Uma Caracterização em Larga Escala da Arquitetura de Sistemas Docker

Lucas Monteiro, Laerte Xavier, Marco Tulio Valente

[1]Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)

{lucasmonteiro,laertexavier,mtov}@dcc.ufmg.br

Resumo. *Docker é uma plataforma de software que permite a execução de aplicações de software em diferentes ambientes. Aplicações que utilizam Docker possuem diversos padrões arquiteturais que nem sempre são simples de se analisar. Dada a demanda atual por performance e simplicidade nas fases de desenvolvimento, testes e entrega de aplicações, essa plataforma tem sido amplamente utilizada, porém pouco analisada na literatura. Sendo assim, foram estudados mais de 29K arquivos coletados do GitHub, com a finalidade de se caracterizar a arquitetura de sistemas que usam múltiplos containers Docker. Por fim, são apresentadas conclusões sobre a arquitetura dessas aplicações.*

1. Introdução

A demanda por rapidez em execução e economia de recursos computacionais acarretou o crescimento da utilização da plataforma de software Docker em substituição às tradicionais máquinas virtuais [Initiative 2017]. A tecnologia por trás dessa infraestrutura possibilita que uma aplicação seja empacotada juntamente com suas dependências em um *container*, possibilitando a sua execução em qualquer ambiente. Além disso, desenvolvedores e administradores beneficiam-se de uma ambiente confiável e de baixo custo para criar, enviar e executar aplicações distribuídas[1]. Nesse contexto, Docker Compose destaca-se por ser uma ferramenta para definir e executar aplicações Docker com *múltiplos containers* inicializados através um único comando.

Um estudo recente de uma empresa de consultoria classificou Docker como o quinto projeto de software aberto mais relevante, com um *score* 22.61; perdendo apenas para Linux (*score* 100), Git (*score* 31.1), MySQL (*score* 25.23) e Node.js (*score* 22.75) [Dan Nguyen-Huu 2017]. Para produção desse *score*, foram combinadas diversas métricas, como frequência de buscas no Google, ofertas de empregos relacionadas ao projeto, *posts* no StackOverflow e popularidade no GitHub. Apesar do crescimento da utilização dessas tecnologias, poucos estudos concentram-se na análise e caracterização dos sistemas que as utilizam. Nesse contexto, um estudo recente analisou o ecossistema Docker no GitHub, concentrando-se principalmente em aplicações de um único *container* [Cito et al. 2017]. Entretanto, o estudo não discute aplicações que dependem de múltiplos *containers*, nem tão pouco caracteriza a arquitetura desses sistemas.

Neste trabalho, foram analisados sistemas Docker hospedados no GitHub com o objetivo de caracterizar, em larga escala, a arquitetura dos mesmos. Especificamente, foram analisados mais de 29K arquivos de configuração a fim de se revelar: (i) as principais

[1]https://aws.amazon.com/pt/docker/

linguagens de programação dos sistemas que utilizam essas tecnologias; (ii) o tamanho desses sistemas; (iii) sua popularidade [Borges et al. 2016]; (iv) as versões de Docker Compose mais utilizadas; (v) quantos serviços são providos; (vi) as dependências entre serviços; (vii) as redes que participam; e (viii) o número de portas que são expostas. Dessa forma, a principal contribuição deste trabalho é prover uma ampla caracterização da arquitetura de sistemas Docker com *múltiplos containers*.

2. Docker e Docker Compose

Containers são baseados em técnicas de virtualização de sistemas operacionais cujo objetivo é prover ambientes virtuais que permitam que processos sejam executados em redes isoladas. *Containers* definidos por Docker representam uma extensão dessas funcionalidades, uma vez que proveem uma API (*Application Programming Interface*) de alto nível com diversas funções, tais como iniciar, parar e deletar aplicações. Dessa forma, aumentam a reprodutibilidade de aplicações, permitindo o agrupamento de conteúdo em um único objeto que pode ser implantado em múltiplas máquinas [Boettiger 2015]. Nesse contexto, Docker Compose apresenta-se como uma ferramenta para definir e executar aplicações Docker com múltiplos *containers* através de um único arquivo de configuração chamado `docker-compose.yml`, o qual não necessariamente precisa estar no diretório raiz da aplicação. Dentre os principais usos dessa tecnologia, destacam-se: otimização de ambientes de desenvolvimento, de testes, de homologação e de fluxos de integração contínua.

O arquivo `docker-compose.yml` define diversas características da aplicação, tais como: serviços, portas e redes. Um serviço contém configurações que serão aplicadas a cada *container* iniciado (*e.g.,* servidores HTTP ou de banco de dados). Portas são pontos de acesso que a aplicação expõe para ambientes externos ao *container*, *i.e.,* para que sistemas externos se comuniquem com uma aplicação dentro de um *container* uma porta precisa ser definida como ponto de entrada. Por fim, redes são definidas para permitir a comunicação entre *containers*. *Containers* em redes distintas não podem se comunicar, *e.g.,* uma aplicação que possua serviços de *frontend* e *backend* pode definir que cada serviço referente a essas camadas execute em uma única rede comunicando-se entre elas.

```
1  version: '3'
2  services:
3      web:
4          build: .
5          ports:
6          - ''5000:5000"
7          networks:
8          - frontend
9      redis:
10         image: redis
11         networks:
12         - backend
```

Listagem 1. Exemplo de arquivo `docker-compose.yml`

A Listagem 1 apresenta um exemplo de arquivo `docker-compose.yml`. Nele, são definidos dois serviços: *web* e *redis* (linhas 3 e 9). Além disso, a aplicação expõe a porta 5000 para execução na máquina *host* (linha 6). A versão 3 é utilizada como padrão

(linha 1) e, por fim, observa-se que as redes *frontend* e *backend* são definidas nas linhas 8 e 12, respectivamente.

3. Metodologia

Seleção de Sistemas. Os sistemas analisados neste estudo foram obtidos através do BigQuery[2], um serviço de armazenamento e análise de dados na *web* oferecido pelo Google. Nele são disponibilizadas diversas bases de dados públicas, tais como: GitHub, Hacker News e Chicago Crime Data. A base referente ao GitHub contém mais de 3TB de dados, possuindo um *snapshot* completo de mais de 2 milhões de projetos. Desta forma, foram consultados todos os repositórios que utilizam Docker Compose, bem como a localização dos arquivos de nome `docker-compose.yml` em tais repositórios. A busca foi realizada no dia 10 de Junho de 2017 e retornou 33.135 registros. Após a obtenção desses registros, foi criado um *script* de consulta à API do GitHub com o objetivo de analisar o conteúdo de cada um desses arquivos. Dado que o BigQuery possui apenas um *snapshot* dos dados, e não a informação em tempo real, após buscar todos os arquivos no GitHub, foram retornados 30.142 arquivos (*i.e.,* 2.993 desses arquivos não foram encontrados).

Definição de Métricas. Segundo [Shaw and Garlan 1996], a arquitetura define o que é o sistema em termos de componentes computacionais e os relacionamentos entre estes componentes, os padrões que guiam a sua composição e restrições de comunicação. Dessa forma, foram definidas métricas de interesse que ajudam a entender melhor a arquitetura planejada dos sistemas estudados. São elas:

- **Linguagens de programação** mais utilizadas pelos sistemas;
- **Tamanho dos sistemas** analisados em termos de espaço utilizado em disco;
- **Popularidade** em termos de número de estrelas;
- **Versões** do Docker Compose mais utilizadas;
- **Dependências** de inicialização entre serviços da aplicação;
- **Redes** que cada serviço participa;
- Quantidade de **serviços** por aplicação;
- **Portas** expostas pelos serviços.

Extração de Métricas. Dos 30.142 arquivos obtidos, 776 foram considerados inválidos uma vez que não seguiam a sintaxe definida pelo padrão YAML. Dessa forma, 29.366 arquivos foram analisados com o objetivo de coletar cada uma das métricas definidas.

4. Resultados

Linguagem de Programação. A Figura 1 apresenta as dez linguagens de programação mais utilizadas em sistemas Docker. No total, foram observadas mais de 80 linguagens associadas aos repositórios com arquivo `docker-compose.yml`. Entre as principais linguagens, destacam-se: Python, com 5.888 sistemas e JavaScript, com 5.045.

Tamanho dos sistemas. A Figura 2 apresenta a distribuição do tamanho (em *megabytes*) de cada um dos sistemas analisados. Nesse caso, o primeiro quartil é igual a 154 KB; a mediana é 2.47 MB; e o terceiro quartil, 12.9 MB. Além disso, a análise dos *outliers* revela que o maior sistema analisado possui mais de 5 GB. Trata-se do repositório UC-DAVISLIBRARY/SCRIBEAPI, da Universidade da Califórnia - Davis. Nele, encontra-se o *framework* Scribe, utilizado para realizar *crowdsourcing* de documentos.

[2]https://cloud.google.com/bigquery

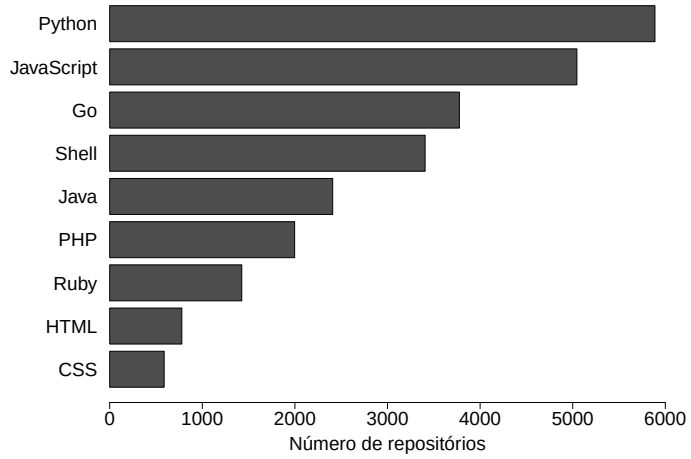

Figura 1. Top 10 linguagens mais utilizadas em sistemas Docker

Popularidade A Figura 3 apresenta a distribuição do número de estrelas por repositório. O primeiro quartil e a mediana são 0, enquanto que o terceiro quartil é 2. Dessa forma, observa-se que, em geral, os sistemas analisados não são populares, com reduzido número de estrelas no GitHub. Por outro lado, a análise dos *outliers* revela que o sistema SOCKETIO/SOCKET.IO, responsável por prover um *framework* de aplicação em tempo real, possui mais de 30K estrelas.

Versão. Do total de arquivos analisados, a distribuição de versões de Docker Compose mais utilizadas é apresentada na Tabela 1. Nesse caso, observa-se que a grande maioria das aplicações (64%) ainda utiliza a primeira versão. Por outro lado, apenas 2.45% dos sistemas analisados utilizam a versão mais recente, beneficiando-se de melhorias, tal como acesso a novas funcionalidades.

Versão	Número de arquivos
1.0	18773 (64.94%)
2.0	9866 (33.60%)
3.0	721 (2.45%)

Tabela 1. Versões mais utilizadas do Docker Compose

Serviços. Dentre os mais de 29K arquivos analisados, foram encontrados um total de 86.067 serviços. A Figura 4 apresenta a distribuição do número de serviços por sistema, com primeiro quartil, mediana e terceiro quartil iguais a 1, 2 e 3, respectivamente. Nesse caso, observa-se que a grande maioria dos sistemas analisados utiliza uma pequena quantidade de serviços (mediana igual a 2), o que pode ser um indicativo de que: (i) a tecnologia Docker tem sido utilizada com mais frequência em sistemas simples; ou que (ii) os desenvolvedores ainda não utilizam, de fato, todo o potencial da ferramenta. Entretanto, a análise de *outliers* destaca o repositório FRANCIS826/DOCKER-PHP-MINIMAL que define 382 serviços com o objetivo de definir um ambiente de execução para diversas versões da linguagem de programação PHP.

Dependências. Dos 86.067 serviços analisados, foram encontradas 10.664 dependências.

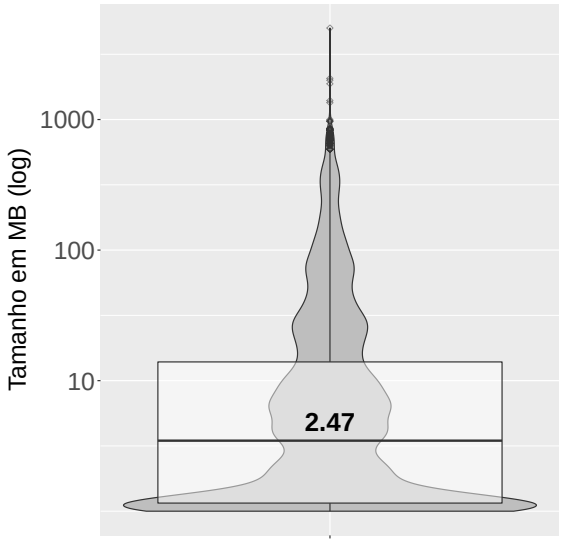

Figura 2. Distribuição do tamanho dos sistemas analisados

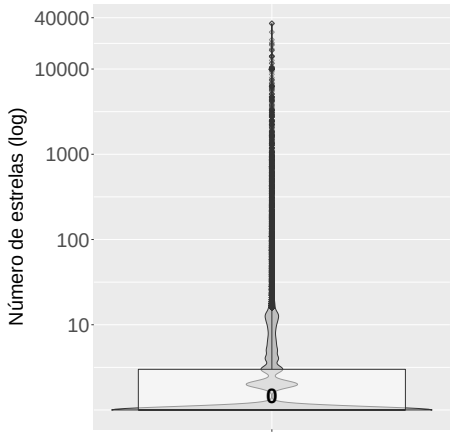

Figura 3. Distribuição do número de estrelas

A Figura 5 apresenta a distribuição de dependências por sistema. Nesse caso, nota-se que o primeiro quartil, a mediana e o terceiro quartil são iguais a 0 (*i.e.,* não expressam nenhuma dependência entre os serviços definidos). Tais valores podem ser explicados, principalmente, pelo fato de que Docker não garante que um serviço estará pronto (somente iniciado) antes de outro serviço dependente iniciar. Além disso, o fato de que os sistemas que utilizam essa tecnologia serem mais simples (conforme detalhado anteriormente) pode impactar na baixa quantidade de dependências entre serviços. Entretanto, a análise dos *outliers* mostra que Docker também é utilizado para aplicações complexas, como é o caso do sistema APPCELERATOR/AMP, com 13 dependências. Essa aplicação provê um CaaS (*Container as a Service*) para Docker. CaaS é um modelo para organizações e desenvolvedores trabalharem na construção, entrega e execução de aplicações em qualquer lugar. Dada a complexidade desse tipo de aplicação, ela é dependente de vários serviços especificados no arquivo `docker-compose.yml`, tais como: Apache, Kafka e MySQL.

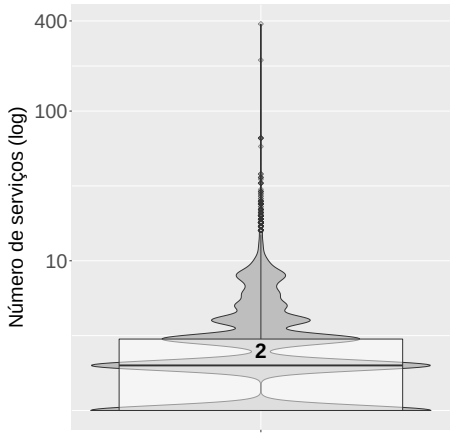

Figura 4. Distribuição da quantidade de serviços

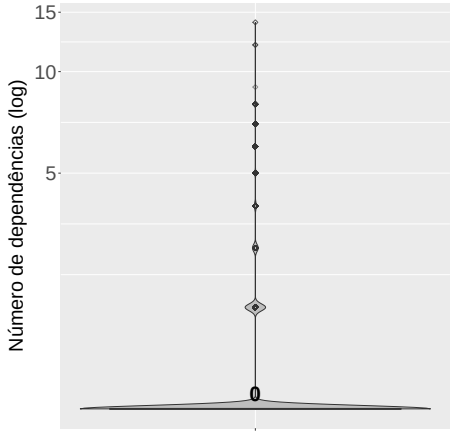

Figura 5. Distribuição da quantidade de dependências

Redes. Do total de serviços encontrados, foram obtidas 1.833 redes. A Figura 6 mostra a análise de redes, onde nota-se que o primeiro quartil, a mediana e o terceiro quartil são 0. Segundo a documentação de Docker, é recomendado que um usuário defina suas próprias redes para controlar quais *containers* podem comunicar-se entre si. Os baixos valores obtidos mostram que: (i) a definição de redes pode não ser relevante para a grande parte das aplicações; ou (ii) os desenvolvedores não conhecem os benefícios dessa funcionalidade. Entretanto, a análise dos *outliers* evidência uma aplicação que possui seis redes: DESEC-IO/DESEC-STACK. Trata-se de um sistema que provê uma infraestrutura básica para serviços que tornam roteadores residenciais acessíveis pela internet. Nesse caso, devido as características intrínsecas ao domínio da aplicação, observamos a utilização de redes que definem a comunicação entre seus serviços, *e.g.,* a rede *front*.

Portas. Por fim, a Figura 7 apresenta a distribuição da quantidade de portas expostas pelos sistemas estudados. Nesse caso, observa-se que o primeiro quartil é 0, enquanto que a mediana e o terceiro quartil são 1. Conforme esperado, os valores obtidos evidenciam que grande parte dos sistemas expõe apenas um único canal de comunicação com sistemas externos. Em outras palavras, nota-se que somente uma aplicação principal é acessada externamente, enquanto que todas as outras encapsuladas no *container* comunicam-se

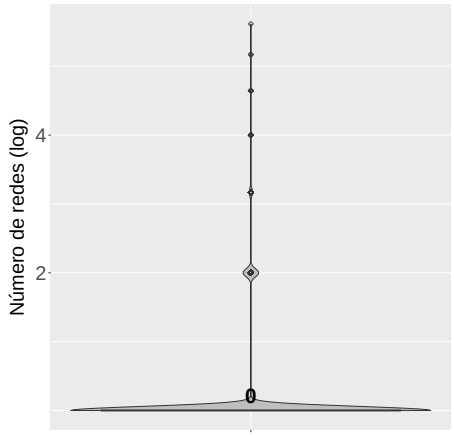

Figura 6. Distribuição da quantidade de redes

internamente entre elas. Por outro lado, a análise dos *outliers* revela a utilização de Docker Compose com 106 portas no sistema COAXIAL/MOSHIMOSHI. Trata-se de uma aplicação VoIP que necessita trabalhar com diversas portas utilizadas pelo próprio VoIP, tais como SIP (*Session Initiation Protocol*) e RTP (*Real-time Transport Protocol*).

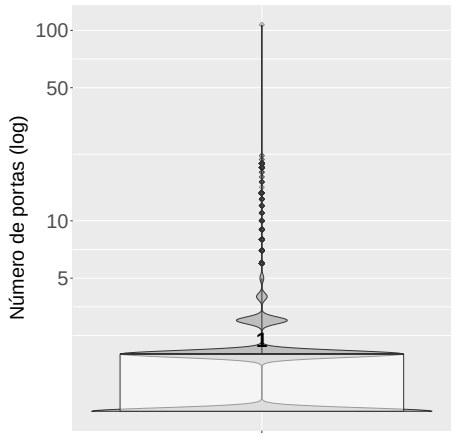

Figura 7. Distribuição da quantidade de portas

5. Ameaças à validade

Validade de Construção. Este estudo parte do princípio de que todo arquivo de configuração de múltiplos *containers* Docker utiliza o nome padrão `docker-compose.yml`. Sistemas que utilizam arquivos com nomes alternativos foram desconsiderados pela busca. Além disso, repositórios que possuem cópias através de *forks* podem enviesar os resultados apresentados, uma vez que tiveram seus arquivos de configuração analisados de forma repetida.

Validade Interna. O algoritmo implementado para fazer a extração das métricas estudadas foi baseado nos padrões estabelecidos na documentação de Docker. Sendo assim, aplicações que usam diferentes padrões de configuração podem ter tido suas métricas calculadas erroneamente.

Validade Externa. Este estudo limitou-se a analisar os sistemas hospedados no GitHub. Dessa forma, não é possível garantir que as características aqui apresentadas representem todo o ecossistema de aplicações Docker com múltiplos *containers*.

6. Trabalhos Relacionados

A literatura que trata a arquitetura dos sistemas definidos por Docker é bastante limitada. Num estudo recente [Cito et al. 2017] analisaram empiricamente a qualidade de *containers* Docker no GitHub. Para tanto, foram analisados 70.197 arquivos. Um dos objetivos do trabalho foi comparar o uso de Docker na população geral de projetos do GitHub e como essa ferramenta é utilizada pelos projetos mais populares. Neste trabalho, porém, foram analisados aspectos arquiteturais de aplicações Docker, visando extrair métricas que caracterizam essas aplicações, incluindo serviços, dependências, redes e portas.

7. Conclusão

Este trabalho consistiu de uma análise em larga escala das características dos arquivos de configuração de aplicações Docker que utilizam múltiplos *containers*. Foram analisados mais de 29K arquivos coletados de projetos públicos do GitHub. Métricas para a caracterização da arquitetura de sistemas Docker foram definidas e discutidas em relação aos resultados alcançados.

Este trabalho representa um primeiro passo na caracterização deste tipo de aplicação, uma vez que existem poucos trabalhos na literatura que abrangem essa tecnologia emergente. Dessa forma, espera-se incluir, como trabalho futuro, diferentes propriedades dos arquivos de configuração para análise, além de ampliar as bases de dados, a fim de minimizar as possibilidades de enviesamento das métricas coletadas.

Agradecimentos

Esta pesquisa é financiada pela FAPEMIG, CNPq e CAPES.

Referências

Boettiger, C. (2015). An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of GitHub repositories. In *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344.

Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., and Gall, H. C. (2017). An empirical analysis of the docker container ecosystem on GitHub. In *14th International Conference on Mining Software Repositories (MSR)*, pages 323–333.

Dan Nguyen-Huu, Dharmesh Thakker, M. S. (2017). The boss index: Tracking the explosive growth of open-source software.

Initiative, O. C. (2017). Faq open containers initiative.

Shaw, M. and Garlan, D. (1996). *Software Architecture - Perspectives on an emerging discipline*. Prentice Hall.