Conteinerização e CI/CD

João Pedro Dias[1]

Universidade de Évora, Évora, Portugal
m42055@alunos.uevora.pt

Resumo Ao olhar em detalhe para o desenvolvimento de aplicações de software na atualidade, é notório o aumento da complexidade das mesmas. Este aumento de complexidade é explicado pela necessidade cada vez maior de aplicações com maior poder computacional e que consigam acolher uma maior quantidade de utilizadores. O referido aumento na complexidade de desenvolvimento faz com que o setor necessite de uma renovação e de procurar alternativas que agilizem este mesmo processo, tornando-o mais simples e mais veloz. No presente artigo serão abordadas duas metodologias que ajudam a atingir o objetivo previamente mencionado, falo de conteinerização e CI/CD. Estes dois conceitos serão delineados ao longo do artigo, apresentando também algumas *frameworks* que disponibilizam a utilização dos mesmos. Em última análise, olhar-se-á para as vantagens resultantes da utilização conjunta destas duas metodologias.

Keywords: Conteinerização · CI/CD · Desenvolvimento · Software

1 Introdução

A sociedade contemporânea tem aumentado gradualmente a sua dependência sobre sistemas de software, havendo inclusive alguns sectores cujo seu desempenho depende de aplicações e sistemas de elevado nível de complexidade. Por esse motivo, estes sistemas requerem novas abordagens e tecnologias de desenvolvimento para aumentar a performance e também para melhorar o processo de entrega do produto final. Dentro destas novas tecnologias podemos encontrar dois conceitos muito recentes, conteinerização e CI/CD, que serão a matéria em estudo do presente artigo.

Os primeiros dois capítulos tocarão no aspeto da conteinerização. Esta temática tem sido extremamente impulsionada pela ferramenta Docker que, ao reutilizar o conceito já existente de execução de processos em ambientes isolados, tornou esta abordagem acessível e de fácil compreensão a todos os programadores [1]. No entanto é necessário perceber os aspetos da conteinerização, como a criação de imagens de aplicações e a execução de contentores, para só depois fazer o uso devido desta metodologia de desenvolvimento. Ao longo desta fase inicial do artigo serão ainda revistas algumas alternativas ao Docker, de maneira a entender as principais diferenças e perceber as circunstâncias em que cada uma deve ser utilizada.

A segunda temática abordado neste artigo, que diz respeito à integração e entrega contínua (CI/CD), é posteriormente explorada ao longo do terceiro capítulo. Estes dois conceitos tem como principal objectivo automatizar muitas das tarefas, que anteriormente eram realizadas por humanos, na fase de implementação de alterações numa aplicação. Como consequência deste automatismo, as equipas ficam com mais tempo para se focarem em desenvolver a aplicação, ao invés de corrigir erros de integração e *deploy*. É por isso uma tecnologia com grande potencial, mas com a contrapartida de carecer de uma boa compreensão por parte da equipa antes da implementação, já que uma pipeline CI/CD mal elaborada pode gerar consequências negativas e irreversíveis no produto final.

Na secção final deste artigo, e após as duas temáticas serem descritas, tentar-se-á perceber as vantagens de utilizar as duas em simultâneo, ou seja, as vantagens de utilizar uma pipeline CI/CD em aplicações conteinerizadas. Esta estratégia de desenvolvimento merece ser analisada pois é notório o aumento da procura da mesma por parte de grandes empresas no sector de desenvolvimento de aplicações de *software*. A escolha tem origem na melhor velocidade de entrega do produto e também na necessidade de operar com estratégias mais adequadas para equipas de grandes dimensões.

2 Conteinerização

O conceito de conteinerização é algo que nos últimos anos tem revolucionado a maneira como se constrói software, trazendo largos benefícios não só ao nível do desenvolvimento de aplicações mas também no que diz respeito à sua manutenção e *deploy* em diversos ambientes. Apesar de ser possível executar processos de forma isolada desde uma longa data, apenas em 2013, a ideia de conteinerizar aplicações começou a ganhar relevância. Tendo como maior impulsionador desta renovação a empresa Dotcloud, que através da ferramenta Docker, facilitou o uso de contentores para qualquer programador [2].

Resumidamente, o termo conteinerização pode ser definido como o processo de encapsulamento, dentro de um contentor, de todo o código fonte de uma aplicação juntamente com todas as suas dependências [3]. Estes contentores, podem depois ser executados uniformemente em qualquer infraestrutura, garantindo sempre o mesmo *output* para as aplicações contidas no seu interior, e nunca perdendo as suas propriedades. É ainda possível executar em simultâneo vários contentores, inclusive várias instâncias do mesmo contentor, porque ao serem executados diretamente sob o sistema operativo da maquina anfitriã, partilham os seus recursos e conseguem comunicar entre si .

Esta abordagem traz por isso diversas vantagens no processo de desenvolvimento de software. A portabilidade é uma dessas vantagens, visto que ao produzirmos um contentor executável baseado numa aplicação não ficamos dependentes do sistema operativo no qual a aplicação foi desenvolvida. Isto permite que possamos disponibilizar a nossa aplicação para qualquer plataforma, desde que tenha capacidade para executar contentores. Outra das vantagens é a agilidade e eficiência que oferece na etapa de desenvolvimento. O facto de todos os

contentores partilharem os mesmos recursos da maquina anfitriã oferece maior velocidade e maior poder de comunicação entre contentores quando comparado a outras alternativas na industria.

Estas e outras vantagens fazem com que este procedimento seja diversas vezes comparado à utilização de maquinas virtuais. Ao contrário do que acontece durante a conteinerização, quando se usa máquinas virtuais para alojar a nossa aplicação, todo o ambiente a seu redor têm de ser reproduzido, inclusive o sistema operativo. Isto implica uma virtualização dos recursos físicos da máquina anfitriã, resultando na alocação destes recursos exclusivamente para uso daquela máquina virtual. Assim, é fácil perceber a principal vantagem em usar conteinerização, visto que os recursos usados por um contentor nunca ficam bloqueados apenas para ele próprio.

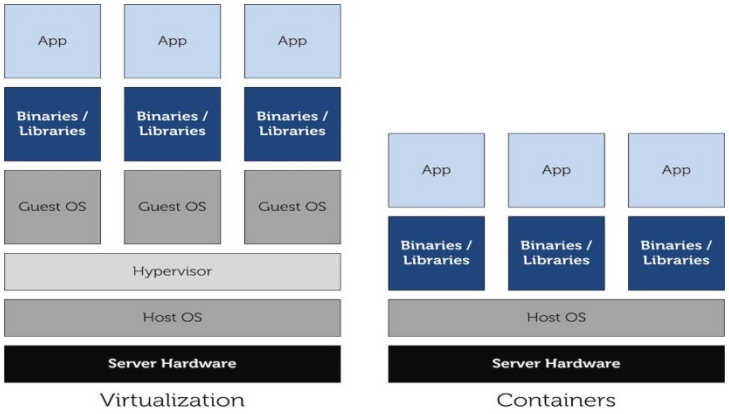

Figura 1. À esquerda um exemplo de virtualização com três maquinas virtuais e à direita um exemplo de conteinerização com três contentores

Estudos que comparam a utilização de ambas as alternativas demonstram que metade dos programadores que escolhem conteinerização em virtude de virtualização, o fazem devido à melhor performance que a primeira opção oferece [4]. Esta diferença de performance pode ser justificado pela ausência de comunicação entre *kernels* na conteinerização, dado que no caso de virtualização o *kernel* do sistema operativo da maquina virtual necessita comunicar com o *kernel* do sistema operativo anfitrião. No entanto, virtualização pode ser visto como a melhor opção quando se pretende um melhor isolamento da própria aplicação em termos de segurança [5].

Ainda no contexto da segurança, quando temos mais que um contentor interligados, formando em conjunto uma só aplicação, podem surgir algumas vulnerabilidades. Isto porque, um agente malicioso que consiga aceder a um destes contentores pode facilmente chegar aos que partilham a mesma rede. O mesmo se pode dizer para contentores que estejam a ser executados no mesmo sistema ope-

rativo, e que não façam parte daquela aplicação. Nesta situação, caso exista uma vulnerabilidade no sistema operativo, todos os contentores assentes no mesmo podem ser afetados [3].

3 Docker e Alternativas

Existem diversas alternativas para pôr em prática desenvolvimento baseado na conteinerização de aplicações, mas sem dúvida a mais utilizada é o Docker. Docker é um mecanismo *open-source* que automatiza o processo de implantação de uma aplicação num contentor. Oferecendo assim diversas funcionalidades, tais como, a criação de imagens de aplicações, criação de contentores e execução das aplicações, contidas nestes mesmos contentores. Numa visão mais geral, o Docker pretende ser uma ferramenta que permita simular um ambiente leve, onde a aplicação possa ser executada, e também ambiciona facilitar o processo de migração de uma aplicação do ambiente de testes para produção [1].

O que distingue o Docker das outras alternativas no mercado para conteinerizar aplicações é acima de tudo a grande comunidade que a suporta. Por ser a solução mais utilizada mundialmente, faz com que exista atualmente uma grande variedade de imagens, que podem ser reutilizadas, agilizando o processo de criação de novas aplicações. Grande parte destas imagens são acessíveis a qualquer programador através do repositório DockerHub [6].

Dentro das alternativas ao Docker destaca-se a Linux Containers, com os projetos LXC e LXD [7]. Esta alternativa serviu de base para o desenvolvimento do Docker numa fase inicial, e oferece ao utilizador de sistemas Linux, a possibilidade de criar contentores com um sistema operativo diferente no seu interior, simulando ao máximo um ambiente semelhante ao oferecido por uma máquina virtual. A principal diferença é o facto de não se virtualizar o hardware, usando assim diretamente o *kernel* do sistema anfitrião e partilhando os seus recursos pelos vários contentores em execução. Esta partilha de recursos é semelhante ao comportamento do Docker, sendo que no Docker os contentores apenas contem uma única aplicação enquanto que em Linux Containers um contentor oferece um ambiente completo, que suporta várias aplicações.

Existem ainda outras alternativas ao Docker como o caso de Rkt [8] e Podman [9]. À semelhança do Docker estas alternativas também oferecem um mecanismo para criar, gerir e executar contentores com aplicações no seu interior. No entanto algumas diferenças podem ser notadas como no caso do Podman que permite também trabalhar com *pods* (conjunto de contentores), semelhante ao comportamento que obtemos de ferramentas como Kubernetes, que permitem orquestrar em simultâneo uma grande quantidade de contentores. Adicionalmente, estas alternativas também conseguem utilizar diversos tipos de imagens, inclusive imagens criadas a partir do Docker.

Apesar de existirem várias alternativas, Docker continua a ser o líder do mercado no que diz respeito à conteinerização, oferecendo também outros produtos como docker-compose, para executar vários contentores em simultâneo no caso

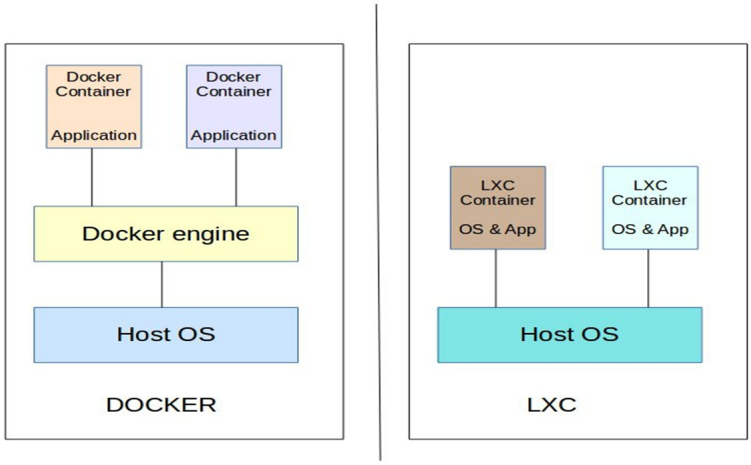

Figura 2. À esquerda um exemplo onde se utiliza Docker e à direita um exemplo que faz uso de Linux Containers

de aplicações que estejam divididas em vários módulos, e docker-swarm para gerir aglomerados de contentores [1].

4 CI/CD - Continuous Integration/Continuous Delivery

De modo a compreender a importância do conceito CI/CD é preciso perceber primeiro o seu significado. A sigla CI/CD traduz-se em *Continuous Integration* e *Continuous Delivery*, que em português representa integração e entrega contínua de software. É ainda possível estender esta temática através do termo *Continuous Deployment*, refletindo a automatização por completo de todo o processo CI/CD.

A primeira prática denominada de integração contínua está fortemente ligada ao desenvolvimento de software feito em equipa. É normal que num projeto com alguma complexidade exista uma equipa de programadores, onde cada elemento está autorizado a fazer alterações ao código da aplicação. Logo é necessário garantir, que no final do ciclo de desenvolvimento, todas as alterações de autores diferentes serão integradas corretamente, resultando num produto funcional. O conceito de integração continua [10] veio, de certo modo, agilizar e automatizar este processo, assegurando que antes de ser integrada qualquer alteração ao repositório global da aplicação, é efetuado o *build* da mesma juntamente com alguns testes pré-definidos. Assim, qualquer erro que exista nos desenvolvimentos será detetado ainda numa fase inicial do ciclo de desenvolvimento, podendo ser rapidamente corrigido.

Após a implementação do processo de integração contínua, é possível automatizar também o processo de implementação da aplicação, com as novas alterações, num ambiente que não seja produção. Este processo é conhecido por entrega contínua, e é possível porque após a integração com sucesso de uma nova

alteração, ficamos com um produto final funcional e que sabemos que foi testado com êxito. Para isto é necessário automatizar algumas tarefas que permitam copiar e instalar a aplicação no ambiente desejado. Nestes ambientes a aplicação pode então ser testada por completo com todas as suas dependências, de forma a que, assim que seja desejado, a mesma possa ser implementada no ambiente de produção, através do mesmo automatismo.

Como uma extensão do processo de entrega contínua temos o processo de implementação contínua ou *continuous deployment*. Este último processo vai um passo mais longe e automatiza por completo todo o pipeline CI/CD, removendo a necessidade de uma autorização externa na fase de implementação da aplicação em ambiente de produção [10]. Como é obvio este processo requer um pipeline bem robusto e que execute uma variedade de testes, pois é possível que uma simples alteração, que seja automaticamente colocada em produção, não possa ser revertida, causando assim estragos consideráveis.

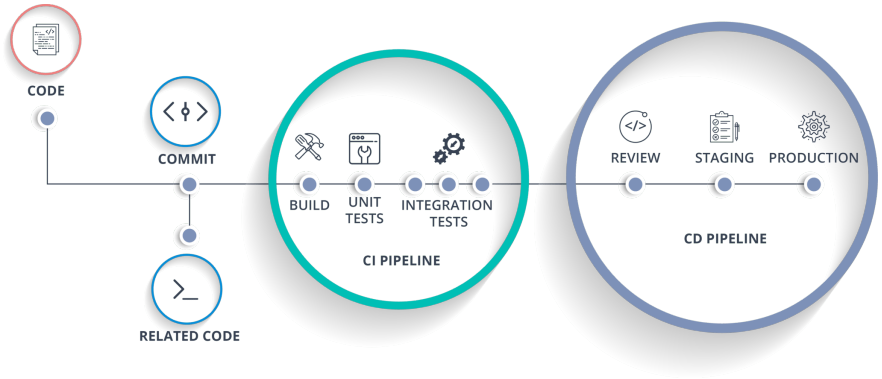

Figura 3. Etapas do pipeline CI/CD no processo de desenvolvimento de software

Percebendo todo o conceito de CI/CD é fácil compreender a necessidade que uma equipa de desenvolvimento tem em usar estes processos. Numa época em que tempo é dinheiro e onde o software é cada vez mais complexo, é possível, assim, ganhar tempo de desenvolvimento e realizar entregas dos produtos mais eficientes. Também o facto de erros de desenvolvimento serem detetados numa fase mais inicial faz com que o código seja entregue com maior qualidade e que os programadores não percam tanto tempo a resolver erros numa fase mais avançada do projeto [11].

De forma a implementar toda esta pipeline de desenvolvimento de software existem à disposição variadas *frameworks* como o Jenkins, GitLab ou Travis CI. Existem algumas diferenças entres estas *frameworks*, como por exemplo, o facto do Jenkins não oferecer nenhum repositório para guardar o código, tendo de se recorrer a *plugins* para fazer a ligação a serviços como o GitHub, enquanto que o GitLab oferece o seu próprio repositório. No que diz respeito ao processo de

construir a aplicação após uma alteração, as *frameworks* são muito semelhantes, recorrendo a ferramentas como o Docker, que agilizam este processo. Por último existem ainda algumas diferenças no que diz respeito à construção da pipeline CI/CD, não só na maneira de criar os ficheiros de configurações mas também a nível de interface gráfica.

5 CI/CD em Aplicações Conteinerizadas

Os conceitos previamente abordados, apesar de serem recentes no contexto do desenvolvimento de software, vieram revolucionar o mesmo na maneira como agilizaram todo o processo de desenvolvimento, desde a fase implementação até à fase de manutenção, passando pela fase de testes. No entanto, ao juntarmos estes dois conceitos num só projeto é notório o surgimento de algumas vantagens que seriam impossíveis de alcançar usando apenas uma destas tecnologias.

O facto da conteinerização habilitar a criação de contentores leves que permitem a execução de software de forma portátil faz com que seja uma boa técnica para utilizar em CI/CD. Isto porque ao utilizar o método *agile* de desenvolvimento, as muitas alterações executadas resultarão numa grande carga ao nível da necessidade de realizar uma grande quantidade de testes à construção da aplicação. Ao utilizar contentores nesta etapa o processo será muito mais rápido e não vai sofrer influências de elementos que não digam respeito à aplicação em específico [12].

Um aspeto importante no processo de desenvolvimento moderno é a arquitetura de aplicações baseada em microserviços. A conteinerização é um grande aliado a este tipo de arquitetura, pois facilita a criação destes microserviços, recorrendo a tecnologias distintas, sem perder a ligação que possa existir entre os vários serviços. Ao aplicar integração e entrega continua sob este tipo de aplicações é possível separar as diversas equipas para trabalharem num só serviço ou tecnologia, reduzindo a preocupação da aplicação falhar na altura de integrar todos os serviços [13].

6 Conclusão

Com este artigo foi possível definir e explorar o conceito de conteinerização e também de CI/CD. Ambos os conceitos estão em destaque no processo de desenvolvimento de software e são uma mais valia para qualquer programador quer esteja a desenvolver uma aplicação muito ou pouco complexa. Quando aliados, estas duas técnicas trazem ainda mais vantagens, facilitando a tarefa de cada um.

No caso da conteinerização, é notória a influencia do Docker como principal impulsionador. Esta abordagem permite a qualquer programador focar-se, exclusivamente, no desenvolvimento da aplicação, deixando de parte preocupações que dizem respeito a saber se a aplicação vai ter o mesmo comportamento em ambientes diferentes ao inicial. É por isso uma excelente alternativa à técnica

de virtualização, muito usada anteriormente, onde se têm de replicar todo um ambiente para conseguir implementar uma ou mais aplicações.

Quando olhamos para CI/CD as conclusões são semelhantes, no âmbito em que esta metodologia originou um aumento da velocidade de todo o ciclo de desenvolvimento. Ao termos a possibilidade de automatizar todas estas tarefas, consegue-se alocar mais tempo para os programadores se focarem em novos desenvolvimentos ou até mesmo correções na aplicação. Contudo é sempre necessário controlar o pipeline, de modo a nos certificarmos que tudo corre como planeado.

Finalmente ao abordarmos os dois conceitos em conjunto percebe-se que surgem novas vantagens que melhoram o desenvolvimento de aplicações conteinerizadas. Como já existem diversas *frameworks* de integração e entrega contínua que recorrem a Docker, a integração torna-se bastante fácil e intuitiva, tornando-se num processo de desenvolvimento bastante ágil e com performance mais elevada. A junção destas duas tecnologias facilita ainda o trabalho em equipa, pois em muitos casos um projeto terá equipas a trabalhar separadamente em serviços distintos, cabendo ao pipeline executar a integração dos mesmos.

Referências

1. 5.Turnbull, J.: The Docker Book. (2018).
2. Cantrill, B., 2016. The Container Revolution: Reflections After The First Decade.
3. Conteinerization Explained | IBM, `https://www.ibm.com/cloud/learn/containerization`. Last accessed 11 Jun 2020
4. Diamanti 2019. 2019 Container Adoption Benchmark Survey. Available at: `https://diamanti.com/wp-content/uploads/2019/06/Diamanti_2019_Container_Survey.pdf`. Last accessed 11 Jun 2020
5. Jeroen Scheepers, M.: Virtualization and Containerization of Application Infrastructure: A Comparison. (2017).
6. Docker Hub, `https://hub.docker.com`. Last accessed 11 Jun 2020
7. Linux Containers, `https://linuxcontainers.org`. Last accessed 11 Jun 2020
8. CoreOS, `https://coreos.com/rkt/`. Last accessed 11 Jun 2020
9. Podman, `https://podman.io`. Last accessed 11 Jun 2020
10. Krief, M.: Learning DevOps. Packt Publishing Ltd., Birmingham (2019).
11. Why is CI/CD Important in Software Development Lifecycle?, `https://medium.com/@james.richardson.987123/why-is-ci-cd-important-in-software-development-lifecycle-c75f52cbed83`. Last accessed 11 Jun 2020
12. CI/CD With Containers - Dzone Refcardz, `https://dzone.com/refcardz/cicd-with-containers?chapter=4`. Last accessed 11 Jun 2020
13. Atkinson, B., Edwards, D.: Generic Pipelines Using Docker. Apress, Berkeley, CA (2018).