
Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

*SmartMetropolis* – Plataforma e Aplicações para Cidades Inteligentes

WP4 – Infraestrutura

# Relatório de Atividade Quarto Trimestre: Desenvolvimento, implantação e operação de infraestrutura computacional

Natal-RN, Brasil
Janeiro 2017

## Equipe Técnica

*Docentes*

Prof. Dr. Carlos Eduardo da Silva (Coordenador) – IMD/UFRN

*Discentes*
Gabriela Cavalcante da Silva
Rafael Varela
Welkson Renny de Medeiros

*Pesquisadores vinculados*
Thomas Filipe da Silva Diniz - Anolis TI

# Sumário

# Lista de Figuras

# 1 Introdução

O projeto Smart Metropolis, conduzido pelo Instituto Metrópole Digital (IMD) da Universidade Federal do Rio Grande do Norte (UFRN), tem como objetivo o desenvolvimento de soluções de tecnologia da informação e comunicação para Cidades Inteligentes e Humanas. O projeto é organizado em seis Pacotes de Trabalho (Work Packages - WP) temáticos, liderados cada por um coordenador. Cada WP possui um conjunto de objetivos a serem alcançados ao longo dos cinco anos de execução do projeto. Este relatório está inserido no contexto do WP 4 - Infraestrutura Computacional, e visa relatar as atividades desenvolvidas no âmbito da tarefa de **Desenvolvimento, implantação e operação de infraestrutura computacional**.

Esta tarefa tem como objetivo aspectos relacionados a formação de mão de obra capacitada para operar a infraestrutura computacional do projeto, assim como todos os aspectos relacionados à implantação de uma instância da plataforma FIWARE a nível de produção no ambiente computacional do IMD. A equipe atuante nesta tarefa é formada por: Prof. Dr. Carlos Eduardo da Silva (Coordenador da tarefa), Welkson de Medeiros (Bolsista Mestrado), Gabriela Cavalcante da Silva (Bolsista Graduação) Rafael Varela (Bolsista Graduação). Além disso, a equipe recebe o apoio da empresa incubada Anolis TI através de Thomas Filipe da Silva Diniz.

Após considerações sobre os resultados alcançados no terceiro trimestre do projeto foram elencadas as seguintes atividades para o quarto trimestre:

- Implantação da instância FIWARE de produção

  A implantação de uma instância FIWARE operacional para uso pelos membros do projeto, foi dividida em duas etapas: implantação de nuvem "OpenStack", e implantação de GEs FIWARE. No terceiro trimestre a primeira etapa foi realizada, deixando para o quarto trimestre do projeto a operação da nuvem em caráter experimental e a produção de manual técnico detalhando todo o processo de implantação da instância FIWARE do projeto.

  A segunda etapa do processo de implantação tem como objetivo o estudo e a implantação de algumas GEs (Keyrock, AuthZForce e Wilma) em um ambiente de produção. Durante o quarto trimestre, as atividades se concentraram na utilização dessas GEs de forma a fornecer subsídios para os outros WPs, permitindo o apoio ao desenvolvimento da aplicação SGEOL. Além disso, foram implantadas também as GEs que dão suporte à funcionalidades de IoT.

- Estudo sobre ambientes de gerenciamento de *containers* "Kubernetes"

  Durante o terceiro trimestre foi realizado um estudo experimental sobre ambientes de gerenciamento de *containers* usando a plataforma Docker. Tal estudo teve o objetivo de fornecer subsídios para a implantação de GEs FIWARE. Para o quarto trimestre, foi realizado um estudo preliminar sobre o ambiente de gerenciamento de *containers* kubernetes.

- Resultados da Implantação de Ambiente de Desenvolvimento para SGeoL

  A Implantação de Infraestrutura de suporte para a Aplicação SGeoL visa atender a uma demanda dos WPs de Aplicação e *Middleware*, que iniciaram o desenvolvimento de uma aplicação para a prefeitura municipal do Natal (Smart Geo Layers - SGeoL).

  Como atividade do quarto trimestre tivemos o acompanhamento da equipe de desenvolvimento do SGeoL no uso do ambiente, na expectativa de fornecer subsídios que possam ser incorporados à implantação e operação do ambiente de produção.

O restante deste documento está organizado da seguinte forma: Seção 2 apresenta o um relato acerca das atividades relacionadas à implantação da instância FIWARE do projeto, incluindo implantação da nuvem OpenStack, e implantação de algumas GEs. Seção 3 apresenta o estudo sobre o ambiente de gerenciamento de *containers* kubernetes. Seção 4 apresenta um relato sobre a implantação do ambiente de desenvolvimento para o projeto Smart Geo Layers. As considerações finais são apresentadas na Seção 5.

# 2 Implantação de instância FIWARE de Produção

Como mencionado anteriormente, a implantação de uma instância FIWARE para ser utilizada pelos membros do projeto foi dividida em duas etapas: implantação de nuvem OpenStack, e implantação de GEs. Neste contexto, esta seção descreve as atividades que foram conduzidas durante o quarto trimestre do projeto relacionadas à ambas as etapas.

No que diz respeito à implantação da nuvem OpenStack que dará apoio à instância FIWARE do projeto, o quarto trimestre foi dedicado à sua operação, e à produção de manual técnico detalhando todo o processo de implantação. A segunda etapa do processo de implantação teve como objetivo a implantação e uso de algumas GEs em um ambiente de produção.

Para o quarto trimestre do projeto o foco foi na utilização das GEs relacionadas aos mecanismos de segurança da plataforma FIWARE, uma vez que essas GEs serão utilizadas por qualquer aplicação que venha a ser desenvolvida. Além disso, a experiência com as GEs de segurança visa fornecer subsídios para o desenvolvimento da aplicação SGEOL.

Ainda como parte da segunda etapa de implantação, foi iniciada a implantação das GEs relacionadas a IoT (*Internet of Things*), outro conjunto de GEs com alto potencial de uso por parte de outras aplicações.

## 2.1 Implantação de Nuvem OpenStack

Ao longo do quarto trimestre, a nuvem foi utilizada por um conjunto reduzido de usuários e como infraestrutura onde o ambiente de desenvolvimento do SGEOL foi implantado. Esta experiência permitiu à equipe a realização de alguns ajustes em sua infraestrutura, decorrente das demandas recebidas. Além disso, alguns contratempos encontrados com os equipamentos cedidos pelo *datacenter* do IMD não permitiram a implantação da nuvem de acordo com o projeto definido anteriormente. Com isso, foram feitas algumas realocações de serviços OpenStack entre servidores físicos.

O manual de implantação da nuvem visa fornecer subsídios para que a equipe de TI do IMD tenha condições de colaborar na operação e manutenção da infraestrutura, além de constituir documentação técnica importantíssima para os membros da equipe que por ventura venham a desempenhar tarefas de implantação, operação e manutenção de uma nuvem OpenStack. O manual apresenta o projeto de implantação detalhado da nuvem, assim como os ajustes realizados em sua arquitetura devido à falta de recursos. O manual se encontra em documento a parte, e por ter sido desenhado para uso interno pela equipe do projeto e de TI do IMD, não será divulgado.

## 2.2 Experiência de Uso das GEs de Segurança

O Keyrock é a implementação de referência do Identity Management GE (IdM), sendo responsável pelo gerenciamento de identidades de usuários, organizações e aplicações, assim como de suas respectivas credenciais, e da autenticação dessas entidades. Por outro lado, o AuthZForce é a implementação de referência do *Authorization* PDP GE (PDP - *Policy Decision Point*), sendo responsável por gerenciar

políticas de autorização na plataforma FIWARE. O PEP Proxy Wilma funciona como um *proxy* transparente que fica na frente dos serviços REST para garantir a proteção. O Wilma intercepta as requisições dos clientes, realizando sua autenticação junto ao IDM KeyRock, e autorização junto ao PDP (AuthZForce). Os desenvolvedores interagem com o IDM KeyRock para registrarem suas aplicações no IdM, e para gerenciar a segurança de suas aplicações (credenciais, papéis e políticas de autorização). As políticas definidas pelos desenvolvedores por meio do IdM são utilizadas pelo PDP para avaliar requisições de acesso, garantindo que somente usuários com as devidas permissões tenham acesso aos recursos protegidos.

Devido à dependência entre esses componentes, as atividades do quarto trimestre focaram em sua utilização em conjunto. Nossa prioridade foi finalizar o desenvolvimento das aplicações de exemplo que demonstrem o funcionamento entre o IdM Keyrock, Pep *Proxy* e AuthZForce.

Durante a finalização das aplicações, nos deparamos com alguns problemas. O primeiro deles foi encontrado quando estávamos implementando a autorização com o PDP, e surgiu uma notificação feita pelo AuthZForce sobre a falta de domínio para a aplicação que estávamos tentando acessar. Para tentar resolver o problema, criamos um domínio pelo AuthZForce, porém não obtivemos sucesso para conectar esse domínio a aplicação.

Foi quando encontramos no *Admin Guide* do FIWARE IDM[1] uma seção sobre a configuração para conectar o FIWARE IDM ao AuthZforce PDP GE, para poder criar as permissões para a aplicação. Para configurar essa conexão foi necessário editar o arquivo de configuração *openstack_dashboard/local/local_settings.py*, localizado no módulo do Horizon. O seguinte trecho foi editado:

```
ACCESS_CONTROL_URL = 'http://192.168.99.100:8080'
ACCESS_CONTROL_MAGIC_KEY = 'undefined'
```

Após solucionamos esse erro, outro problema foi detectado no *log* do PEP:

```
pep-proxy_1  | 2016-12-14 16:49:45.366  - ERROR: Root - User access-token not
    authorized:  User not authorized in AZF for the given action and resource
```

Esse erro acontecia mesmo enviando um *token* válido obtido através Keyrock. Para verificar o que poderia estar ocorrendo, realizamos um debug detalhado do AuthZforce e verificamos que tudo estava funcionando corretamente, ele retornava para o PEP a resposta *Permit*, o que é esperado caso o usuário tenha acesso ao recurso solicitado. Podemos ver isso no *log* abaixo:

```
pep-proxy_1 | 2016-12-16 21:16:31.699  - DEBUG: AZF-Client - Decision:  [ 'Permit' ]
pep-proxy_1 | 2016-12-16 21:16:31.699  - ERROR: Root - User access-token not
    authorized:  User not authorized in AZF for the given action and resource
```

Como podemos ver, recebemos o *Permit* como conteúdo da resposta do AuthZForce, mas ainda assim o PEP não liberava o acesso ao recurso solicitado. Com isso, passamos para os arquivos do PEP, e encontramos no arquivo *lib/azf.js* o trecho:

```
log.debug('Decision: ', decision);
if (decision === 'Permit') {
  success();
} else {
  error(401, 'User not authorized in AZF for the given action and resource');
}
```

Nesse trecho o código compara a variável *decision*, obtida como resposta do AuthZForce, com a string *Permit*. De acordo com o log do AuthZForce, o resultado da comparação deveria ser *True*, foi então que

[1] http://fiware-idm.readthedocs.io/en/latest/admin_guide.html#authzforce-ge-configuration

fomos verificar o tipo da variável *decision* e vimos que era *object*. Porém, para o javascript quando === é usado, o tipo *object* é diferente do tipo *string*, por isso o resultado da comparação seria sempre *False*, mesmo que o conteúdo fosse igual. Para testar, mudamos o === para ==, e tudo funcionou corretamente. Outra solução é fazer um parser antes da comparação, para converter a variável *object* para *string*.

Depois de encontrarmos a solução para o erro recebido, decidimos criar uma *issue*[2] no repositório oficial do PEP, para que o problema fosse corrigido no código fonte.

## 2.3 Implantação de GEs de IoT

Ao final do trimestre, foi iniciada a implantação das GEs de IoT do FIWARE. Esta seção apresenta um relato desta experiência até o momento, onde foi adotada uma perspectiva de implantação de uma infraestrutura local com o intuito de disponibilizar recursos necessários para as aplicações e/ou coisas que se conectarão em nossa rede.

### 2.3.1 Os componentes

Segundo a documentação do FIWARE as GEs relacionadas a IoT são: "Backend Device Management", "Backend Configuration Manager"(IoT Discovery) e "IoT Broker". O propósito destes componentes é permitir o acesso a coisas e dispositivos através de uma interface unificada que é fornecida pelo GE "Orion Context Broker", disponibilizado pelo capítulo técnico de dados do FIWARE. Com isso, é possível identificar diversos cenários de implantação para os GEs de IoT que visam atender a diversas demandas de uso. Iniciamos apresentando uma breve descrição sobre cada componente, antes de apresentar os cenários de implantação.

O Backend Device Management (IDAS) realiza a conexão entre dispositivos (sensor de umidade, temperatura e entre outros) ao FIWARE-based ecosystems (ambiente de IoT baseado no FIWARE onde as coisas estarão conectadas), traduzindo protocolos específicos de IoT para o protocolo de informação NGSI, que é o modelo de troca padrão de dados do FIWARE. Para o IDAS, trabalhar com protocolos específicos de IoT só é possível, por causa dos agentes, que são componentes responsáveis por fazer a comunicação entre um sensor e o Backend Device Management utilizando o protocolo específico de qualquer sensor. Por exemplo: se um sensor de temperatura trabalhar com o protocolo MQTT, logo o IDAS deverá ter um agente responsável para se comunicar usando o protocolo MQTT.

O IDAS faz com que seus dispositivos sejam representados em uma plataforma FIWARE como entidades NGSI em um Context Broker, isto é, você pode consultar por mudanças no status dos parâmetros do dispositivo, consultando aos atributos da entidade correspondente da NGSI presentes no Context Broker. Além disso, ele possibilita acionar comandos para atuação nos dispositivos apenas atualizando atributos específicos nas entidades NGSI representadas no Context Broker. Desta maneira, as interações dos desenvolvedores com seus dispositivos são suportadas pelo Context Broker, proporcionando uma API homogênea e uma interface como para todos os outros dados não-IoT em um FIWARE *ecosystem*.

O Backend Configuration Manager (IoT Discovery) é um Habilitador Genérico responsável por gerenciar e facilitar a descoberta e disponibilidade de dispositivos e coisas IoT. Assim como outros Habilitadores Genéricos oferecidos pelo FIWARE, o IoT Discovery utiliza o conceito de entidades de contexto, que são entidades genéricas descritas por meio de atributos e metadados.

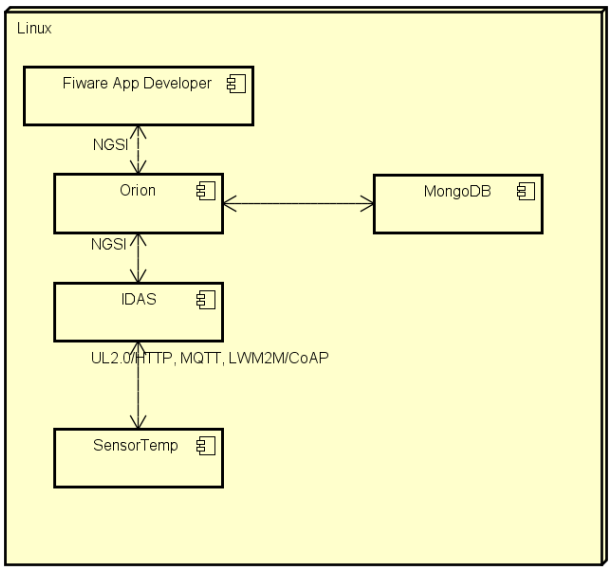
O IoT Broker atua como um mediador entre os dispositivos de IoT e os usuários que consomem suas informações. Em conjunto com o IoT Discovery, o IoT Broker fornece funcionalidades de composição de dispositivos. Ele permite que os usuários façam consultas por dados provenientes de dispositivos de

[2] https://github.com/ging/fiware-pep-proxy/issues/36

IoT sem que os mesmos precisem necessariamente conhecer quais dispositivos são capazes de prover os dados requisitados, a forma de acesso a esses dispositivos ou onde eles estão localizados. Todo o processo de aquisição dos dados é abstraído pelo IoT Broker. Além disso, o IoT Broker suporta também a realização de consultas assíncronas a dados de IoT (representados por meio de entidades de contexto) através de mecanismos de processamento de eventos complexos do FIWARE. Desse modo, um usuário pode ser notificado sempre que um atributo de uma ou mais entidades de contexto for alterado, quando um atributo ultrapassar determinado valor, quando o mesmo estiver dentro de determinado intervalo, etc.

Como mencionado anteriormente, esses componentes podem ser implantados em diversos cenários. Entretanto, de acordo com a documentação do FIWARE, somente um dos cenários está disponível para uso no FIWARE Lab, e portanto decidimos focar neste primeiro cenário. A Figura 1 apresenta uma visão geral do cenário mais simples, onde dispositivos são registrados junto ao Orion Context Broker e disponibilizados para as aplicações. Neste cenário, o IDAS realiza o papel de tradutor de protocolos entre o protocolo utilizado pelo dispositivo e o protocolo NGSI utilizado pelo FIWARE. O MongoDB é utilizado pelo Orion para armazenamento dos últimos valores lidos dos dispositivos.

Figura 1: Arquitetura Simples




### 2.3.2 Experiência com os GEs de IoT

Na sequência apresentamos um relato da experiência de implantação e utilização dos GEs de IoT do FIWARE.

A experiência de implantação dos GEs de IoT teve como alvo o cenário mais simples de utilização dos GEs de IoT (apresentado na Figura 1). De acordo com experiências anteriores, iniciamos a implantação utilizando *containers* Docker. Entretanto, encontramos diversos repositórios que disponibilizavam uma estrutura contendo Orion, IDAS e Agentes UL2.0. As duas primeiras tentativas não tiveram sucesso, envolvendo diversos erros com dependências entre pacotes e versões de software, bibliotecas e sistemas operacionais. Conseguimos implantar o ambiente a partir da personalização de uma descrição Docker de uma aplicação de exemplo do FIWARE.

Para efetuar testes na infraestrutura de IoT foi usado um cliente em Python, recomendado pela documentação do FIWARE. Este cliente foi executado em um máquina virtual visando simular um sensor de temperatura.

Após instalação da máquina virtual, feito a clonagem do repositório que contém os códigos em Python para que os arquivos de configuração e *scripts* de testes ficassem dentro da VM que simularia o sensor cliente, sendo assim foi executado o comando:

```
$ git clone https://github.com/telefonicaid/fiware-figway.git
```

Feito o clone do repositório, o próximo passo foi fazer a configuração do cliente, bastando para isso editar o arquivo *config.ini* situado no diretório fiware-figway/python-IDAS4. Este arquivo é configurado de acordo com o ambiente em teste, onde são passados dados relacionados a autenticação (se utilizada), endereços e portas dos GEs (Orion e IDAS), além de um identificador para a coisa sendo simulada.

Depois do cliente configurado, foram feitos alguns testes usando os scripts fornecidos pela aplicação de teste. Os scripts estão no diretório fiware-figway/python-IDAS4/Sensors_UL20. As operações de registro de novo sensor e listagem dos sensores registrados foram executadas com sucesso. Entretanto, encontramos dificuldades no envio de informações (simulando o envio de uma leitura pelo sensor). A saída obtida é apresentada em seguida.

```
$  * Asking to http://10.7.173.138:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=Sensor1
* Headers: {'Fiware-Service': 'fiwareiot', 'content-type': 'application/json', '
    Fiware-ServicePath': '/', 'X-Auth-Token': 'NULL'}
* Sending PAYLOAD:
t|34

...

* Status Code: 400
* Response:
{"name":"UNSUPPORTED_TYPE","message":"The request content didn't have the expected
    type [text/plain ]"}
```

Entretanto, testes com os componentes disponibilizados pelo FILAB funcionaram com sucesso. Desse modo, sugerimos que seja realizado um estudo mais aprofundado acerca dos componentes de IoT do FIWARE, com o objetivo de se implantar um ambiente de IoT local.

Durante o procedimento de estudo, implantação e testes de uma infraestrutura local de IoT baseada em FIWARE notou-se que a documentação ainda deixa bastante a desejar. Além disso, a existência de diversos repositórios, com configurações conflitantes, fez com que um tempo considerável fosse consumido nesta tarefa. De todos os componentes o IDAS mostrou ser o componente mais complexo para achar o *containers* adequado, visto que em várias tentativas de uso do IDAS os *containers* tinham problemas no agente, na *adminport* e problemas com a versão do Docker. Foram realizados também alguns testes preliminares envolvendo a manipulação de sensores com o IoT Broker, entretanto, devido aos problemas encontrados com a aplicação cliente, os mesmos não puderam ser concluídos.

## 2.4 Discussão

O principal resultado alcançado neste quarto trimestre do projeto corresponde à instância de nuvem OpenStack implantada e operacional. Entretanto, devido à limitação de recursos, não foi possível a implantação de um ambiente com alta disponibilidade. Isso acarretou uma alteração em seu projeto de implantação de modo a permitir o início de sua operação com uma quantidade de recursos reduzida, embora a arquitetura implantada esteja preparada para o seu crescimento com a adição de recursos de hardware. A nuvem se encontra atualmente em uso por um conjunto reduzido de usuários, de forma que sugerimos o aumento da base de usuários de maneira controlada, em conjunto com o seu monitoramento para avaliar a utilização de recursos. Além disso, baseado nas experiências alcançadas com o processo

de implantação da nuvem, indicamos como próximos passos o estudo e implantação das ferramentas de monitoramento de uma infraestrutura FIWARE disponibilizadas pelo seu capítulo técnico FIWARE OPS Tools.

Com relação a implantação das GEs em ambiente de produção, notamos com o resultado que houve uma parcela considerável de tempo dedicado a correções de erros, quer seja por uma dificuldade de encontrar de forma clara e acessível informações na documentação, ou por erro no próprio codigo fonte da GE. Por outro lado, esses entraves contribuíram para que tivessem um conhecimento mais profundo do código fonte e do funcionamento das GEs.

Em relação às GEs de segurança, foram finalizados os exemplos de autenticação que estávamos implementando: autenticação básica com AuthZForce, autenticação com PEP *Proxy*, e autorização com PDP. Esta última aplicação era a que estava pendente até resolvermos o bug do PEP. A finalização desses exemplos permitiu que tivéssemos toda a estrutura para dar encaminhamento a autenticação no projeto Smart Geo Layers. Como próximos passos, temos o estudo mais aprofundado em mecanismos de segurança e controle de acesso (autenticação e autorização) e o desenvolvimento de um projeto detalhado de autenticação e autorização para o SGEOL.

Em relação às GEs de IoT, foram encontradas diversas dificuldades em sua implantação. Entretanto, o estudo com a implantação destas GEs foram iniciadas já ao final do quarto trimestre do projeto. Desse modo, recomendamos que se defina qual o cenário de uso dos componentes de IoT pelo projeto antes que o estudo seja retomado.

# 3 Estudo sobre ambientes de gerenciamento de *containers* Kubernetes

Kubernetes é uma plataforma *open-source* de gerenciamento de *containers* em *cluster*, permitindo assim que as aplicações sejam gerenciadas através de múltiplos *hosts*, provendo mecanismos de implantação, manutenção e escalabilidade. Seu desenvolvimento foi iniciado em 2014 pelo Google, baseado em uma década de experiência no uso de *containers* em *clusters* para disponibilização de serviços em ambientes de produção [1]. O Google usa o Kubernetes como plataforma de gerenciamento de *containers* (*Google Container Engine*) no produto *Google Cloud Platform* [3].

Neste contexto, e com vistas a se definir um ambiente para gerenciamento de *container* em *cluster* para a infraestrutura do projeto, esta seção apresenta um estudo preliminar acerca da solução kubernetes.

A arquitetura e os componentes do Kubernetes serão apresentados nas próximas seções.

## 3.1 Arquitetura

Um *cluster* Kubernetes é formado por um nó *master* onde são disponibilizados os serviços de gerenciamento do *cluster* (autenticação, autorização, agendamento, controlador, distribuição das configurações entre os nós do *cluster* via ETCD, etc.), e os nós onde são executados os *containers* conforme apresentado na Figura 2. Os nós podem ser máquinas físicas ou virtuais.

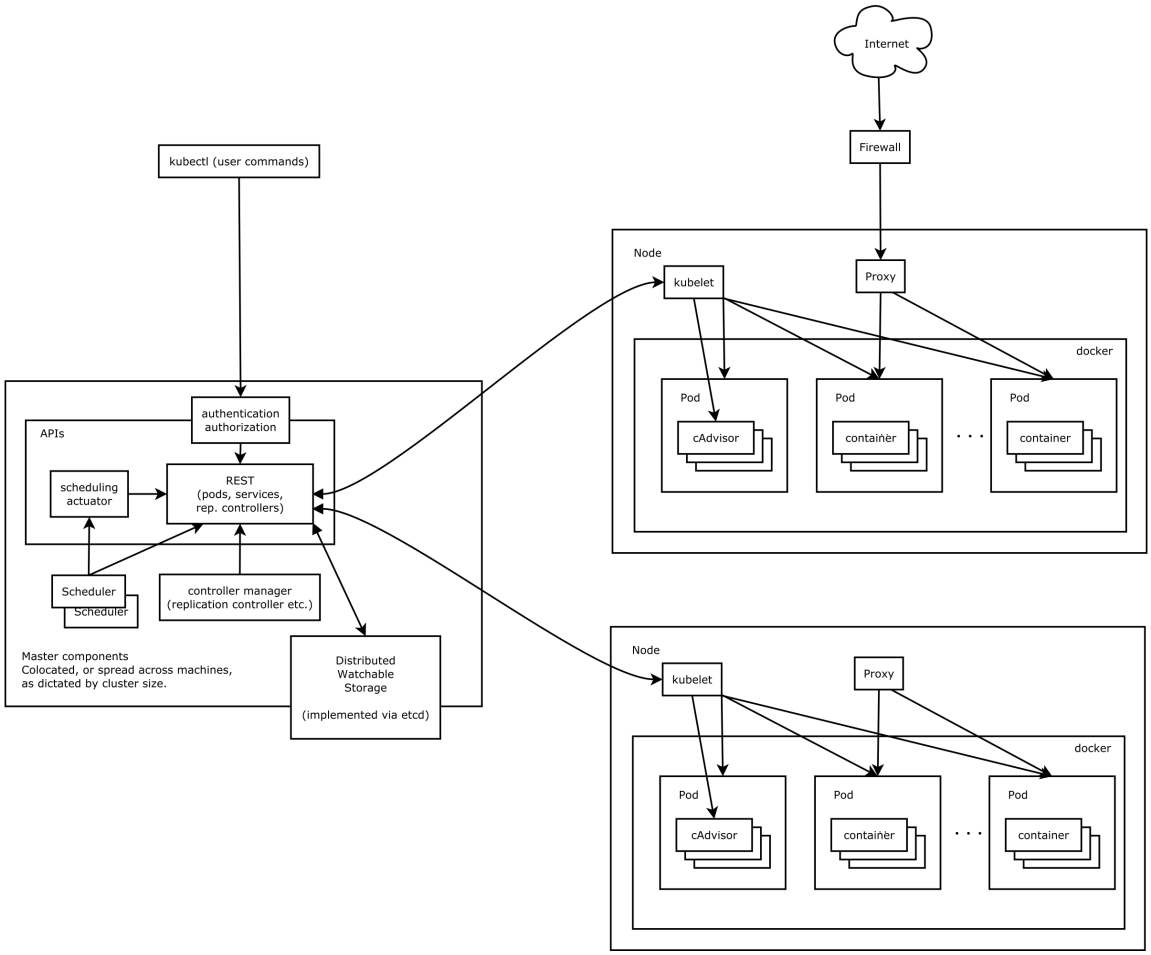
Para que a comunicação entre os *Nodes* e o *Master* seja possível são necessários alguns serviços que são disponibilizados por componentes. A seguir serão apresentados os principais componentes do *Master* e dos *Nodes*.

Os principais componentes do servidor *Master* são:

- Etcd: Sistema que armazena dados (no caso do Kubernetes o ETCD armazena as configurações dos nós) no formato chave-valor e distribui entre múltiplos nós dentro do *cluster*;

[3] https://cloud.google.com/container-engine/

Figura 2: Arquitetura do Kubernetes




- API *Server*: Implementa uma interface RESTful para que os clientes possam ajustar a carga de trabalho do *cluster*;
- *Controller Manager Service*: O *Controller Manager Service* é responsável por regular o estado do *cluster* e executar tarefas de rotina, tais como verificar se o número de réplicas definidas para o serviço está igual ao número de réplicas que estão em execução;
- *Scheduler Service*: Verifica a carga dos nós para que seja possível distribuir os *containers* dentro do *cluster*, evitando assim que um nó tenha excesso de recursos em uso enquanto outro está sub-utilizado;

Nos *Nodes* os componentes principais são:

- Docker: O Docker é utilizado para encapsular as aplicações em *containers*, de forma que as mesmas ficam isoladas das demais, sendo o principal componente em execução nos nós;
- *Kubelet Service*: É o serviço no nó responsável por interagir com o ETCD (configuração em formato de chave-valor) e com o *control plane*, onde recebe os comandos e demais configurações que devem ser aplicadas no nó;
- *Proxy Service*: É responsável por disponibilizar os serviços para acesso externo, e encaminhar as requisições para os *containers* corretos;

Um cluster Kubernetes é formado por unidades de trabalho (*Work Unit*), elementos básicos para se implantar uma aplicação com Kubernetes. As principais unidades de trabalho são o *Pod*, *Service* e o *Replication Controller*, e serão apresentados a seguir.

Um *pod* geralmente representa um ou mais *containers* que devem ser controlados como uma única "aplicação". Os containers de um *pod* são gerenciados como uma unidade, compartilhando um mesmo ambiente, volumes e espaço de IP. De forma geral, o design de um *pod* consiste em um *container* principal, e alguns *containers* auxiliares que facilitam tarefas relacionadas. Cada programa é executado e gerenciado por seu próprio *container*, mas são fortemente ligados a aplicação principal.

Para o Kubernetes, um *service* é uma abstração que define um conjunto lógico de *pods* e uma política para acesso de cada um. Isso permite fazer o *deploy* de uma unidade de serviço que está ciente de todos os *backend containers* para transmitir tráfego. As aplicações externas só precisam de se preocupar com um único ponto de acesso, mas beneficiam de um *backend* escalável ou pelo menos um *backend* que pode ser trocado quando necessário.

Um *Replication Controller* garante que um número especificado de "réplicas"do pacote esteja sendo executado a qualquer momento. Ou seja, um *Replication Controller* garante que um *pods* ou um conjunto homogêneo de *pods* esteja sempre disponível. Se houver muitos *pods*, ele vai "matar"alguns. Se houver poucos, o *Replication Controller* iniciará mais. A vantagem comparando aos *pods* criados manualmente, é que os *pods* mantidos por um *Replication Controller* são automaticamente substituídos se falharem, forem excluídos ou terminados.

## 3.2 Experimentos realizados em ambiente de desenvolvimento

Os testes iniciais com o Kubernetes foram realizadas em uma máquina de desenvolvimento com as seguintes especificações:

- Sistema Operacional: macOS Sierra (versão 10.12.2)
- Memória: 8GB de RAM
- Disco: 250 GB (SSD)

O Kubernetes foi instalado utilizando a solução *MiniKube*[4] em um ambiente virtualizado com *VirtualBox* seguindo as orientações da documentação oficial. A solução é compatível com OS X e Linux.

Para avaliar o *cluster* instalado foi utilizado o FIWARE GE AuthZForce. O *deploy* foi realizado com as seguintes instruções:

```
kubectl run authz --image=fiware/authzforce-ce-server:release-5.4.1
```

Para ativar a escalabilidade automática baseado em um limite de uso de CPU e definir o limite mínimo e máximo de réplicas foi utilizado a seguinte instrução:

```
kubectl autoscale deployment authz --min=10 --max=15 --cpu-percent=80
```

As demais informações sobre o ambiente podem ser visualizadas usando a interface web do Kubernetes (*Dashboard UI*) utilizando o comando:

```
minikube dashboard
```

Mais detalhes sobre o uso do *kubectl* podem ser encontrados na documentação oficial[5].

[4] https://kubernetes.io/docs/getting-started-guides/minikube
[5] https://kubernetes.io/docs/user-guide/kubectl-overview/

### 3.3 Experimentos realizados usando a nuvem do SmartMetropolis

Os experimentos com Kubernetes na nuvem OpenStack do projeto SmartMetropolis foram realizados usando a solução *Murano*[6]. Murano é um catálogo de aplicações para OpenStack, onde o usuário da nuvem via interface web seleciona as aplicações que deseja instalar, e o Murano cria as instâncias (máquinas virtuais) e instala a solução escolhida.

Para instalar o Kubernetes via Murano os seguintes passos foram efetuados:

- Acesso ao OpenStack Horizon (https://10.7.49.201/);
- Seleciona *Applications* -> *Catalog* -> *Browser*, seleciona "Kubernetes Cluster", e clica em "Quick Deploy";
- Informa o nome do *cluster* no campo "Cluster Name", o número mínimo e máximo de nós nos campos "Initial"e "Maximum Number of Kubernetes nodes", e clica em "Próximo";
- Seleciona o tipo de instância que será utilizada (ex.: m1.medium), a imagem (ex.: Debian, Ubuntu, etc.), o par de chaves SSH, a zona, e por fim clica em "Criar";

Após isso, o Murano irá criar as instâncias (máquinas virtuais) e instalar o Kubernetes nos diversos nós que foram definidos no passo anterior.

### 3.4 Discussão

As atividades relacionadas a estudos de soluções de alta-disponibilidade realizadas nesse trimestre foram realizadas com o uso da solução Kubernetes. Durante os testes realizados com Kubernetes foi possível fazer *deploy* do AuthZForce em um ambiente de *cluster*, e durante esse processo foram detectados problemas que precisam ser tratados nas próximas atividades.

Ao efetuar o *deploy* em 2 réplicas usando a imagem Docker padrão do AuthZForce disponibilizada pelo projeto FIWARE foi observado que será necessário efetuar mudanças na imagem com intuito de permitir o compartilhamento das pastas de domínios e políticas de controle de acesso (ex.: /domains) entre os diversos nós do *cluster*. Os demais GE's também tem suas particularidades, tais como pastas que também necessitam ser distribuídas entre os diversos nós do *cluster*, bancos de dados, controle de sessão nos servidores de aplicação (ex.: Tomcat), etc. Para as próximas ações, será necessário estudar os GE's em busca dessas particularidades, e efetuar mudanças e testes para que tais sistemas se adequem a realidade de um ambiente distribuído e de alta-disponibilidade.

## 4 Ambiente de Desenvolvimento para SGeoL

O ambiente de desenvolvimento para o SGeoL é composto por um conjunto de GEs FIWARE, implantadas e configuradas de forma integrada, atraves de um ambiente de gerenciamento de *containers*. A integração dessas GEs foi feita utilizando o Docker-Compose, onde os *containers* e seus respectivos serviços, portas, links e demais particularidades são definidos em um arquivo de configurações declarativas *docker-compose.yml*. O repositório do projeto SGeOL-Docker está disponível no GitLab do IMD[7].

[6] https://wiki.openstack.org/wiki/Murano

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

Depois que a infra-estrutura foi entregue a equipe de *Middleware*, juntamente com as credenciais das VM's, subimos um ambiente similar, que está sendo utilizado para explorar a utilização dos componentes Keyrock, PEP Proxy, PDP, e Wirecloud, sendo realizado pela bolsista Gabriela Cavalcante.

Durante o quarto trimestre foi possível realizar a entrega do ambiente implantado para o SGeoL, ao WP de *Middleware*, para que testes pudessem ser realizados e assim tivéssemos um *feedback* para melhorias. Até o momento não recebemos retorno, mas estamos a disposição da equipe do SGeoL.

# 5 Considerações Finais

Este documento apresentou um relato das atividades realizadas durante o terceiro trimestre de execução do projeto Smart Metropolis por parte do WP4 - Infraestrutura, no âmbito da tarefa de **Desenvolvimento, implantação e operação de infraestrutura computacional**. Durante o quarto trimestre, o foco principal da equipe foi na implantação da instância FIWARE a ser utilizada por outros membros do projeto, além de dar suporte ao desenvolvimento da aplicação SGEOL.

A implantação da instância FIWARE envolveu a implantação de uma nuvem OpenStack que se encontra operacional e em uso por alguns membros do projeto. Foi produzido um manual técnico detalhando o projeto e a implantação da nuvem, já considerando mudanças decorrentes da limitação de recursos para a nuvem do projeto. Além disso, foram implantadas as GEs FIWARE relacionadas ao capítulo técnico de segurança. Estas GEs foram estudadas a fundo, resultando na descoberta de erros de documentação, e código fonte, que foram comunicados à equipe responsável por seu desenvolvimento. Foram desenvolvidas com sucesso aplicações de exemplo demonstrando os diversos cenários de autenticação e autorização propostos pela documentação FIWARE. Com isso, a equipe adquiriu hoje um expertise no uso destas GEs que irão contribuir para o desenvolvimento de aplicações pelos outros membros do projeto, sendo inclusive procurados por aluno de doutorado da UFCG em busca de ajuda na implantação e uso destas GEs. Foi realizado também um estudo preliminar nas GEs relacionadas a IoT, iniciado ao final do quarto trimestre, com foco em sua implantação em um ambiente local. Embora as GEs tenham sido implantadas em um cenário de uso bastante simples, testes com uma aplicação cliente disponibilizada pela equipe FIWARE não obtiveram sucesso.

No tocante ao gerenciamento da instância FIWARE do projeto, foi realizado um estudo acerca da solução Kubernetes, buscando um ambiente de gerenciamento de infraestrutura com suporte a alta-disponibilidade. Tal estudo demonstrou que os *containers* disponibilizados pelos diversos repositórios do FIWARE necessitam de uma personalização para funcionarem em um ambiente de produção com alta-disponibilidade.

Como próximos passos, recomendamos o estudo e implantação das ferramentas de monitoramento disponibilizadas pelo FIWARE, ou de ferramentas equivalentes, para monitorar a instância FIWARE do projeto. Em paralelo, julgamos adequado aumentarmos a base de usuários da nuvem do projeto, e modo a se avaliar o consumo de recursos. Uma vez que a equipe foi capaz de implantar e utilizar com sucesso os GEs de segurança do FIWARE, sugerimos como próximo passo a definição dos mecanismos de autenticação e autorização da aplicação SGEOL explorando tais GEs. Em relação às GEs de IoT, sugerimos a definição de um cenário de uso antes de sua implantação, uma vez que essas GEs podem ser implantadas de diversas maneiras. Baseado no estudo realizado com a solução Kubernetes, sugerimos sua aplicação na infraestrutura do projeto, usando-se como base os componentes utilizados pela aplicação SGEOL, contribuindo assim para tratar questões de alta-disponibilidade.

# Referências

[1] BURNS, Brendan et al.. *Borg, Omega, and Kubernetes*. Communications of the ACM, v. 59, n. 5, p. 50-57, 2016.