FIWARE-SDC

Release

September 30, 2016

Contents

CHAPTER 1

Introduction

Sagitta is a Java implementation of the SDC Manager GE developed as a part of the FIWARE platform.

Sagitta (the Software Deployment and Configuration -SDC- GE), which is the key enabler used to support automated deployment (installation and configuration) of software on running virtual machines. As part of the complete process of deployment of applications, the aim of Sagitta is to deploy software product instances upon request of the user using the API or through the Cloud Portal.

The SDC Manager source code can be found here

This documentation offers deeper information on SDC Manager.

Documentation

1.1 FIWARE SDC | Sagitta

1.1.1 Introduction

This is the code repository for FIWARE Sagitta, the reference implementation of the Software Deployment and Configuration GE.

This project is part of FIWARE. Check also the FIWARE Catalogue - Software Deployment and Configuration GE.

Any feedback on this documentation is highly welcome, including bugs, typos or things you think should be included but aren't. You can use FIWARE SDC - GitHub issues to provide feedback.

For documentation previous to release 4.4.2 please check the manuals at FIWARE public wiki:

- FIWARE SDC - Installation and Administration Guide
- FIWARE SDC - User and Programmers Guide

Top

1.1.2 GEi overall description

The FIWARE Software Deployment and Configuration (SDC) GE is is the key enabler used to support automated deployment (installation and configuration) of software on running virtual machines. As part of the complete process of deployment of applications, the aim of SDC GE is to deploy software product instances upon request of the using the SDC API or through the Cloud Portal, which in turn uses the PaaS Manager GE (see FIWARE PaaS Manager).

After that, users will be able to deploy artifacts, that are part of the application, on top of the deployed product instances.

Top

1.1.3 Build and Install

The recommended procedure is to install using RPM packages in CentOS 6.x as it is explained in the following document . If you are interested in building from sources, check this document.

Requirements

- System resources: see these recommendations.
- Operating systems: CentOS (RedHat), being CentOS 6.5 the reference operating system.

Installation

Using FIWARE package repository (recommended)

Refer to the documentation of your Linux distribution to set up the URL of the repository where FIWARE packages are available (and update cache, if needed):

```
http://repositories.testbed.fiware.org/repo/rpm/x86_64
```

Then, use the proper tool to install the packages:

```
$ sudo yum install fiware-sdc
```

and the latest version will be installed. In order to install a specific version:

```
$ sudo yum install fiware-sdc-{version}-1.noarch
```

where {version} being the specific version to be installed

Upgrading from a previous version

Unless explicitly stated, no migration steps are required to upgrade to a newer version of the Software Deployment and Configuration components:

- When using the package repositories, just follow the same directions described in the *Installation* section (the `install` subcommand also performs upgrades).
- When upgrading from downloaded package files, use `rpm -U` in CentOS

Upgrading database

In case the database needs to be upgrade, the script db-changelog.sql should be execute. To do that, it just needed to execute:

```
psql -U postgres -d $db_name << EOF
\i db-changelog.sql
```

Using installation script

The installation of fiware-sdc can be done in the easiest way by executing the script:

```
scripts/bootstrap/centos.sh
```

The script will ask you the following data to configure the configuration properties:

- The database name for the fiware-sdc
- The postgres password of the database
- the keystone url to connect fiware-sdc for the authentication process
- the admin keystone user for the authentication process
- the admin password for the authentication process

Top

1.1.4 Running

As explained in the *GEi overall description* section, there are a variety of elements involved in the Software Delivery and Configuration architecture, apart from those components provided by this Software Delivery and Configuration GE (at least, an instance of configuration engine like Chef server of Puppet master). Please refer to their respective documentation for instructions to run them.

In order to start the software deployment and configuration service, as it is based on a web applicatin on top of jetty, just you should run:

```
$ service fiware-sdc start
```

Then, to stop the service, run:

```
$ service fiware-sdc stop
```

We can also force a service restart:

```
$ service fiware-sdc restart
```

Configuration file

The configuration of SDC is in configuration_properties table in the database. There, it is required to configure:

```
$ openstack-tcloud.keystone.url: This is the url where the keystone-proxy is deployed
$ openstack-tcloud.keystone.user: the admin user
$ openstack-tcloud.keystone.password: the admin password
$ openstack-tcloud.keystone.tenant: the admin tenant
$ sdc_manager_url: the final url, mainly https://sdc-ip:8443/sdc
```

In addition, to configue the SDC application inside the webserver, it is needed to change the context file. To do that, change sdc.xml found in distribution file and store it in folder $SDC_HOME/webapps/:

```
<New id="sdc" class="org.eclipse.jetty.plus.jndi.Resource">
  <Arg>jdbc/sdc</Arg>
  <Arg>
      <New class="org.postgresql.ds.PGSimpleDataSource">
          <Set name="User"> <database user> </Set>
          <Set name="Password"> <database password> </Set>
          <Set name="DatabaseName"> <database name>   </Set>
          <Set name="ServerName"> <IP/hostname> </Set>
          <Set name="PortNumber">5432</Set>
      </New>
  </Arg>
</New>
```

Checking status

In order to check the status of the service, use the following command (no special privileges required):

```
$ service fiware-sdc status
```

Top

1.1.5 API Overview

The Software Deployment and Configuration offers a REST API, which it can be used for both managing the software catalogue and the installation of software in virtual machines.

For instance, it is possible to obtain the software list in the catalogue with the following curl

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml"
-H "X-Auth-Token: your-token-id" -H "Tenant-Id: your-tenant-id"
-X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog/product"
```

Please have a look at the API Reference Documentation section bellow and at the programmer guide.

API Reference Documentation

- FIWARE SDC v1 (Apiary)

Top

1.1.6 Testing

Unit tests

The `test` target for each module in the SDC is used for running the unit tests in both components of SDC GE. To execute the unit tests you just need to execute:

```
mvn test
```

Please have a look at the section building from source code in order to get more information about how to prepare the environment to run the unit tests.

Acceptance tests

In the following path you will find a set of tests related to the end-to-end funtionalities.

- SDC Aceptance Tests

To execute the acceptance tests, go to the test/acceptance folder of the project and run:

```
lettuce_tools --tags=-skip.
```

This command will execute all acceptance tests (see available params with the -h option)

End to End testing

Although one End to End testing must be associated to the Integration Test, we can show here a quick testing to check that everything is up and running. It involves to obtain the product information storaged in the catalogue. With it, we test that the service is running and the database configure correctly:

```
https://{SDC\_IP}:{port}/sdc/rest
```

The request to test it in the testbed should be:

```
curl -v -k -H 'Access-Control-Request-Method: GET' -H 'Content-Type: application xml'
-H 'Accept: application/xml' -H 'X-Auth-Token: 5d035c3a29be41e0b7007383bdbbec57'
-H 'Tenant-Id: 60b4125450fc4a109f50357894ba2e28' -X GET
'https://localhost:8443/sdc/rest/catalog/product'
```

the option -k should be included in the case you have not changed the security configuration of SDC. The result should be the product catalog.

If you obtain a 401 as a response, please check the admin credentials and the connectivity from the sdc machine to the keystone (openstack-tcloud.keystone.url in configuration_properties table)

Top

1.1.7 Advanced topics

- Installation and administration
  - Software requirements
  - Building from sources
  - Resources & I/O Flows
- User and programmers guide

Top

1.1.8 Support

Ask your thorough programming questions using **'stackoverflow'_** and your general questions on **'FIWARE Q&A'_**. In both cases please use the tag *fiware-sagitta*

Top

1.1.9 License

Top

1.2 SDC - User and Programmers Guide

1.2.1 Introduction

Welcome the User and Programmer Guide for Software Deployment and Configuration.

1.2.2 Accessing SDC from the CLI

The access through the CLI is made using the curl program. Curl [http://curl.haxx.se/] is a client to get documents/files from or send documents to a server, using any of the supported protocols (HTTP, HTTPS, FTP, GOPHER, DICT, TELNET, LDAP or FILE) and therefore is also usable for OpenStack Compute API. Use the curl command line tool or use libcurl from within your own programs in C. Curl is free and open software that compiles and runs under a wide variety of operating systems.

The normal operations sequence to deploying an environment and an application on top of it could be summarized in the following list:

API Authentication

All the operations in the SDC API needs to have a valid token to access it. To obtain the token, you need to have an account in FIWARE Lab (account.lab.fi-ware.org). With the credentials (username, password and tenantName) you can obtain a valid token. From now on, we asume that the value of your tenant-id is "your-tenant-id"

```
$ curl -v -H "Content-Type: application json" -H "Accept: application/json" -X
POST "http://cloud.lab.fi-ware.org:4731/v2.0/tokens" -d '{"auth":{"tenantName":
"your-tenant-id","passwordCredentials":{"username":"youruser",
"password":"yourpassword"}}}'
```

You will receive the following answer, with a valid token (id).

```
{
access: {
   token: {
      expires: "2015-07-09T15:16:07Z"
      id: "756cfb31e062216544215f54447e2716"
      tenant: {
      ..
}
```

For all the SDC request, you will need to include the following header:

```
X-Auth-Token: 756cfb31e062216544215f54447e2716
Tenant-Id: your-tenant-id
```

For the rest of the explanation, we are going to configure a set of variables:

```
export SDC_IP =  saggita.lab.fi-ware.org
```

Catalogue Management API

Next we detail some operations that can be done in the catalogue management api

Product API

Get the Product List from the catalogue

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog/product"
```

This operation lists all the products stored in the catalogue. The following example shows an XML response for the Product List API operation.

```
<products>
            <product>
                    <name>tomcat</name>
                    <description>tomcat J2EE container</description>
            </product>
            ...
            <product>
                    <name>mysql</name>
                    <description>mysql</description>
                    <attributes>
                            <key>key1</key>
                            <value>value1<value/>
                            <description>description1</description>
                    </attributes>
                    </metadatas>
                    <metadatas>
                            <key>installator</key>
```

```
                            <value>chef</value>
                            <description>mysql installator</description>
                    </metadatas>
            </product>
    <products>
```

Get the Details of a particular Product List

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog
    /product/{product-name}"
```

This operation lists the environments stored in the catalogue. The following example shows an XML response for the list Environment API operation. It is possible to see it contains a list of tiers including products to be installed.

```
<product>
        <name>mysql</name>
        <description>mysql</description>
        <attributes>
                <key>key1</key>
                <value>value1<value/>
                <description>description1</description>
        </attributes>
        </metadatas>
        <metadatas>
                <key>installator</key>
                <value>chef</value>
                <description>mysql installator</description>
        </metadatas>
</product>
```

Add a New Product to the catalogue

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X POST "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog
    /product/{product-name}"
```

with the following payload (with metadatas and attributes)

```
<product>
        <name>{product-name}</name>
        <description>Description</description>
        <attributes>
                <key>key1</key>
                <value>value1<value/>
                <description>description1</description>
        </attributes>
        <metadatas>
                <key>installator</key>
                <value>chef</value>
                <description>mysql installator</description>
        </metadatas>
</product>
```

Delete a Product from the catalogue

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X DELETE "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog
    /product/{product-name}"
```

Product Release API

Get the Releases List of a particular Product

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog
    /product/{product-name}/release"
```

This operation lists the product releases of {product-name} stored in the catalogue. The following example shows an XML response for the list of ProductRelease API operation.

```
<productReleases>
        <productRelease>
                <releaseNotes>{product-name} 0.6.15</releaseNotes>
                <version>0.6.15</version>
                <product>
                        <name>{product-name}</name>
                        <description>desc</description>
                </product>
                <supportedOOSS>
                        <id>1</id>
                        <v>0</v>
                        <osType>94</osType>
                        <name>Ubuntu</name>
                        <description>Ubuntu 10.04</description>
                        <version>10.04</version>
                </supportedOOSS>
        </productRelease>
        <productRelease>
                <version>0.9.0</version>
                <product>
                        <name>{product-name}</name>
                        <description>{product-name} 0.6.15</description>
                </product>
                <supportedOOSS>
                        <id>1</id>
                        <v>0</v>
                        <osType>94</osType>
                        <name>Ubuntu</name>
                        <description>Ubuntu 10.04</description>
                        <version>10.04</version>
                </supportedOOSS>
        </productRelease>
</productReleases>
```

Get the Details of a Particular Product Release

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog
    /product/{product-name}/release/{version}"
```

This operation lists the details of a Product Release.

```
<productReleases>
        <productRelease>
                <releaseNotes>{product-name} 0.6.15</releaseNotes>
                <version>0.6.15</version>
                <product>
                        <name>{product-name}</name>
                        <description>desc</description>
                </product>
                <supportedOOSS>
                        <id>1</id>
                        <v>0</v>
                        <osType>94</osType>
                        <name>Ubuntu</name>
                        <description>Ubuntu 10.04</description>
                        <version>10.04</version>
                </supportedOOSS>
        </productRelease>
</productReleases>
```

Add a New Release to a Product into the catalogue

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X POST "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog
    /product/{product-name}/release"
```

with the following payload

```
<productReleaseDto>
        <productName>{product-name}</productName>
        <version>{version}</version>
        <productDescription>description of {product-name}-{version}/productDescription>
</productReleaseDto>
```

Delete the Release of a Product

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X DELETE "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog
    /product/{product-name}/release"
```

Get All Product and Releases of the catalogue

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/catalog/productandrelease"
```

This operation lists all product releases stored in the Catalogue and available for users.

```
<productAndReleaseDtoes>
        <productAndReleaseDto>
                <product>
                        <name>tomcat</name>
                        <description>tomcat J2EE container</description>
                </product>
                <version>6</version>
        </productAndReleaseDto>
        ...
```

```
        <productAndReleaseDto>
                <product>
                        <name>nodejs</name>
                        <description>nodejs</description>
                </product>
                <version>0.6.15</version>
        </productAndReleaseDto>
</productAndReleaseDtoes>
```

Product Instance Provisioning API

Install a Product in a Virtual Machine

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X POST "http://saggita.lab.fi-ware.org:8080/sdc/rest/vdc
    /{your-tenant-id}/productInstance"
```

where {your-tenant-id} is the tenant-id in this guide. The payload of this request can be as follows:

```
<productInstanceDto>
        <vm>
        <ip>{ip}</ip>
        <fqn>{fqn}</fqn>
        <hostname>{hostname}</hostname>
        </vm>
        <product>
        <productDescription/>
        <name>{product-name}</name>
        <version>{product-version}</version>
        </product>
        <attributes>
                <key>custom_att_02</key>
                <value>default_value_plain</value>
                <type>Plain</type>
        </attributes>
</productInstanceDto>
```

The response obatined should be:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <task href="https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/task/{task-id}" startTime="2012-11-08T09:13:18.311+01:00"
    status="RUNNING">
    <description>Install product {product-name} in  VM {vm}</description>
    <vdc>your-tenant-id</vdc>
</task>
```

Given the URL obtained in the href in the Task, it is possible to monitor the operation status (you can check Task Management). Once the environment has been deployed, the task status should be SUCCESS.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <task href="https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/task/{task-id}" startTime="2012-11-08T09:13:28.311+01:00"
     status="SUCCESS">
    <description>Install product {product-name} in  VM {vm}</description>
    <vdc>your-tenant-id</vdc>
</task>
```

Get the list of Product Instances deployed

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/productInstance"
```

The Response obtained includes all the blueprint instances deployed

```
<productInstances>
        <productInstance>
                <id>8790</id>
                <date>2014-10-30T12:49:35.528+01:00</date>
                <name>{productInstance-name}</name>
                <status>INSTALLED</status>
                <vm>
                <ip>{ip}</ip>
                <fqn>{fqn}</fqn>
                <hostname>{hostname}</hostname>
                </vm>
                <vdc>{your-tenant-id}</vdc>
                <productRelease>
                        <version>{product-version}</version>
                        <product>
                                <name>{product-name}</name>
                                <metadatas>
                                        <key>key1</key>
                                        <value>value1</value>
                                        <description>desc</description>
                                </metadatas>
                        </product>
                </productRelease>
        </productInstance>
        ...
        <productInstance>
                ...
        </productInstance>
</productInstances>
```

Get details of a certain Product Instance

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/productInstance/{productInstance-name}"
```

This operation does not require any payload in the request and provides a BlueprintInstance XML response.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<productInstance>
        <id>8790</id>
        <date>2014-10-30T12:49:35.528+01:00</date>
        <name>mykurentoinstance-kurento-1-003237_kurento_5.0.3</name>
        <status>INSTALLED</status>
        <vm>
                <ip>130.206.126.23</ip>
                <hostname>mykurentoinstance-kurento-1-003237</hostname>
                <domain />
                <fqn>mykurentoinstance-kurento-1-003237</fqn>
                <osType />
```

```
        </vm>
        <vdc>{your-tenant-id}</vdc>
        <productRelease>
                <version>{product-version}</version>
                <product>
                        <name>{product-name}</name>
                        <metadatas>
                                <key>key1</key>
                                <value>value1</value>
                                <description>desc</description>
                        </metadatas>
                </product>
        </productRelease>
</productInstance>
```

Uninstall a Product Instance

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X DELETE "https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/productInstance/{productInstance-name}"
```

This operation does not require a request body and returns the details of a generated task.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
/{your-tenant-id}/task/{task-id}" startTime="2012-11-08T09:45:44.020+01:00"
 status="RUNNING">
    <description>Uninstall Product</description>
    <vdc>your-tenant-id</vdc>
</task>
```

With the URL obtained in the href in the Task, it is possible to monitor the operation status (you can checkTask Management). Once the environment has been undeployed, the task status should be SUCCESS.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="https://saggita.lab.fi-ware.org:8443/sdc/rest//vdc
/{your-tenant-id}/task/{task-id}" startTime="2012-11-08T09:13:19.567+01:00"
 status="SUCCESS">
    <description>Unistall product {product-name}</description>
    <vdc>your-tenant-id</vdc>
</task>
```

Node Management API

Load a particular node

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/chefClient/{node-name}"
```

This operation lists information of a specific node.

Delete a particular node

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
```

```
    -X DELETE "https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/chefClient/{node-name}"
```

Task Management

Get a specific task

```
$ curl -v -H "Content-Type: application/json" -H "Accept: application/xml" -H
"X-Auth-Token: 756cfb31e062216544215f54447e2716" -H "Tenant-Id: your-tenant-id"
    -X GET "https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/task/{task-id}"
```

This operation recovers the status of a task created previously. It does not need any request body and the response body in XML would be the following.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <task href="https://saggita.lab.fi-ware.org:8443/sdc/rest/vdc
    /{your-tenant-id}/task/{task-id}" startTime="2012-11-08T09:13:18.311+01:00"
     status="SUCCESS">
    <description>Install product {product-name} in  VM {vm}</description>
    <vdc>your-tenant-id</vdc>
</task>
```

The value of the status attribute could be one of the following:

Value	Description
QUEUED	The task is queued for execution.
PENDING	The task is pending for approval.
RUNNING	The task is currently running.
SUCCESS	The task is completed successfully.
ERROR	The task is finished but it failed.
CANCELLED	The task has been cancelled by user.

1.3 SDC Manager - Installation and Administration Guide

1.3.1 Introduction

This guide defines the procedure to install the different components that build up the SDC Manager GE, including its requirements and possible troubleshooting. The guide includes two different ways of installing SDC Manager: Installation from rpm or installation from source (building previously the rpm).

1.3.2 Requirements

In order to execute the SDC, it is needed to have previously installed the following software:

- Chef node
- Chef server [http://wiki.opscode.com/display/chef/Installing+Chef+Server]. For CentOS it is possible to follow the following instructions[http://blog.frameos.org/2011/05/19/installing-chef-server-0-10-in-rhel-6-scientificlinux-6/]
- SDC node
- open jdk 7

- PostgreSQL [http://www.postgresql.org/]

SDC should be installed in a host with at least 2Gb RAM.

1.3.3 Installation via script (for CentOS)

The installation of fiware-sdc can be done in the easiest way by executing the script

```
scripts/bootstrap/centos.sh
```

that is in the github repository of the project.

In order to perform the installation via script, git should be installed (yum install git). Just clone the github repository:

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

and go to the folder

```
cd fiware-sdc/scripts/bootstrap
```

Assign the corresponding permissions to the script centos.sh and execute under root user

```
./centos.sh
```

The script will ask you the following data:

- The database name for the fiware-sdc
- The postgres password of the database
- the keystone url to connect fiware-sdc for the authentication process
- the admin keystone user for the authentication process
- the admin password for the authentication process

Once the script is finished, you will have fiware-sdc installed under /opt/fiware-sdc/ . Please go to the Sanity Check section in order to test the installation. This script does not insert the fiware-sdc data into the keystone, so this action has to be done manually. In order to complete the installation please refer to Register SDC application into keystone section.

The SDC installation via script does not include either Chef server installation nor the Puppet installation. To perform these installations please refer to the corresponding sections included in this guide.

1.3.4 Manual Installation (for CentOS)

Install SDC from RPM

The SDC is packaged as RPM and stored in the rpm repository. Thus, the first thing to do is to create a file in /etc/yum.repos.d/fiware.repo, with the following content.

```
[Fiware]
name=FIWARE repository
baseurl=http://repositories.testbed.fi-ware.org/repo/rpm/x86_64/
gpgcheck=0
enabled=1
```

After that, you can install the SDC just doing:

```
yum install fiware-sdc
```

and the latest version will be installed. In order to install a specific version

```
yum install fiware-sdc-{version}-1.noarch
```

where {version} being the specific version to be installed

Install SDC from source

Requirements: To install SDC from source it is required to have the following software installed in your host previously:

- git
- java 1.7
- maven

Here we include a small guide to install the required software. If you find any problem in the installation process, please refer to the official sites:

Install git

```
sudo yum install git
```

Install java 1.7

```
sudo yum install java-1.7.0-openjdk-devel
```

Install maven 2.5

```
sudo yum install wget
wget http://mirrors.gigenet.com/apache/maven/maven-3/3.2.5/binaries
/apache-maven-3.2.5-bin.tar.gz

su -c "tar -zxvf apache-maven-3.2.5-bin.tar.gz -C /usr/local"
cd /usr/local
sudo ln -s apache-maven-3.2.5 maven
```

Add the following lines to the file /etc/profile.d/maven.sh

```
# Add the following lines to maven.sh
export M2_HOME=/usr/local/maven
export M2=$M2_HOME/bin
PATH=$M2:$PATH
```

In order to check that your maven installation is OK, you should exit your current session with “exit” command, enter again and type

```
mvn -version
```

if the system shows the current maven version installed in your host, you are ready to continue with this guide.

Now we are ready to build the SDC rpm and finally install it

The SDC is a maven application so, we should continue with the following instructions:

- Download SDC code from github

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

- Go to fiware-sdc folder and compile, launch test and build all modules

```
cd fiware-sdc/
mvn clean install
```

- Create a zip with distribution in target/sdc-server-dist.zip

```
$ mvn assembly:assembly -DskipTests

#$ cp target/distribution/sdc-server-dist {folder}
#$ {folder}/sdc-server-dist/bin/generateselfsigned.sh start
#$ cd {folder}/sdc-server-dist/bin ; ./jetty.sh start
```

- You can generate a rpm o debian packages (using profiles in pom)

for debian/ubuntu:

```
$ mvn install -Pdebian -DskipTests
(created target/sdc-server-XXXXX.deb)
```

for centOS (you need to have installed rpm-bluid. If not, please type "yum install rpm-build"):

```
$ mvn package -P rpm -DskipTests
(created ./target/rpm/sdc/RPMS/noarch/fiware-sdc-XXXX.noarch.rpm)
```

Finally go to the folder where the rpm has been created (./target/rpm/sdc/RPMS/noarch) and execute

```
cd target/rpm/fiware-sdc/RPMS/noarch
rpm -i <rpm-name>.rpm
```

Please, be aware that the supported installation method is the RPM package. If you use other method, some extra steps may be required. For example, you would need to generate manually the certificate (see the section about "Configuring the HTTPS certificate" for more details):

```
fiware-sdc/bin/generateselfsigned.sh
```

Requirements: Installation instructions

Chef server

Chef server installation (Centos 6.5) The SDC installation involves also to install the chef-server package, which can be obtained in [http://www.getchef.com/chef/install/]. If you find any problem in the chef-server installation process, please refer to the chef-serve official site. This small guide has been tested on Centos6.5

Go to this url and select the chef-server version you are interested in, depending also on your own operating system. Copy the url to download the selected chef-server version and type

```
wget <chef-server-url>
```

in this example we have

```
chef-server-url = https://opscode-omnibus-packages.s3.amazonaws.com
                            /el/6/x86_64/chef-server-11.1.6-1.el6.x86_64.rpm
```

In case you do not have wget installed on your system, please type 'yum install wget' to install it. We can just execute

```
mv chef-server-11.1.6-1.el6.x86_64.rpm chef-server-package.rpm
rpm -Uvh chef-server-package.rpm
```

Verify the the hostname for the Chef server by running the 'hostname'command. The hostname for the Chef server must be a FQDN. This means hostaname.domainame. In case it is not configure, you can do it

```
hostname chef-server.localdomain
```

and include it in the /etc/hosts

After that, it is required to configure the certificates and other staff in the chef-server, with chef-server-ctl. This command will set up all of the required components, including Erchef, RabbitMQ, and PostgreSQL.

```
sudo chef-server-ctl reconfigure
```

In order to test verify the installation of Chef Server 11.x by running the following command:

```
sudo chef-server-ctl test
```

After that, you can obtain the different certificates for the different clients in /etc/chef-server. There you can find a chef-validator.pem (needed for all the nodes), the chef-server-gui for the GUI. You can copy them in order to use them later.

Chef server cookbook repository The FIWARE cookbook repository is in FIWARE SVN repository. To upload the recipes into the chef server you need:

- To dowload the svn repository ('yum install svn' if not installed):

```
svn checkout https://forge.fiware.org/scmrepos/svn/testbed/trunk/cookbooks
```

- Inside the cookbooks folder, create a file update with the following content. It will update the repository and upload into the chef-server

```
svn update
knife cookbook upload --all -o BaseRecipes/
knife cookbook upload --all -o BaseSoftware/
knife cookbook upload --all -o GESoftware/
```

Chef-client installation and configuration The next step is to configure a client in the chef-server so that you can execute the chef-server CLI. To do that, you need to install the chef-client

```
curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

Before you configure the chef-client you should add the admin.pem and chef-validator.pem to the directory where chef-client finds its configuration (By default should be $HOME/.chef), the admin.pem and chef-validator.pem files should be placed in this directory before starting the chef-client configuration.

To configure chef-client, type the following command. You can accept all the default

```
knife configure --initial
```

The script will ask the following parameters:

- Please enter the chef server URL: use the FQDN (type "hostname" to find out) for the Chef server
- A name for the new user: use "station1"
- A name for the admin user [admin]: keep the default option
- location of the existing admin's private key: type the new location given
- the validation clientname: [chef-validator]: keep the default option
- location of the validation key: [/etc/chef-server/chef-validator.pem]: type the new location given

- the path to a chef repository (or leave blank): type the location chosen in the previous section
- password for the new user: type the password you have in mind

It is possible that the first time you got an error due to the autosigned-certificate of the chef-server. If this is the case, please follow the instructions you have in the screen and type 'knife ssl fetch' to accept this certificate.

Once you have a configured client, you can run the CLI. Just one:

```
knife client list
```

```
knife user list
```

Puppet

To install Puppet component, please refer to the following Puppet Installation Guide at [https://github.com/telefonicaid/fiware-puppetwrapper/blob/develop/doc/installation-guide.rst]

Requirements: Install PostgreSQL

The SDC node needs to have PostgreSQL installed in service mode and a database called SDC. For CentOS, these are the instructions:

Firstly, it is required to install the PostgreSQL [http://wiki.postgresql.org/wiki/YUM_Installation].

```
yum install postgresql postgresql-server postgresql-contrib
```

Start Postgresql

Type the following commands to install the postgresql as service and restarted

```
chkconfig --add postgresql
chkconfig postgresql on
service postgresql initdb
service postgresql start
```

Then, you need to configure postgresql to allow for accessing. In /var/lib/pgsql/data/postgresql.conf

```
listen_addresses = '0.0.0.0'
```

We need to create the sdc database. To do that we need to connect as postgres user to the PostgreSQL server and set the password for user 'postgres' using alter user as below:

```
su - postgres
postgres$ psql postgres postgres;
psql (8.4.13)
Type "help" for help.
postgres=# alter user postgres with password '<postgres-password>';
postgres=# create database sdc;
postgres=# grant all privileges on database sdc to postgres;
postgres=#\q
exit
```

where <postgres-password> is the passowrd for postgres user.

In /var/lib/pgsql/data/pg_hba.conf, change the table at the end of the file to look like:

```
#TYPE   DATABASE  USER        CIDR-ADDRESS          METHOD
#"local" is for Unix domain socket connections only
local   all       all                               ident
# IPv4 local connections:
host    all       all         127.0.0.1/32          md5
# IPv6 local connections:
host    all       all         ::1/128               md5
```

Restart the postgres

```
service postgresql restart
```

Check that the database has been created correctly:

```
$ su - postgres
postgres$ cd /opt/fiware/sdc-/resources
$ psql postgres postgres
postgres=#\c sdc
postgres=# \i db-initial.sql
postgres=# \i db-changelog.sql
exit
```

Then we need to create the database tables for the sdc. To do that obtain the files from [https://github.com/telefonicaid/fiware-sdc/blob/develop/migrations/src/main/resources] and execute

```
$ psql -d sdc -a -f db-initial.sql
$ psql -d sdc -a -f db-changelog.sql
```

Configure SDC application Once the prerequisites are satisfied, you change the context file. To do that, change sdc.xml found in distribution file and store it in folder $SDC_HOME/webapps/.

See the snipet bellow to know how it works:

```
<New id="sdc" class="org.eclipse.jetty.plus.jndi.Resource">
    <Arg>jdbc/sdc</Arg>
    <Arg>

        <New class="org.postgresql.ds.PGSimpleDataSource">
            <Set name="User"> <database user> </Set>
            <Set name="Password"> <database password> </Set>
            <Set name="DatabaseName"> <database name>   </Set>
            <Set name="ServerName"> <IP/hostname> </Set>
            <Set name="PortNumber">5432</Set>
        </New>

    </Arg>
</New>
```

Configuring the SDC as service Once we have installed and configured the SDC, the next step is to configure it as a service. To do that just create a file in /etc/init.d/fiware-sdc with the following content

```
#!/bin/bash
# chkconfig: 2345 20 80
# description: Description comes here....
# Source function library.
. /etc/init.d/functions
start() {
```

```
    /opt/fiware-sdc/bin/jetty.sh start
}
stop() {
    /opt/fiware-sdc/bin/jetty.sh stop
}
case "$1" in
    start)
        start
    ;;
    stop)
        stop
    ;;
    restart)
        stop
        start
    ;;
    status)
        /opt/fiware-sdc/bin/jetty.sh status
    ;;
    *)
        echo "Usage: $0 {start|stop|status|restart}"
esac
exit 0
```

Now you need to execute:

```
chkconfig --add fiware-sdc
chkconfig fiware-sdc on
service fiware-sdc start
```

The configuration of SDC is in configuration_properties table. There, it is required to configure:

- openstack-tcloud.keystone.url: This is the url where the keystone-proxy is deployed
- openstack-tcloud.keystone.user: the admin user
- openstack-tcloud.keystone.password: the admin password
- openstack-tcloud.keystone.tenant: the admin tenant
- sdc_manager_url: the final url, mainly http://sdc-ip:8080/sdc

The updates of the columns are done in the following way

```
su - potgres

postgres$ psql -U postgres -d sdc
Password for user postgres: <postgres-password-previously-chosen>

postgres=# UPDATE configuration_properties SET value='<the value>'
where key='sdc_manager_url';

postgres=# UPDATE configuration_properties SET value='<the value>'
where key='openstack-tcloud.keystone.user';

postgres=# UPDATE configuration_properties SET value='<the value>'
where key='openstack-tcloud.keystone.pass';

postgres=# UPDATE configuration_properties SET value='<the value>'
where key='openstack-tcloud.keystone.tenant';
```

```
postgres=# UPDATE configuration_properties SET value='<the value>'
where key='openstack-tcloud.keystone.url';
```

The last step is to create a sdc client in the chef-server, so that, the SDC can communicate with the chef-server. To do that, we can use the chef-server-web-ui, which is usually deployed on https://chef-server-ip, go to https://chef-server-ip/clients and create a sdc client as administrator. Then, it is required to copy the private key.

In the sdc machine, it is required to copy this private key in /etc/chef/sdc.pem (you can configure the path also in the properties)

Register SDC application into keystone The last step involves to regiter the SDC, chef-server, puppetwrapper and puppetmaster endpoints into the keystone endpoint catalogue. To do that, you should write into the config.js in the keystone-proxy the following lines:

```
 {"endpoints": [
    {"adminURL": "sdc-base-url",
    "region": "myregion",
    "internalURL": "sdc-base-url",
    "publicURL": "sdc-base-url"
    }
    ],
    "endpoints_links": [],
    "type": "sdc",
    "name": "sdc"
},
{"endpoints": [
    {"adminURL": "chef-server-url",
    "region": "myregion",
    "internalURL": "chef-server-url",
    "publicURL": "chef-server-url"
    }
    ],
    "endpoints_links": [],
    "type": "chef-server",
    "name": "chef-server"
},
{"endpoints": [
    {"adminURL": "puppet-wrapper-url"
     "region": "myregion"
     "internalURL": "puppet-wrapper-url"
     "publicURL": "puppet-wrapper-url"
    }
    ],
    "endpoints_links": [],
    "type": "puppetwrapper",
    "name": "puppetwrapper"
},
{"endpoints": [
    {"adminURL": "puppet-master-url"
     "region": "myregion"
     "internalURL": "puppet-master-url"
     "publicURL": "puppet-master-url"
    }
    ],
    "endpoints_links": [],
    "type": "puppetmaster",
    "name": "puppetmaster"
},
```

where myregion should be the name of the openstack region defined and puppet-wrapper-url, chef-server-url, sdc-base-url are typically urls of the form:

```
puppet-wrapper-url = https://puppetwrapper-ip:port/puppetwrapper/
sdc-base-url = https://sdc-ip:port/sdc/rest
chef-server-url = http://chef-server-ip:port
```

Creating images sdc-aware

The images to be deployed by the SDC, should have some features, like to have the chef-client installed and configured correctly with the chef-server. In the roadmap, it is considered to avoid all this process and to make possible any image to be SDC-aware, installing and configuring everything in booting status.

```
mkdir /etc/chef
mkdir /var/log/chef
curl -L https://www.opscode.com/chef/install.sh | bash
```

You should copy the chef-validator.pem from the chef-server into /etc/chef

Then, it is required to create a file called client.rb in /etc/chef. The validation.pem should be obtained from the chef-server in the folder /etc/chef-server and its called chef-validator.pem and rename to validation.pem in the /etc/chef folder of the image

```
log_location            "/var/log/chef/client.log"
ssl_verify_mode         :verify_none
validation_client_name "chef-validator"
validation_key          "/etc/chef/validation.pem"
client_key              "/etc/chef/client.pem"
chef_server_url         "https://cher-server-ip"
```

Finally, to start chef-client in boot time

```
chef-client -i 60 -s 6
```

Configuring the HTTPS certificate

The service is configured to use HTTPS to secure the communication between clients and the server. One central point in HTTPS security is the certificate which guarantee the server identity.

Quickest solution: using a self-signed certificate

The service works "out of the box" against passive attacks (e.g. a sniffer) because a self-signed certificated is generated automatically when the RPM is installed. Any certificate includes a special field call "CN" (Common Name) with the identity of the host: the generated certificate uses the host IP as identity .

The IP used in the certificate should be the public IP (i.e. the floating IP). The script, which generates the certificate, knows the public IP asking to an Internet service (http://ifconfig.me/ip). Usually this obtains the floating IP of the server, but of course it will not work without a direct connection to Internet.

If you need to regenerate a self-signed certificate with a different IP address (or better, a convenient configured hostname), please run:

```
/opt/fiware-sdc/bin/generateselfsigned.sh myhost.mydomain.org
```

By the way, the self-signed certificate is at /etc/keystorejetty. This file will not be overwritten although you reinstall the package. The same way, it will not be removed automatically if you uninstall the package.

Advanced solution: using certificates signed by a CA

Although a self-signed certificate works against passive attack, it is not enough by itself to prevent active attacks, specifically a "man in the middle attack" where an attacker try to impersonate the server. Indeed, any browser warns user against self-signed certificates. To avoid these problems, a certificate conveniently signed by a CA may be used.

If you need a certificate signed by a CA, the more cost effective and less intrusive practice when an organization has several services is to use a wildcard certificate, that is, a common certificate among all the servers of a DNS domain. Instead of using an IP or hostname in the CN, an expression as ".fiware.org" is used.

This solution implies:

- The service must have a DNS name in the domain specified in the wildcard certificate. For example, if the domain is ".fiware.org" a valid name may be "sdc.fiware.org".
- The clients should use this hostname instead of the IP
- The file /etc/keystorejetty must be replaced with another one generated from the wildcard certificate, the corresponding private key and other certificates signing the wild certificate.

It is possible that you already have a wild certificate securing your portal, but Apache server uses a different file format. A tool is provided to import a wildcard certificate, a private key and a chain of certificates, into /etc/keystorejetty:

```
# usually, on an Apache installation, the certificate files are at /etc/ssl/private
/opt/fiware-sdc/bin/importcert.sh key.pem cert.crt chain.crt
```

If you have a different configuration, for example your organization has got its own PKI, please refer to: http://docs.codehaus.org/display/JETTY/How%2bto%2bconfigure%2bSSL

1.3.5 Sanity Check procedures

Sanity Check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

End to End testing

Although one End to End testing must be associated to the Integration Test, we can show here a quick testing to check that everything is up and running. It involves to obtain the product information storaged in the catalogue. With it, we test that the service is running and the database configure correctly.

```
https://{SDC\_IP}:{port}/sdc/rest
```

The request to test it in the testbed should be

```
    curl -v -k -H 'Access-Control-Request-Method: GET'
    -H 'Content-Type: application xml' -H 'Accept: application/xml'
    -H 'X-Auth-Token: 5d035c3a29be41e0b7007383bdbbec57'
    -H 'Tenant-Id: 60b4125450fc4a109f50357894ba2e28' -X GET
    'https://localhost:8443/sdc/rest/catalog/product'
```

the option -k should be included in the case you have not changed the security configuration of SDC. The result should be the product catalog.

If you obtain a 401 as a response, please check the admin credentials and the connectivity from the sdc machine to the keystone (openstack-tcloud.keystone.url in configuration_properties table)

List of Running Processes

Due to the SDC basically is running over jetty, the list of processes must be only the Jetty and PostgreSQL. If we execute the following command:

```
ps -ewF | grep 'postgres\|jetty' | grep -v grep
```

It should show something similar to the following:

```
    postgres  2396     1  0 58141  9228   0 11:51 ?        00:00:00 /usr/bin/postgres
     -D /var/lib/pgsql/data -p 5432
    postgres  2397  2396  0 47554  1224   0 11:51 ?        00:00:00 postgres:
     logger process
    postgres  2399  2396  0 58167  4400   0 11:51 ?        00:00:00 postgres:
     checkpointer process
    postgres  2400  2396  0 58141  1652   0 11:51 ?        00:00:00 postgres:
     writer process
    postgres  2401  2396  0 58141  1416   0 11:51 ?        00:00:00 postgres:
     wal writer process
    postgres  2402  2396  0 58349  2944   0 11:51 ?        00:00:00 postgres:
     autovacuum launcher process
    postgres  2403  2396  0 48110  1720   0 11:51 ?        00:00:00 postgres:
     stats collector process
    root      2859     1  0 599252 884004 0 11:59 ?        00:00:29 java
    -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8585
    -Dspring.profiles.active=fiware -Xmx1024m -Xms1024m
    -Djetty.state=/opt/fiware-sdc/jetty.state -Djetty.logs=/opt/fiware-sdc/logs
    -Djetty.home=/opt/fiware-sdc -Djetty.base=/opt/fiware-sdc -Djava.io.tmpdir=/tmp
    -jar /opt/fiware-sdc/start.jar jetty-logging.xml jetty-started.xml
```

Network interfaces Up & Open

Taking into account the results of the ps commands in the previous section, we take the PID in order to know the information about the network interfaces up & open. To check the ports in use and listening, execute the command:

```
netstat -p -a | grep $PID
```

Where $PID is the PID of Java process obtained at the ps command described before, in the previous case 2396 (jetty) and 2859 (postgresql). The expected results for the postgres process must be something like this output:

```
tcp      0   0 0.0.0.0:postgres   0.0.0.0:*         LISTEN      2396/postgres
udp6     0   0 localhost:59289    localhost:59289   ESTABLISHED 2396/postgres
unix  2   [ ACC ]   STREAM   LISTENING   35218   2396/postgres
/var/run/postgresql/.s.PGSQL.5432
unix  2   [ ACC ]   STREAM   LISTENING   35220   2396/postgres
/tmp/.s.PGSQL.5432
```

and the following output for the jetty process:

```
tcp    0   0 0.0.0.0:8585        0.0.0.0:*          LISTEN      2859/java
tcp6   0   0 [::]:pcsync-https   [::]:*            LISTEN      2859/java
unix  2    [ ]       STREAM    CONNECTED    48445   2859/java
unix  2    [ ]       STREAM    CONNECTED    62299   2859/java
unix  3    [ ]       STREAM    CONNECTED    48380   2859/java
```

Databases

The last step in the sanity check, once that we have identified the processes and ports is to check the different databases that have to be up and accept queries. For he first one, if we execute the following commands:

```
psql -U postgres -d sdc
```

For obtaining the tables in the database, just use

```
sdc=# \dt

                List of relations
 Schema |             Name               | Type  |  Owner
--------+--------------------------------+-------+----------
 public | artifact                       | table | postgres
 public | artifact_attribute             | table | postgres
 public | attribute                      | table | postgres
 public | configuration_properties       | table | postgres
 public | installableinstance            | table | postgres
 public | installableinstance_attribute  | table | postgres
 public | installablerelease             | table | postgres
 public | metadata                       | table | postgres
 public | nodecommand                    | table | postgres
 public | os                             | table | postgres
 public | product                        | table | postgres
 public | product_attribute              | table | postgres
 public | product_metadata               | table | postgres
 public | productinstance                | table | postgres
 public | productrelease                 | table | postgres
 public | productrelease_os              | table | postgres
 public | productrelease_productrelease  | table | postgres
 public | task                           | table | postgres

(18 rows)
```

1.3.6 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

Resource availability

The resource availability should be at least 1Gb of RAM and 6GB of Hard disk in order to prevent enabler's bad performance. This means that bellow these thresholds the enabler is likely to experience problems or bad performance.

Resource consumption

State the amount of resources that are abnormally high or low. This applies to RAM, CPU and I/O. For this purpose we have differentiated between:

- Low usage, in which we check the resources that the Tomcat requires in order to load the PaaS Manager.
- High usage, in which we send 100 concurrent accesses to the PaaS Manager.

The results were obtained with a top command execution over the following machine configuration:

Name	Type
Type Machine	Virtual Machine
CPU	1 core @ 2,4Ghz
RAM	1,4GB
HDD	9,25GB
Operating System	CentOS 6.3

The results of requirements both RAM, CPU and I/O to HDD is shown in the following table:

Resource Consumption	Low UsageType	High Usage
RAM	1GB ~ 63%	3GB ~ 78%
CPU	0,8% of a 2400MHz	90% of a 2400MHZ
I/O HDD	6GB	6GB