Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

SmartMetropolis – Plataforma e Aplicações para Cidades Inteligentes

WP4 – Infraestrutura

Ferramentas de Monitoramento e Análise de Infraestrutura da Plataforma Fiware

Natal-RN, Brasil
Julho 2017

Equipe Técnica

Docentes

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

Discentes

Marcelo Avelino de Medeiros

Sumário

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, que no contexto do segundo ano de execução do projeto Smart Metropolis, tem os seguintes objetivos principais:

- Operação de Infraestrutura: Este objetivo engloba a operação e manutenção da instância Fiware do projeto, incluindo ferramentas para monitoramento da infraestrutura, e para gerenciamento de aplicações.
- Segurança da Informação e Controle de Acesso: Este objetivo engloba o estudo aprofundado dos mecanismos de segurança da plataforma Fiware com o fim de oferecer uma solução de controle de acesso que possa ser utilizada pelas aplicações que irão executar sobre a infraestrutura Fiware.

Neste contexto, este relatório corresponde ao entregável definido pela **Meta 1.6: Estudo das ferramentas de monitoramento e análise disponibilizadas pela plataforma FIWARE (OPS-Health)**, que envolve o estudo em ferramentas de monitoramento e análise da infraestrutura computacional de uma instância Fiware.

Neste trimestre o foco foi a implantação do serviço Sanity Check, disponibilizado pela plataforma FIWARE. Neste sentido, apresentamos uma descrição das diversas ferramentas disponibilizadas pelo OPS-Health (Seção **??**), seguido de um breve relato sobre a experiência de utilização desta ferramenta (Seção **??**).

2 Sanity Checks

A plataforma Fiware, em seu capítulo técnico OPS Tools[1] [1] oferece um conjunto de ferramentas para operação, monitoramento e manutenção dos nós que compõem a infraestrutura computacional de instâncias Fiware. Dentre elas, estão as ferramentas responsáveis pelas operações de monitoramento da nuvem, permitindo que o os administradores e usuários tenham uma noção da saúde dos nós e regiões da nuvem computacional.

Uma dessas ferramentas é o Sanity Checks, o qual possui dois componentes ou funcionalidades principais: Sanity Checks Engine e Sanity Checks Dashboard. Essas duas ferramentas e as demais componentes do Ops-Health foram descritas genericamente no relatório referente ao primeiro trimestre deste ano.

O Sanity Check é responsável por verificar o estado de saúde de cada nó FIWARE Lab. Ele dispõe basicamente de dois componentes: o Sanity Check Engine, que contém os casos de testes e as operações para realizar testes do tipo E2E na nuvem e o Sanity Check Dashboard, cujo papel principal é exibir os resultados das execuções de monitoramento feitas pelo Sanity Check Engine, funcionando como o front-end para ele.

Na sequência, detalhamos a experiência com o Sanity Checks Engine.

[1] `http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_OPS_Tools`

3 Sanity Checks Engine

O estudo do Sanity Checks Engine envolveu sua instalação em ambiente virtualizado, assim como o entendimento de como funciona os casos de teste utilizados pelo mesmo.

3.1 Instalação - Command line

Nesta primeira etapa, foi feita a instalação do Sanity Checks via linha de comando. Para tanto, devem seguir-se alguns passos, conforme o que é descrito no repositório do Sanity Checks.

O primeiro passo é baixar o repositório:

```
$ git clone https://github.com/Fiware/ops.Health.git
```

Feito isso, deve ser levado em conta a instalação dos requisitos para a instalar o Sanity Checks:

- Python 2.7 ou mais recente;
- Pip [1];
- Virtualenv [2] ou Vagrant [3];
- D-Bus [4] configurado e rodando no sistema;
- dbus-python v0.84.0 [5];
- pygobject v2.20.0 [6].

Além disso, é necessário instalar os pacotes disponíveis em um arquivo “requirements.txt” (está disponível ao baixar o repositório), que contém os pacotes python necessário na instalação e as suas devidas versões.

Feito isso, devemos seguir as etapas seguintes para a instalação do Sanity Checks para rodar os testes, utilizando uma virtualenv conforme a recomendação do Fiware Lab:

1. Instale o D-Bus (D-Bus e python-dbus);
2. Crie uma *virtualenv*:

   ```
   $ virtualenv WORKON_HOME/fiware-region-sanity-tests \\
    --system-site-packages
   ```

3. Ative a *virtualenv*:

   ```
   $ source WORKON_HOME/fiware-region-sanity-tests/bin/activate
   ```

4. Vá para o diretório do OPSHealth - Sanity Checks:

   ```
   $ cd ops.Health/fiware-region-sanity-tests
   ```

5. Instale os pacotes exigidos na *virtualenv*. Os pacotes que devem ser instalados estão no arquivo “requirements.txt", que está nesse diretório.

   ```
   $ pip install -r requirements.txt --allow-all-external
   ```

3.2 Casos de Teste

O Sanity Checks possui uma base de casos de testes para determinados serviços OpenStack no seu repositório. Os testes podem ser executados via linha de comando todos para uma mesma região (utilizando o comando *./sanity_checks RegionOne*) ou um determinado teste para aquela região (utilizando o comando *./sanity_checks RegionOne.test_name*).

Esses casos de teste estão disponíveis em arquivos desenvolvidos em Python. O Sanity Checks disponibiliza basicamente três arquivos com esses testes, de acordo com a aplicação de cada caso de teste para os Serviços OpenStack: base para todos os serviços, serviços que utilizam rede, serviços que não utilizam rede.

Esses casos de testes desenvolvidos em Python se baseiam na estrutura de testes unitários, utilizando o framework UnitTest (ou PyUnit), o qual é uma ferramenta do Python muito parecida com a ferramenta de testes unitários utilizada em Java (JUnit). No código mostrado abaixo há o método "test_create_container"da classe "FiwareRegionsObjectStorageTests". Esse código está no arquivo "fiware_region_obj

```
def test_create_container(self):
        """
        Test whether it is possible to create a new container
        into the object storage.
        """

        suffix = datetime.utcnow().strftime('%Y%m%d%H%M%S')
        container_name = TEST_CONTAINER_PREFIX + suffix

        response = self.swift_operations.create_container(container_name)

        self.assertIsNone(response, "Container could not be created")
        self.test_world['containers'].append(container_name)
        self.logger.debug("Created %s container was created", container_name)

        response = self.swift_operations.get_container(container_name)
        self.assertEquals('x-container-object-count' in response[0], \
        True, "There is no container header in response")
        self.assertEquals(len(response[-1]), 0, "The container \
                is not empty")  # The list of items should be 0.
        self.logger.debug("Getting %s container details from \
                the object storage", container_name)
```

Esse caso de teste executa a operação de criar um container na nuvem. Dentro do repositório do Engine há um diretório com outros arquivos Python que têm as operações utilizadas na nuvem. No código acima, o método create_container das operações do Swift é chamado no método de test_create_container. Essa é a estrutura básica de testes unitários: há um método (criar container) e uma classe de teste (teste de criar container) onde a classe dos testes estende a classe das operações.

A tabela a seguir apresenta a descrição de cada um dos testes disponibilizados hoje pelo Sanity Checks comuns a todos os serviços do OpenStack. Estes casos de teste estão localizados no arquivo "fiware_region_base_tests.py ".

Além dos testes comuns a todos os serviços, o Sanity Checks possui outros dois arquivos com testes relacionados aos serviços de Object Storage (Swift) e para regiões que possuem e não possuem serviço de rede (Neutron e Nova).

Tabela 1: Casos de Teste comuns a todos os serviços

Caso de Teste	**Descrição**
test_allocate_ip	Testa a alocação de um IP público.
test_flavors_not_empty	Testa se uma região tem flavors.
test_images_not_empty	Testa se uma região tem imagens.
test_required_images	Testa se uma região possui todos os requisitos de imagens (como especificado nas configurações).
test_cloud_init_aware_images	Testa se uma região tem imagens 'cloud-init-aware' (adequado para planos).
test_base_image_for_testing_exists	Testa se a região tem uma imagem usada para teste.
test_create_security_group_and_rules	Testa a criação de um novo grupo de segurança com regras.
test_create_keypair	Testa a criação de uma nova keypair.

Os casos de teste da tabela 2 apresentam a descrição dos casos de teste para regiões que possuam o serviço de Object Storage (Swift) e estão contigos no arquivo "fiware_region_object_storage_tests.py".

Tabela 2: Casos de Teste - Object Storage (Swift)

Caso de Teste	**Descrição**
test_create_container	Testa se é possível criar um novo container no object storage.
test_delete_container	Testa se é possível deletar um container.
test_create_text_object_and_download_it_from_container	Testa se é possível fazer o upload e o download de um arquivo de texto.
test_delete_an_object_from_a_container	Testa se é possível deletar um objeto de um container.
test_create_big_object_and_download_it_from_container	Testa se é possível fazer o upload e o download de um arquivo grande (mais de 5 Megabytes).

Os casos de teste da tabela 3 apresentam a descrição dos casos de teste para Regiões sem um serviço de rede OpenStack (Neutron e Nova) e estão localizados no arquivo "fiware_region_without_networks_tests.py".

Os casos de teste da tabela 4 apresentam a descrição dos casos de teste para Regiões que tenham um serviço de rede OpenStack (Neutron e Nova) e estão localizados no arquivo "fiware_region_with_networks_tests.py".

3.3 Configuração e Execução

Quando o Sanity Checks é instalado é necessário alterar as variáveis de acesso à nuvem e configuração dos testes no arquivo "settings.json". Esse arquivo está no diretório "ops.Health/fiware-region-sanity-tests/etc". Nele é necessário configurar as credenciais de acesso:

- keystone_url: URL de autenticação;
- user_id: ID do usuário utilizado nos testes;

Tabela 3: Casos de Teste - Regiões sem um serviço de rede OpenStack (Neutron e Nova)

Caso de Teste	**Descrição**
test_deploy_instance_with_custom_metadata	Testa se é possível fazer o deploy de uma instância com custom metadata.
test_deploy_instance_with_keypair	Testa se é possível fazer o deploy de uma instância com uma nova keypair.
test_deploy_instance_with_sec_group	Testa se é possível fazer o deploy de uma instância com um novo security group.
test_deploy_instance_with_all_params	Testa se é possível fazer o deploy de uma instância com todos os parâmetros.
test_deploy_instance_and_associate_public_ip	Testa se é possível fazer o deploy de uma instância e atribuir a alocação de um IP público.
test_deploy_instance_and_e2e_connection_using_public_ip	Testa se é possível fazer o deploy de uma instância, atribuir um IP público alocado e estabelecer uma conexão SSH.
test_deploy_instance_and_e2e_snat_connection	Testa se é possível fazer o deploy de uma instância e conectá-la à INTERNET (SNAT) sem atribuir um IP público.
test_deploy_instance_and_check_metadata_service	Testa se é possível fazer o deploy de uma instância e verificar se os serviços metadata estão trabalhando corretamente (PhoneHome service).

- tenant_id: ID do projeto;
- tenant_name: Nome do projeto;
- user_domain_name: Nome do domínio do usuário;
- project_domain_name: Nome do domínio do projeto;
- username: nome do usuário;
- password: senha de acesso à nuvem.

Além de informar as credenciais de acesso à nuvem também é necessário definir as configurações dos testes. Essas definições são informadas na tag test_configuration e trazem os valores referentes ao endpoint do servidor PhoneHome (usando para alguns casos de testes), as configurações do glance e do swift (definir os valores de glance_configuration e do swift_configuration). Essas definições podem ser por exemplo os links dos arquivos que serão utilizados para testar Download e Upload no swift. Além disso, é preciso definir a URL do serviço Metadata do OpenStack.

Por fim, é preciso configurar alguns valores da tag "region_configuration". Esses valores não precisam todos ser definidos. São eles:

- external_network_name: is the network for external floating IP addresses
- shared_network_name: nome da rede compartilhada usada para os testes E2E;
- test_object_storage: habilitar ou não testes no serviço de object storage (true para habilitar);
- test_flavor: especificar as flavors e instâncias para os testes;
- test_image: especificar sa bases de imagens usadas nos testes;
- test_login_name: especifica o usuário para login em uma instância dos testes.

4 Sanity Checks Dashboard

O Sanity Checks Dashboard funciona basicamente como o front-end do Sanity Checks. O seu principal objetivo é mostrar os resultados dos testes realizados e os status das regiões após esses testes. Para tanto, o Dashboard também mostra em detalhes os logs dos testes que apresentaram falhas nos resultados.

Para realizar a instalação do Sanity Checks Dashoboard foi seguido procedimento padrão definido no repositório do projeto, portanto, foi utilizado o pacote do Dashboard para a instalação, por meio do seguinte comando:

```
$ sudo yum install fiware-fihealth-dashboard
```

Embora o Dashboard tenha sido instalado, não exploramos sua interface pois estávamos lidando com alguns erros apresentados pelo Sanity Checks Engine.

5 Considerações Finais

Este documento apresentou um relato de experiência com o Sanity Checks Engine e Dashboard da plataforma FIWARE. Embora o Sanity Checks Engine tenha sido instalado com sucesso, e a estrutura de testes tenha sido bem entendida, alguns erros na realização dos testes impediram o avanço nos estudos. Levantamento preliminar apontam para problemas de acesso à nuvem de desenvolvimento e no uso de certificados digitais auto-assinados.

Como próximos passos, pretendemos explorar uma instância simplificada de nuvem para investigação dos erros mencionados, além de rodar alguns testes utilizando a nuvem acadêmica do IMD.

Referências

[1] Silvio Cretti, *D.21.1.2: Platform deployment, operations, analytics and support tools*, Technical report Number: ICT-2013-FI-632893-WP21-D.21.1.2, Disponível em `https://forge.fiware.org/docman/view.php/7/5652/D.21.1.2+Platform+deployment%2C+operations%2C+analytics+and+support+tools.pdf`, 2016.

Tabela 4: Casos de Teste - Regiões com um serviço de rede OpenStack (Neutron e Nova)

Caso de Teste	**Descrição**
test_create_network_and_subnet	Testa se é possível criar uma nova rede com sub redes.
test_external_networks	Testa se há redes externas configuradas na região.
test_create_router_no_external_network	Testa se é possível criar um novo roteador configurando o gateway.
test_create_router_external_network	Testa se é possível criar um novo roteador sem um gateway padrão. whether it is possible to create a new router with a default gateway.
test_create_router_no_external_network_and_add_network_port	Testa se é possível criar um novo roteador sem um gateway externo e um link para uma nova porta de rede.
test_deploy_instance_with_new_network	Testa se é possível fazer o deploy de uma instância com uma nova rede.
test_deploy_instance_with_new_network_and_metadata	Testa se é possível fazer o deploy de uma instância com uma nova rede e custom metadata.
test_deploy_instance_with_new_network_and_keypair	Testa se é possível fazer o deploy de uma instância com uma nova rede e uma nova keypair.
test_deploy_instance_with_new_network_and_sec_group	Testa se é possível fazer o deploy de uma instância com uma nova rede e um novo security group.
test_deploy_instance_with_new_network_and_all_params	Testa se é possível fazer o deploy de uma instância com uma nova rede todos os parâmetros.
test_deploy_instance_with_new_network_and_associate_public_ip	esta se é possível fazer o deploy de uma instância com uma nova rede e atribuindo um IP público alocado.