manutenibilidade prejudicada, quanto de projetos de larga escala com o uso de múltiplas funções.

Olhando especificamente para as métricas de manutenibilidade, diversas métricas levantam pontos de partida para análises mais aprofundadas e entendimento das práticas, especialmente no que diz respeito à quantidade de funções presentes e tamanho de arquivos. Também vale ressaltar que algumas métricas de escala de projeto se apresentaram muito diferentes entre as duas linguagens de programação presentes nos projetos, o que também tem potencial de análise aprofundada para melhor entendimento do uso e adequação das linguagens com as plataformas *Serverless*.

6 Conclusão

Com as métricas coletadas, podemos observar que a ferramenta foi capaz de extrair o conteúdo de interesse seguindo a estratégia proposta, obtendo assim métricas de valor para iniciar estudos de práticas em projetos Serverless e cumprindo o objetivo de ser um ponto de partida tanto instrumental quanto teórico para desenvolvimento do estudo de práticas e padrões neste contexto.

Apesar do resultado favorável, ferramenta em seu estado inicial ainda possui algumas limitações como a possibilidade de alguns falso-positivos ainda passarem pela checagem feita após o filtro dos repositórios, e nem todas as possibilidades para os projetos serverless são cobertas na extração das métricas,

Atualmente há limitações no que diz respeito à identificação de arquivos como arquivos de configuração e de código, onde, por exemplo, um projeto JavaScript pode ter arquivos com extensões específicas como .ts para formatações em TypeScript, ou arquivos de configuração que podem ser específicos do projeto ou de extensões, além do fato que nem todo arquivo com as extensões consideradas necessariamente é um arquivo de configuração de interesse. Da mesma forma, as expressões regulares que identificam declarações de funções podem não identificar todas as funções presentes nos projetos, em especial quando se diz respeito à JavaScript, com a estratégia proposta buscando pelo padrão *exports* de declaração, que pode nem sempre ser utilizado.

Em especial, a ferramenta deve evoluir em diversos aspectos que podem enriquecer ainda mais as informações disponíveis para o estudo das práticas. Há espaço para derivação de métricas, análises gráficas automatizadas e até mesmo a adição de novas métricas como outros tipos de arquivos, melhoria do processo de detecção do provedor de serviços na nuvem, entre outros. Há também algumas métricas, como a identificação do provedor de serviços na nuvem, que não demonstraram relevância nas análises num primeiro momento, com exceção de demonstrar uma predominância do uso da Amazon, que indica que as métricas podem diferir ao observarmos projetos que usam outros provedores. Esse indício aponta que essa métrica pode passar a ser mais rica com o aumento da escala do estudo, analisando uma quantidade maior de repositórios no futuro.

Com os resultados obtidos, existem diversos indícios levantados pelas métricas que levantam questionamentos sobre as práticas aplicadas e o estado de manutenibilidade em especial dos projetos que utilizam Serverless atualmente. Esses indicativos são exatamente o que se espera como um ponto de partida para guiar estudos mais aprofundados e experimentos específicos em cada caso, para se entender melhor as práticas e clarificar comportamento e decisões que podem ser de fato práticas comuns e necessárias, padrões, e até mesmo antipadrões para esse tipo de projeto.

Referências

[1] Yussupov, Vladimir, et al. *"Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends."Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing. 2019.*

[2] Lenarduzzi, Valentina, et al. *Toward a technical debt conceptualization for serverless computing. IEEE Software 38.1 (2020): 40-47.*

[3] Nupponen, Jussi, and Davide Taibi. *Serverless: What it is, what to do and what not to do. 2020 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, 2020.*

[4] Taibi, Davide, et al. *Patterns for Serverless Functions (Function-as-a-Service): A Multivocal Review (2020.*

[5] Leitner, Philipp, et al. *"A mixed-method empirical study of Function-as-a-Service software development in industrial practice."Journal of Systems and Software 149 (2019): 340-359.*

[6] Baldini, Ioana, et al. *"Serverless computing: Current trends and open problems."Research advances in cloud computing. Springer, Singapore, 2017. 1-20.*

[7] Gong, Chunye, et al. *"The characteristics of cloud computing."2010 39th International Conference on Parallel Processing Workshops. IEEE, 2010.*

[8] Gamma, Erich, et al. *Design patterns: Elements of reusable software architecture. Reading: Addison-Wesley (1995).*

[9] Brown, William H., et al. *AntiPatterns: refactoring software, architectures, and projects in crisis. John Wiley & Sons, Inc., 1998.*

[10] Hassan, Ahmed E. *"The road ahead for mining software repositories."2008 Frontiers of Software Maintenance. IEEE, 2008.*

[11] Taibi, Davide, Josef Spillner, and Konrad Wawruch. *"Serverless Computing-Where Are We Now, and Where Are We Heading?."IEEE Software 38.1 (2020): 25-31.*

[12] Wu, Mingyu, Zeyu Mi, and Yubin Xia. *"A Survey on Serverless Computing and Its Implications for JointCloud Computing."2020 IEEE International Conference on Joint Cloud Computing. IEEE, 2020.*

[13] Hong, Sanghyun, et al. *"Go serverless: Securing cloud via serverless design patterns."10th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 18). 2018.*

[14] Van Eyk, Erwin, et al. *"The SPEC cloud group's research vision on FaaS and serverless architectures."Proceedings of the 2nd International Workshop on Serverless Computing. 2017.*

[15] Gábor Zöld. *7+1 Serverless Trends You Need to Know in 2020 - 2020. Disponível em: <https://codingsans.com/blog/serverless-trends>. Acesso em: 17/12/2021.*