ISSN 1677-8464

Foto:Pexels

COMUNICADO TÉCNICO

129

Campinas, SP
Dezembro, 2018

Embrapa

Encapsulando um Sistema Gerenciador de Ensaios de Genotipagem e de Marcadores Moleculares em Containers Docker

Eliseu Germano
Marcelo Gonçalves Narciso
Edgard Henrique dos Santos

Encapsulando um Sistema Gerenciador de Ensaios de Genotipagem e de Marcadores Moleculares em Containers Docker

Eliseu Germano, bacharel em Ciências da Computação, bolsista da Embrapa Arroz e Feijão, Santo Antônio de Goiás, GO; **Marcelo Gonçalves Narciso**, engenheiro de Telecomunicações, doutor em Computação Aplicada, pesquisador na Embrapa Arroz e Feijão, Santo Antônio de Goiás, GO; **Edgard Henrique dos Santos**, analista de sistemas, analista na Embrapa Informática Agropecuária, Campinas, SP.

Resumo

O crescente número de componentes de software a serem integrados e testados durante o processo de implantação de um sistema exige pleno controle de alocação dos recursos computacionais. A solução utilizada durante o desenvolvimento do GeneMaisLab, "Sistema gerenciador de dados de genotipagem", consiste no uso de serviços da plataforma Docker e do sistema de distribuição de carga HAProxy para o escalonamento dos serviços web. O Docker é uma ferramenta projetada para facilitar a criação, a implementação e a execução de aplicativos usando *containers*. Esses permitem que um desenvolvedor empacote um aplicativo com todas as partes que precisa, como bibliotecas e outras dependências, e envie tudo como um único pacote de software.

A solução utilizada no GeneMaisLab pode ser replicada em qualquer sistema, independente da complexidade de adaptação e da composição dos serviços, com auxílio de ferramentas usadas para gerenciamento de *clusters* de containers da plataforma Docker.

Introdução

As práticas contemporâneas de desenvolvimento de software incentivam a construção de sistemas seguindo abordagens baseadas em componentes, fomentando o uso de elementos práticos, lógicos e independentes, agrupados conforme a arquitetura do sistema (Kaur; Mann, 2010). Esta estratégia oferece vantagens como a simplificação quanto ao reúso de componentes. No entanto, torna-se comum o surgimento de inconsistências após a portabilidade de ambientes, aumentando-se a complexidade dos processos de testes, de implementação e de integração do sistema.

Outro desafio durante o desenvolvimento do GeneMaisLab foi garantir o gerenciamento otimizado dos

recursos computacionais alocados para a execução dos componentes, dada a oscilação de demandas de usuários aos serviços disponibilizados. Discutiu-se o uso de um gerenciador de containers que permitisse a alocação dinâmica dos recursos e o equilíbrio entre a ociosidade e o *overhead*, também conhecido por sobrecarga.

A plataforma Docker[2]. foi a solução escolhida para o gerenciamento dinâmico dos pacotes de software do sistema em unidades padronizadas chamadas container. Os *containers* são uma forma de virtualização em nível do sistema operacional, que viabilizam a execução de múltiplos componentes de software isoladamente em um mesmo host (Bernstein, 2014). O encapsulamento dos componentes do GeneMaisLab em containers permitiu uma melhor portabilidade do projeto, além facilitar a integração e a escalabilidade desses componentes.

Ao longo do texto estão descritos os procedimentos utilizados na implantação do sistema GeneMaisLab a partir da plataforma Docker. São apresentados os principais conceitos relacionados à plataforma utilizada no sistema, sua arquitetura e a distribuição de carga entre componentes do sistema com o uso do proxy reverso HAProxy[3]. Por fim, são apresentados os resultados obtidos com o uso da estratégia de containers no projeto e uma breve discussão sobre as funcionalidades da plataforma Docker.

Materiais e métodos

Há diversas estratégias para lidar com a questão de distribuição de carga entre *containers* da plataforma Docker. Uma delas é a utilização de produtos de software spara gerenciamento de *cluster* de *containers*. A forma escolhida no projeto GeneMaisLab foi o uso de um serviço de balanceamento de carga centralizado, conhecido como proxy reverso. A seguir é apresentado o sistema GeneMaisLab, seguido das principais funcionalidades da plataforma Docker utilizadas e posteriormente uma descrição de implantação e de distribuição de cargas dos containers utilizados no projeto.

Descrição do GeneMaisLab

O GeneMaisLab, referenciado como Gene+ em Germano et al. (2017), é um sistema corporativo, desenvolvido em parceria da Embrapa Arroz e Feijão e da Embrapa Informática Agropecuária, com o propósito de gerenciar dados de ensaios de genotipagem e de marcadores moleculares, desde o planejamento até as etapas de considerações finais. O sistema possibilita o registro de eventos durante a execução dos ensaios, viabiliza o rastreamento de informações registradas e permite consultas aos resultados após a finalização dos experimentos. Dentre suas funcionalidades está a comunicação

2 Disponível em: <https://www.docker.com/what-docker>.

3 Disponível em: <http://www.haproxy.org/>.

com outros sistemas da Embrapa que disponibilizam informações e serviços necessários ao GeneMaisLab, como o Alelo, Ideare, SIExp e o KEGG. Isso é feito por meio do *login* corporativo que permite a autenticação dos usuários nos demais sistemas da empresa.

As tecnologias utilizadas estão relacionadas com a linguagem Java e com alguns de seus principais frameworks e bibliotecas. Dentre os utilizados estão o Spring Boot[4], Hibernate[5], *Google Web Toolkit* (GWT)[6], GWT Bootstrap3[7] e Google Web Toolkit Project (GWTP)[8]. Todos esses componentes de softwares foram integrados pela ferramenta de automação de compilação Maven[9] (Germano et al., 2017), que permitiu agilizar o processo de testes e de implementação.

Algumas bibliotecas foram usadas para implementar funcionalidades específicas de regras de negócio do sistema, como o JasperReport, usado na geração de relatórios em vários formatos (PDF, HTML, XLS, CSV e XML). A plataforma Docker foi usada para o gerenciamento de containers.

4 Disponível em: <https://spring.io/docs/reference>

5 Disponível em: <http://hibernate.org/orm/documentation/5.1/>.

6 Disponível em: < http://www.gwtproject.org/doc/latest/DevGuide.html>.

7 Disponível em: <http://gwtbootstrap3.github.io/gwtbootstrap3-demo/apidocs/index.html>.

8 Disponível em: <http://dev.arcbees.com/gwtp/>.

9 Disponível em: < http://maven.apache.org/guides/>.

A Plataforma Docker e as funcionalidades utilizadas no GeneMaisLab

Antes de explorar as funcionalidades da plataforma Docker é necessário o entendimento dos conceitos de virtualização e de geração de *containers*. A virtualização consiste na simulação de uma plataforma de hardware, sistema operacional, dispositivo de armazenamento ou recurso de rede. A geração de *containers* é uma forma de virtualização em nível do sistema operacional, a partir de um ambiente isolado, utilizado para simulação no mesmo host (Scheepers, 2014; Seo et al. 2014; Zhang et al. 2016).

A Figura 1 ilustra um paralelo entre esses conceitos a partir das arquiteturas em camadas representadas por máquinas virtuais do inglês, *virtual machines* (VMs) e por *containers* Docker. As duas camadas inferiores, infraestrutura e sistema operacional, são as mesmas em ambas abordagens. As diferenças são observadas na camada superior. Nas VMs está a camada Hypervisor e nos *containers* Docker está o Docker Engine.

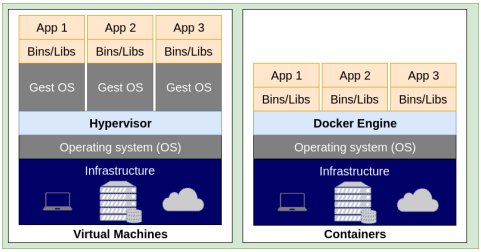

Figura 1. Arquitetura da Plataforma Docker.

O *Hypervisor* é uma camada de software responsável por fornecer ao sistema operacional hospedeiro a abstração da máquina virtual, permitindo utilizar sobre ele sistemas operacionais convidados (guest). Dessa forma, bibliotecas e aplicações que são executadas em máquinas virtuais, trabalham sobre um sistema operacional convidado. Esse conjunto é virtualizado pelo *Hypervisor* e executado no servidor hospedeiro. O *overhead* é o principal ponto explorado pela estratégia de geração de *containers*.

O *Docker Engine* é a camada de *software* que permite a criação, a execução e a implantação de aplicações com o uso de *containers*. Possibilita o isolamento do ambiente de execução das aplicações em um mesmo host, eliminando o overhead (sobrecarga). Isso ocorre porque os recursos do *kernel* (núcleo) do sistema operacional são compartilhados entre containers e gerenciados pelo *Docker Engine*.

Com o compartilhamento do *kernel* do *host*, o custo computacional de gerenciamento torna-se menor em relação ao obtido com o uso de máquinas virtuais.

Do ponto de vista da interface de programação de aplicações do inglês, *Application Programming Interface* (API) da plataforma, os *containers* são criados a partir de imagens Docker, que constituem um pacote encapsulado de um ambiente computacional. Como os dados dos containers são descartados após o seu uso, a plataforma disponibiliza um mecanismo de persistência chamado volume de dados. Esse mecanismo permite o compartilhamento de dados entre os *containers* e o *host*.

O Docker não exporta suas portas para redes externas ao *host*. Dessa forma, existe um mapeamento entre *host* e *containers*. No caso de três *containers*, cada um deles com aplicações na porta 80, deve haver um mapeamento de portas do *host* para essa porta em cada um dos *containers*. Por exemplo, a porta 8081 do *host* poderia ser mapeada para porta 80 do primeiro *container*, a 8082 para o segundo e a 8083 para o terceiro. Considerando esse cenário, caso alguém deseje acessar a aplicação da porta 80 do primeiro *container*, deverá utilizar o endereço IP do *host* com a porta 8081.

A forma de interação com o Docker pode ser feita com o Docker Client e automatizada com o uso de um arquivo de definições (Dockerfile). Esse arquivo é utilizado para preparar o ambiente a partir de um script de execução. Outro arquivo, chamado Docker Compose, pode ser usado para coordenar os *containers* e para especificar configurações do ambientes (rede, volume e serviços) usando o formato YAML[10]. O formato YAML é uma linguagem simples para a escrita de arquivos de configuração.

10 Disponível em:<http://yaml.org/>.

Implantação do GeneMaisLab usando o Docker Compose

Usando o comando scale da API do Docker Compose, é possível configurar o número de containers para execução do sistema. Dessa forma, um comando **$ docker-compose scale web=3**, permite a criação de três *containers* para o sistema. Apesar da facilidade apresentada pela API em viabilizar a escalabilidade dos containers, detalhes em nível de redes precisam ser tratados para um funcionamento adequado. Cada *container* na plataforma Docker deve possuir um IP (interno) e porta para o *host*. Esta porta pode ser determinada pelo usuário e muitas vezes apresenta conflitos.

```
$ docker-compose scale web=3

WARNING: The "web" service specifies a port on the host. If multiple containers for this
service are created on a single host, the port will clash.

Creating and starting genemaislab_web_2 ... error
Creating and starting genemaislab_web_3 ... error


ERROR: for genemaislab_web_2  Cannot start service web:
driver failed programming external connectivity on endpoint genemaislab_web_2
(d2c5d510a70fd9ff596ce8518625e5af528c9886c912c3cb49e1bdf65eaf4a51):
Bind for 0.0.0.0:80 failed: port is already allocated

ERROR: for genemaislab_web_3  Cannot start service web:
driver failed programming external connectivity on endpoint genemaislab_web_3
(d15f6d0afe5278e239dc976181b0e30aea8c53568a3094565bf3e68c4602cb6b):
Bind for 0.0.0.0:80 failed: port is already allocated
```

Figura 2. Problema ao escalar containers mapeados para uma mesma porta.

A solução utilizada pelo GeneMaisLab para lidar com o problema de acesso a uma mesma porta pelos *containers*, como ilustrado na Figura 2, consiste no uso de um componente que funciona como ponto único de acesso via porta 80, mapeando cada um dos containers e suas respectivas portas. Em outras palavras, o componente utilizado é um *proxy* reverso que encaminha cada requisição feita na porta 80 para os serviços *web*. A ferramenta utilizada para este propósito foi o HAProxy, que realiza tanto a função de *proxying* quanto de balanceador de carga. Uma arquitetura em alto nível de abstração das funcionalidades do HAProxy é ilustrada na Figura 3. Nela observa-se como a ferramenta atua no controle de fluxo entre diferentes instâncias de webservers com um banco de dados e um ponto de acesso, passando por um *firewall*. (interno).

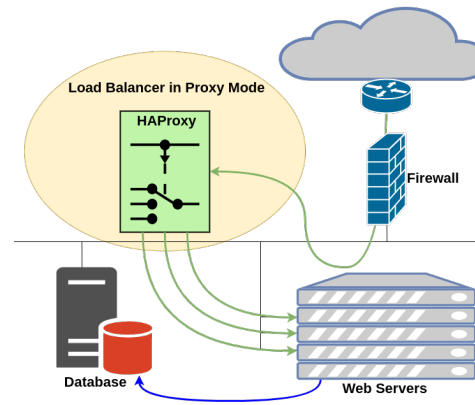

Figura 3. Arquitetura do HAProxy.

Na Figura 4 é tem-se uma arquitetura de comunicação entre componentes do GeneMaisLab e o HAProxy, que atua como elemento intermediário entre o acesso das aplicações e os serviços web (*workers*). Esses componentes são representados como serviços no *Docker Compose*, cujo o *script* YAML é apresentado na Figura 5. No script pode ser observado o serviço de banco de dados (*database*), que no caso do sistema GeneMaisLab foi usado o sistema de gerenciamento de banco de dados PostgreSQL, os

workers que constituem os serviços do GeneMaisLab acessados a partir do Spring Boot (*web*) e o balanceador de carga HAProxy (*loadbalancer*). No *script* da figura também são expressos os relacionamentos entre os serviços, os mapeamentos de diretórios do sistema hospedeiro e os *containers* (utilizando a funcionalidade “volumes” da plataforma Docker).

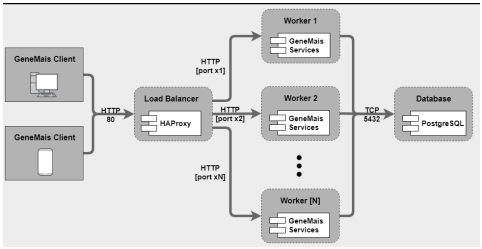

Figura 4. Componentes da arquitetura do GeneMaisLab.

```
version: '3'
services:
  database:
    # postgres 9.4 with the genemais settings
    image: eliseugermano/postgres_genemais:1.0
    ports:
      - "5435:5432"
    privileged: true
    volumes:
      - /srv/docker/postgresql/data:/var/lib/postgresql/data
  web:
    build: genemais
    ports:
      - 8080
    volumes:
      - ~/.m2:/root/.m2
    links:
      - database
    environment:
      - POSTGRES_DB=genemais
  loadbalancer:
    image: dockercloud/haproxy
    ports:
      - 80:80
    links:
      - web
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

Figura 5. Docker-compose file com os componentes do GeneMaisLab.

Resultados e discussão

Nesta seção são apresentados os resultados obtidos a partir da estruturação dos componentes da arquitetura do GeneMaisLab (Figura 4) e possíveis melhorias para a implementação atual. No exemplo de de *script* do arquivo Docker-compose do sistema, apresentado na Figura 5, observa-se o *build*, ou construção dos *conteiners*, do projeto e a implantação do sistema utilizando o comando *up*. Tanto o resultado da implantação, quanto a listagem dos containers criados após a implantação, podem ser verificados na Figura 6.

```
$ docker-compose up -d

Creating network "genemaislab_default" with the default driver
Creating genemaislab_database_1 ...
Creating genemaislab_database_1 ... done
Creating genemaislab_web_1 ...
Creating genemaislab_web_1 ... done
Creating genemaislab_loadbalancer_1 ...
Creating genemaislab_loadbalancer_1 ... done

$ docker-compose ps

        Name                      Command               State          Ports
-------------------------------------------------------------------------------------
genemaislab_database_1   docker-entrypoint.sh postgres   Up      0.0.0.0:5435->5432/tcp
genemaislab_loadbalancer_1   /sbin/tini -- dockercloud-   ...   Up   1936/tcp, 443/tcp, 0.0.0.0:80->80/tcp
genemaislab_web_1        mvn spring-boot:run             Up      0.0.0.0:32788->8080/tcp
```

Figura 6. Construindo e listando os containers após o uso de HAProxy.

Após o processo de implantação utilizando o comando **$ docker-compose scale web=3** do Docker Compose, foram acrescentados dois *containers* web. Pode-se verificar na Figura 7 que o *container* web_1 foi apenas iniciado, e que os containers web_2 e web_3 foram criados. O

Docker gera automaticamente as portas de acesso, deixando para o *container* do HAProxy (representado na Figura 6 por genemaislab_loadbalancer_1) a função de realizar o mapeamento da sua porta 80 para cada uma dos *containers web*. Dessa forma, todos os serviços web do GeneMaisLab passam pelo *genemaislab_loadbalancer_1*, que encaminha requisições para um container web menos sobrecarregado.

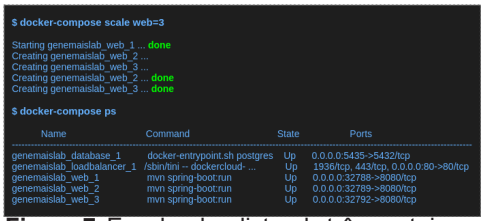

Figura 7. Escalando e listando três containers web após o uso HAProxy.

Conclusão

Com o objetivo de simplificar o processo de implantação do sistema GeneMaisLab, foi utilizada uma abordagem baseada no uso de *containers*, que permitem o encapsulamento de componentes do sistema. Com isso, problemas relacionados à portabilidade e à escalabilidade do projeto foram tratados.

O *HAProxy* foi usado para distribuição de cargas de acessos a containers que executam serviços web do GeneMaisLab. É possível verificar a divisão de tarefas realizada pelos serviços *web* a partir do sistema de *logs* da plataforma Docker e dos registros exibidos pelo Spring Boot. Durante os testes de integração do sistema, não foram observados problemas de consistência de dados após o processo de distribuição de cargas.

Embora o uso da plataforma Docker como solução para gerenciar e escalar os *containers* no GeneMaisLab seja considerada viável para o propósito do sistema, existem cenários em que a composição dos *containers* se torna muito mais complexa, sendo necessário o uso de ferramentas que viabilizem o melhor gerenciamento de *clusters* para containers, como a Kubernetes e a Swarm.

Referências

BERNSTEIN, D. Containers and cloud: from LXC to Docker to kubernetes. **IEEE Cloud Computing**, v.1, n. 3, p. 81-84, Sept. 2014. DOI: 10.1109/MCC.2014.51.

GAO, Y.; WANG, H.; HUANG, X. Applying docker swarm cluster into software defined internet of things. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY IN MEDICINE AND EDUCATION, 8., 2016, Fuzhou. **Proceedings**... Los Alamitos: IEEE, 2016. p. 445-449. DOI: 10.1109/ITME.2016.0106.

GERMANO, E.; OLIVEIRA, H.; NARCISO, M. G.; SANTOS, E.H. **Gene+**: uma solução computacional distribuída para gerenciar ensaios de genotipagem e marcadores moleculares. Campinas: Embrapa Informática Agropecuária, 2017. 10 p. il. (Embrapa informática Agropecuária. Comunicado técnico, 126).

KAUR, A.; MANN, K. S. Component based software engineering. **International Journal of Computer Applications**, v. 2, n. 1, p. 105-108, May 2010.

SCHEEPERS, M. J. Virtualization and containerization of application infrastructure: a comparison. In: TWENTE STUDENT CONFERENCE ON IT, 21., 2014, Enschede. **Proceedings**... Enschede: University of Twente, 2014. p. 1-7. Disponível em: <http://mmc.geofisica.unam.mx/acl/Herramientas/MaquinasVirtuales/VirtualizacionEnLinuxCon-Containers/539ae779eb69a.pdf>. Acesso em: 6 dez. 2018.

SEO, K.-T.; HWANG, H.-S.; MOON, Y.; KWON, O.-Y.; KIM, B.-J. Performance comparison analysis of linux container and virtual machine for building cloud. **Advanced Science and Technology Letters**, v. 66, p. 105-111, 2014. DOI: 10.14257/astl.2014.66.25.

ZHANG, J.; XIAOYI, L.; DHABALESWAR, K. P. Performance characterization of hypervisor- and container-based virtualization for HPC on SR-IOV enabled infiniband clusters. In: IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS, 2016, Chicago. **Proceedings**... Los Alamitos: IEEE, 2016. p. 1777-1784. DOI: 10.1109/IPDPSW.2016.178.

Exemplares desta edição
podem ser adquiridos na:

Embrapa Informática Agropecuária
Av. André Tosello, nº 209 Campus da
Unicamp, Barão Geraldo
CEP: 13083-886 - Campinas - SP
Fone: (19) 3211-5700
www.embrapa.br
www.embrapa.br/fale-conosco/sac

1ª edição
Versão digital (2018)

Comitê Local de Publicações
da Unidade Responsável

Presidente
Stanley R. de M. Oliveira

Secretária-Executiva
Carla Cristiane Osawa

Membros
Adriana F. Gonzalez, Carla Geovana do N. Macário, Flávia B. Fiorini, Jayme Barbedo, Kleber X. Sampaio de Souza, Luiz Antonio Falaguasta Barbosa, Maria Goretti G. Praxedes, Paula Regina K. Falcão, Ricardo Augusto Dante, Sônia Ternes

Suplentes
Michel Yamagushi e Goran Nesic

Supervisão editorial
Kleber X. Sampaio de Souza

Revisão de texto
Carla Geovana N. Macário

Normalização bibliográfica
Maria Goretti G. Praxedes

Projeto gráfico da coleção
Carlos Eduardo Felice Barbeiro

Editoração eletrônica
Júlio César dos S. Souza, sob supervisão de Flávia B Fiorini

Foto da capa
Pexels

CGPE 14934