# Uso de Docker para Desenvolvimento de Aplicações: Um estudo de caso com GEs FIware

Welkson Renny de Medeiros, Gabriela Cavalcante da Silva, Carlos Eduardo da Silva
*Instituto Metrópole Digital*
*Universidade Federal do Rio Grande do Norte*
*Natal, RN, Brasil*
*Email: welkson@gmail.com, gabicavalcantesilva@gmail.com, kaduardo@imd.ufrn.br*

***Resumo*—A plataforma FIware oferece um conjunto de componentes que podem ser reutilizados para o desenvolvimento de aplicações para cidades inteligentes. Entretanto, a integração desses componentes em um ambiente para desenvolvimento, e produção, não é uma tarefa trivial. Este artigo apresenta um estudo sobre mecanismos de gerenciamento de containers Docker, e seu uso para a implantação de ambientes de desenvolvimento baseado em componentes FIware integrados. Como estudo de caso, foram utilizados containers Docker para a implantação da infraestrutura de suporte para uma aplicação sendo desenvolvida no âmbito do projeto Smart Metropolis.**

***Keywords*-FIware; Generic Enablers; Docker; Docker-compose; Containers;**

## I. Introdução

O desenvolvimento de aplicações distribuídas tem diversos desafios, tais como gerenciar a configuração dos componentes envolvidos, a comunicação entre eles, e principalmente, a abordagem mais eficiente para a implantação da solução desenvolvida em um ambiente de produção. No âmbito de aplicações para cidades inteligentes, uma das principais plataformas é a FIware[1], que disponibiliza diversos componentes (*Generic Enablers* GE) que podem ser reutilizados para o desenvolvimento de novas aplicações.

Nesse contexto, este trabalho se propõe a demonstrar como desenvolver soluções baseada em GEs Fiware utilizando-se de uma plataforma de gerenciamento de containers. Como estudo de caso, descrevemos como containers foram utilizados para gerenciar a infraestrutura de uma aplicação sendo atualmente desenvolvida pelo projeto Smart Metropolis.

## II. Aplicações em containers Docker

O Docker[2] é uma plataforma que automatiza a implantação de aplicações dentro de ambientes isolados denominados *containers*, de maneira semelhantes a máquinas virtuais. Entretanto, se apresenta como uma solução mais "leve" por utilizar o mesmo kernel para todas as aplicações. O principal objetivo dessa tecnologia é proporcionar o empacotamento de aplicações com todas as suas dependências (bibliotecas do sistema operacional, ferramentas, etc.) em uma unidade padronizada para desenvolvimento de software, com isolamento a nível de disco, memória e processamento.

Dependendo da arquitetura da aplicação, a mesma pode ser desenvolvida em um ou múltiplos containers que se comunicam entre si. Gerenciar o ciclo de vida de um elevado número de containers manualmente se torna inviável, pois vários fatores terão que ser considerados, tais como ordem em que os mesmos serão iniciados (ex.: a aplicação não pode ser iniciada antes do banco), integração entre containers (ex.: o container do IDM deve ter acesso via IP do PEP, etc.), recriar os containers quando estes forem alterados, etc.

O projeto Docker disponibiliza uma ferramenta open-source denominada Docker-Compose, que permite gerenciar múltiplos containers. Os containers e seus respectivos serviços, portas, links e demais particularidades são definidos no arquivo de configurações declarativas *docker-compose.yml*, e usando o comando *docker-compose* é possível gerenciar de forma integrada o processo de build e execução dos containers.

## III. Arquitetura do SGeOL

A Smart Geo Layers é uma plataforma para visualização, edição e análise de informações multicamadas representadas por dados geoespaciais. Através da Smart Geo Layers os usuários interessados podem efetuar o confrontamento de dados e criar suas próprias camadas. Esta aplicação se baseia em um conjunto de GEs Fiware para diversas funcionalidades, e portanto necessita de um ambiente de desenvolvimento com essas GEs integradas.

A Figura 1 apresenta a arquitetura do SGEoL, destacando a integração entre os diversos componentes do FIware, disponibilizados no formato de GE's. Os componentes sombreados em cinza correspondem aos GEs que serão implantados. Para atender aos requisitos de segurança a aplicação utiliza os GEs Fiware Keyrock e PEP Proxy. Os demais GEs são relacionados a captura de dados de sensores IoT (IOT GEs), informações de contexto (Orion Context Broker), armazenamento e visualização de dados (CKan), e desenvolvimento de front-ends Web (WireCloud).

[1]http://www.fiware.org
[2]https://docs.docker.com

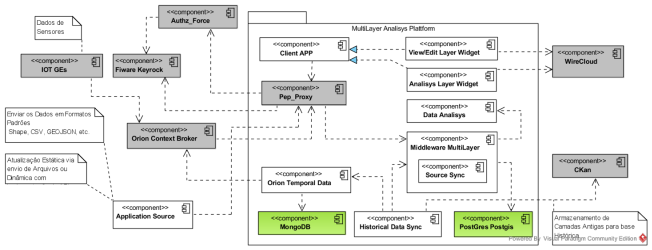


Figura 1. Arquitetura do SGEoL.

Cada GE disponibiliza imagens Docker que podem ser utilizadas para testes e desenvolvimento. Entretanto, tais GEs são configurados para funcionarem de maneira independente uns dos outros. Dessa forma, é necessário realizar algumas modificações para permitir seu uso de maneira integrada.

Para integrar os GE's envolvidos no projeto SGeOL, utilizamos o Docker-Compose. Foi criado um arquivo **docker-compose.yml** especificando detalhes sobre cada serviço, tais como nome do serviço, local da imagem no Docker Hub, portas TCP utilizadas, link's entre outros GE's, entre outras opções de configuração.

Entretanto, encontramos uma dificuldade na integração entre os componentes PEP Proxy e Keyrock. Esta dificuldade está relacionada com o *config.js*, um arquivo de configurações que o PEP Proxy requer contendo o endereço IP do Keyrock e da aplicação back-end, além das credencias do PEP, etc. Esse arquivo pode necessitar ser atualizado a cada inicialização do container onde o GE do IdM está sendo executado, pois quando os containers são iniciados, é possível que os IP's associados a eles mudem, obrigando essa constante edição do arquivo de configuração.

Para resolver esse problema, optamos por integrar os containers através do parâmetro *link* no **docker-compose.yml**. Dessa forma, o Docker passa a resolver o IP do container onde o Keyrock está sendo executado a partir do nome do container, como mostramos abaixo no trecho do **docker-compose.yml** do projeto SGeOL.

```
keyrock:
    image: fiware/idm:v5.3.0
    hostname: keyrock
    container_name: keyrock
    expose:
        - "5000"
        - "8000"
    ports:
        - "5000:5000"
        - "8000:8000"

pepproxy:
    build: Docker/fiware-pep-proxy
    hostname: pepproxy
    container_name: pepproxy
    expose:
        - "80"
    links:
        - keyrock
```

Procedimento 1. Trecho do Docker-Compose.yml envolvendo PEP e IDM

Para que essa atualização automática de IP também seja refletida no arquivo *config.js*, decidimos criar uma imagem personalizada do PEP Proxy, que utiliza a imagem ging/fiware-pep-proxy como referência, e a estende novas funcionalidades, tais como a possibilidade de utilizar um *config.js* com parâmetros especificos do nosso projeto. Essa solução também facilitará futuras modificações na imagem do PEP Proxy, por exemplo, para ele se adeque a novos requisitos do projeto.

No *docker-compose.yml* essa alteração é refletida no uso da instrução *build* em vez de *image*. A instrução *image* informa ao Docker que o mesmo deve procurar essa imagem no Docker Hub, um repositório público de imagens, já no caso da instrução *build* o Docker vai procurar em um diretório dentro do próprio projeto, que nesse caso é o diretório *Docker/fiware-pep-proxy*. Neste diretório há um arquivo *Dockerfile*, que faz referência a imagem do PEP do DockerHub e ao arquivo *config.js*, previamente configurado no diretório do PEP-Proxy.

Durante a evolução do projeto outras imagens podem necessitar de customizações, e para facilitar esse gerenciamento optamos por criar um sub-diretório no repositório SGeOL-Docker denominado *Docker*, e dentro dele outros sub-diretórios para cada imagem a ser personalizada. Esses sub-diretórios das imagens contém um arquivo *Dockerfile* com a especificação da imagem.

O repositório do projeto SGeOL-Docker está disponível no GitLab do IMD[3].

## IV. Conclusão

Neste trabalho apresentamos um estudo sobre a utilização de containers Docker para integração de GE's FIware. Foram desenvolvidos scripts com Docker-compose para integrar os GE's do projeto SmartGeo Layers (SGeoL) como estudo de caso.

Os conhecimentos adquiridos auxiliarão nas próximas etapas, onde serão estudados meios de disponibilizar esses GEs em uma infraestrutura que considera aspectos de alta-disponibilidade e escalabilidade. Para isso, pretendemos realizar um estudo em clusters Kubernetes[4] ou Docker Swarm[5].

[3]http://projetos.imd.ufrn.br/SmartMetropolis-InfraestruturaGroup/SGeoL-Docker

[4]http://kubernetes.io/docs/

[5]https://docs.docker.com/swarm/