FIWARE-GlanceSync: Glancesync

Release

September 13, 2016

Contents

GlanceSync is a command line tool and a server with API to solve the problem of the images synchronisation between regions. It synchronises glance servers connected in the FIWARE Lab, in different regions taking the base of a master region.

This project is part of FIWARE.

CHAPTER 1

Documentation

GitHub's README provides a whole documentation:

1.1 GlanceSync - Glance Synchronization Component

1.1.1 Introduction

This is the code repository for the GlanceSync component, the FIWARE Ops tool used to synchronise the glance images in the different Glance servers connected in the FIWARE Lab.

This project is part of FIWARE.

Although this component has been developed for FIWARE, the software is highly configurable, do not have special requirements beyond OpenStack libraries and may be used with any other project or as a generic tool to synchronise images. Moreover, all the OpenStack interface is in a module and it is possible to adapt the code to support other

platforms.

There is also the possibility to launch the application like a service which allows you synchronise the different regions using a specific FIWARE Lab administrator account. It could be used with a specific token generated by the FIWARE Lab Keystone service.

Any feedback on this documentation is highly welcome, including bugs, typos or things you think should be included but are not. You can use github issues to provide feedback.

Top

1.1.2 Overall description

GlanceSync is a command line tool and a server with API to solve the problem of the images synchronisation between regions. It synchronises glance servers in different regions taking the base of a master region. It was designed for FIWARE project, but it has been expanded to be useful for other users or projects.

GlanceSync synchronises all the images with certain metadata owned by a tenant from a master region to each other region in a federation (or a subset of them). This feature works out of the box without configuration. It requires only the same set of environment variables, which are needed to contact the keystone server, than the glance tool. It is also possible to set these parameters in a file instead of using environment variables. Furthermore, any option in the configuration file can be provided via command line, too.

GlanceSync synchronisation algorithm (i.e. the method to determine if a master image must be synchronised to the other regions) is configurable. By default all public images are synchronised, but it is enough with adding a line in the configuration file to synchronise only the public images with certain metadata (e.g. federated_image=True).

GlanceSync supports also the synchronisation to regions which do not use the same keystone server than the master region and therefore require their own set of credentials. The regions are grouped by *targets*: two regions may be in the same *target* if they use the same credential (therefore, their glance servers are registered in the same keystone server). The only mandatory *target* is the `master` target, where the master region is. Most of the GlanceSync configuration, including the criteria to select which images are synchronised, is defined at target level. It is okay to create several targets using the same credential, for example if some regions only share a minimal set of images and others have a broader list.

GlanceSync by default does not replace existing images. If an image checksum is different between the region to synchronise and the master region, a warning is emitted. The user has the option of forcing the overwriting of a specific image (optionally renaming the old one) including the checksums in a configuration file, using a whilelist or a blacklist.

When the remote image has the same content than the master image, but the metadata differs, GlanceSync updates the metadata, but only a limited set, to avoid overwriting properties considered as local in that glance server. Also the system property `is_public` is updated.

GlanceSync has special support for *AMI* (Amazon Machine Image). Amazon images include a reference to a kernel image (*AKI*) and to a ramdisk image (*ARI*), but they are named by UUID. Therefore GlanceSync has to update this fields to reflect the UUIDs in that particular region.

GlanceSync supports marking an image as obsolete, adding the suffix *_obsolete*. An obsolete image is not synchronisable, but it is managed in a special way: when an image is renamed, the change is propagated to the other regions. Also the visibility of the image is propagated (i.e. if the master image is marked as private, is made private in all the other regions).

The idea of marking the obsoleted images, is allow the administrator of the regions to make a decision about them. These images are not part of set of mandatory images in a federation anymore, but perhaps are in use by their local users.

About UUIDs and image names

This tool does not synchronise using UUID but names (i.e. an image has the same name in all regions, but not the same UUID). Using a UUID to synchronise is generally a bad idea, because some problems may arise with the restriction that a UUID must be unique. Be aware that it is not possible to replace the content of a image, without creating a new one and the old UUID may not be reused. If something similar to an UUID is required, it is better to use a metadata field to simulate it.

The downside of using names, is that a region may have more than a image with the same name. This is specially challenging, when there is more than one image in a destination target, with the name of the image to synchronise. In this situation, GlanceSync takes the first image that is found with the same checkum (or absolutely the first image that is found if there is not a checksum match) and prints a warning for each duplicated image detected. Master images with duplicated names are not synchronised and a warning is printed.

Image names with duplicated names are easy to avoid, with one serious exception: when ordinary users can publish their images as public (shared), the risk of collision increases and escapes of the control of the user. To avoid this, GlanceSync ignore the images of other tenants by default. Anyway, this is a general problem, not only a synchronisation problem, due to more that one image with the same name is very confusing to users that want to use them. Therefore it is better to restrict the publication of shared images.

How it works

First GlanceSync gets a list of the images in the master region. Then runs the algorithm with each specified region (or all the regions registered in the same keystone server than the master region, if not specified). If an error occurs within a region synchronisation, GlanceSync does not run more operations in that region and jumps to the next one.

For each region, GlanceSync starts getting a list of its images. Then calculates with images should be synchronised to this region (this is detailed in the next section).

It some images has metadata pending, it updates them. After updating the metadata, the missing images are upload. The uploading is by size order, this way when there is a problem in the glance server it will be detected earlier with the smallest image (e.g. when there is not enough space). Another reason to start with the smallest first, is because AMI images; the kernel and ramdisk are also images and because they are smaller, are uploaded before the AMI image that needs them.

The last step is to update the kernel/ramdisk fields in AMI images when the kernel/ramdisk images has been uploaded during this synchronisation session.

When a image with the same name is already present in the destination region, Glancesycn checks it they are the same comparing the checksums. When they are different, the following algorithm is applied:

1. Is the checksum in the `dontupdate` list? Print a warning only
2. Is the checksum in the `rename` list? Rename old image (adding the *.old* suffix), change it to private, and upload the master region's image
3. Is the checksum in the `replace` list? Replace the old image with the master region's image
4. Does the parameter `replace` include the keyword *any*? Rename old image and upload the master region's image
5. Does the parameter `rename` include the keyword *any*? Replace the old image with the master region's image
6. Otherwise: print a warning. The user should take an action and fill `dontupdate`, `replace` or `rename` parameters. In the meanwhile, the image is considered *stalled* and it is not synchronised at all.

How the images to be synchronised are selected

There are three parameters in the configuration that affects which images are selected: *forcesync*, *metadata_condition* and *metadata_set*. All of them can be different for each target; when most targets use the same selection criteria, an option is to put this options in the *DEFAULT* section.

This is the algorithm to determine if an image is synchronisable:

1. images with the *_obsolete* suffix, are never synchronised
2. images of other tenants are never synchronised
3. images with duplicated names are never synchronised, to avoid ambiguity.
4. if the UUID of the image is included in `forcesync`, then it is synchronised unconditionally, even if the image is not public.
5. if `metadata_condition` is defined, it contains python code that is evaluated to determine if the image is synchronised. The code can use two variables: image, with the information about the image and `metadata_set`, with the content of that parameter. The more interesting field of image is `user_properties`, that is a dictionary with the metadata of the image. Other properties are *id*, *name*, *owner*, *size*, *region*, *is_public*. The image may be synchronised even if it is not public, to avoid this, check `image.is_public` in the condition. If metadata_set is not defined and `image.is_public`, then the image will be synchronised with all `user_properties`.
6. if `metadata_condition` is not defined, the image is public, and `metadata_set` is defined, the image is synchronised if some of the properties of `metadata_set` is on `image.user_properties`.
7. if `metadata_condition` is not defined, the image is public, and `metadata_set` is not defined, the image is synchronised
8. otherwise, the image is not synchronised.

For example, to synchronise the images in FIWARE Lab, the best choice is setting `metadata_set=nid, sdc_aware, type, nid_version`, because all the images to be synchronised has at least one of those properties.

A trip to synchronise also the images specified in a white list is combine the parameter *forcesyncs* with `metadata_condition=False`

The parameter `metadata_set` has another function. It is used to determine how the metadata is updated in the remote image. If it is not defined, all the metadata is copied from the master image, otherwise, only the properties in `metadata_set` are copied. Be aware that system property *is_public* must not be included in `metadata_set`, because it is not a user property but a system one. Anyway, *is_public* is unconditionally synchronised.

How the obsoleted images are managed

An obsolete image is an image with the *_obsolete* suffix. When an image is marked as obsoleted is not synchronised anymore and therefore it is not upload to regions where it is not present. However, if an image exists in the remote region with the same name but without the suffix, it is renamed and the visibility is updated with the value on the master region. Also the properties specified in *obsolete_syncprops*, if any, are synchronised. The synchronisation of the properties and the visibility is also managed when there is a image in the region to synchronise that is already renamed but without the other changes propagated.

There are some checks to do before propagating the changes of an obsoleted image:

- Are the two images the same? The checksums are compared and only if they are the same the change is done.
- Is the image in the region to synchronise a public image of another tenant? in this case do not touch the image.

- Is there an image with the same name but without the suffix also in the master region and is synchronisable? In this case the image will be synchronised normally without taking in consideration the obsolete image.

Usually obsoleted images are made private, because are not supported anymore. It is possible to restore an image as public for local use after renaming or changing the tenant (to avoid that it is made private again automatically), but before this is important to look out more about the security status of the image.

The treatment of obsolete images can be disabled for a *target* with *support_obsolete_images=False*. This flag affects the image renaming and the metadata updating, but anyway images with '_obsolete' suffix are never synchronised.

Top

1.1.3 Build and Install

Requirements

GlanceSync is designed to run with a mounting point with the images, because it reads the images that are stored directly in the filesystem. Usually this directory is /var/lib/glance/images.

The following software must be installed (e.g. using apt-get on Debian and Ubuntu, or with yum in CentOS):

- Python 2.7
- pip
- virtualenv

Installation

The recommend installation method is using a virtualenv. Actually, the installation process is only about the python dependencies, because the python code do not need installation.

1. Create a virtualenv 'glancesyncENV' invoking *virtualenv glancesyncENV*
2. Activate the virtualenv with *source glancesyncENV/bin/activate*
3. Install the requirements running *pip install -r requirements.txt –allow-all-external*

Now the system is ready to use. For future sessions, only the step2 is required.

Top

1.1.4 Server Configuration

There is the possibility to execute the glancesync like a service. You should launch the server by executing the `run.py` process. You can see in the Running section how to launch the server. In this section we explain the configuration file that have to be defined to work with the GlanceSync Service. Last but not least keep in bear that you will need also configure the client component if you want to launch the core module of synchronization component.

GlanceSync server configuration file

The server have to be launched with a configuration file. By default, the service will take the values either from environment variables or from files located in `/etc/fiware.d`. The name of the files MUST be `fiware-glancesync.cfg` and `fiware-glancesync-logging.cfg`. The options that we take are the following:

1) In the first case, the application try to see if there is defined the variables GLANCESYNC_SETTINGS_FILE, GLANCESYNCAPP_DATABASE_PATH, GLANCESYNCAPP_CONFIG and GLANCESYNC_LOGGING_SETTINGS_FILE. This environment variables will have the location of the configuration files, you can specify them using the following commands

```
$ export GLANCESYNC_SETTINGS_FILE=/Users/foo/fiware-glancesync/app/settings/fiware-glancesync.cfg
$ export GLANCESYNC_LOGGING_SETTINGS_FILE=/Users/foo/fiware-glancesync/app/settings/fiware-glancesync
$ export GLANCESYNCAPP_DATABASE_PATH=/Users/foo/glancesyncENV/lib/python2.7/site-packages/
                                     fiware_glancesync.egg/fiwareglancesync/
$ export GLANCESYNCAPP_CONFIG=/Users/foo/glancesyncENV/lib/python2.7/site-packages/
                                     fiware_glancesync.egg/fiwareglancesync/app/config.py
```

2) If the GLANCESYNC_SETTINGS_FILE and GLANCESYNC_LOGGING_SETTINGS_FILE environment variables are not presented, the application will try to obtain the files from the directory /etc/fiware.d

If no one of the previos option is accomplished the server will launch an error message like the following:

```
ERROR: There is not defined GLANCESYNCAPP_CONFIG environment variable
       pointing to config.py path file
       Please correct at least one of them to execute the program.

ERROR: There is not defined GLANCESYNCAPP_DATABASE_PATH environment variable
       pointing to database path file
       Please correct at least one of them to execute the program.

ERROR: There is neither defined GLANCESYNC_LOGGING_SETTINGS_FILE environment variable pointing
       to fiware-glancesync-logging.cfg nor /etc/fiware.d/etc/fiware-glancesync-logging.cfg
       file. Please correct at least one of them to execute the program.

ERROR: There is neither defined GLANCESYNC_SETTINGS_FILE environment variable
       pointing to fiware-glancesync.cfg nor /etc/fiware.d/etc/fiware-glancesync.cfg
       file. Please correct at least one of them to execute the program.
```

Configure your own configuration files

The GlanceSync server has two configuration files:

- fiware-glancesync.cfg, this is the important one to configure the service and need some modifications
- fiware-glancesync-logging.cfg, this file is used to configure the logging system, it is not needed to

change the content that we have defined by default in the publication of the component.

Related to the first file, how we have mentioned, there is some parameters that have to be configured in order to execute correctly the service. For obvious reason they are not included in the repository:

- **KEYSTONE_URL**, service endpoint of the Keystone service in FIWARE Lab (it usually comes defined in

the installation of the component.

- **ADM_USER**, admin user in Keystone.
- **ADM_PASS**, password of the admin user
- **ADM_TENANT_ID**, tenant id of the admin user.
- **ADM_TENANT_NAME**, tenant name of the admin user (you have to provide either ADM_TENANT_ID or ADM_TENANT_NAME).
- **USER_DOMAIN_NAME**, user domain name, by default for an administrator account you can use the value Default.

Top

1.1.5 Client Configuration

Working without a configuration file

The tool can work without a configuration file or with an empty one. In this case, the following OpenStack environment variables must be filled with the administrator's credential: `OS_USERNAME`, `OS_PASSWORD`, `OS_AUTH_URL`, `OS_TENANT_NAME`, `OS_REGION_NAME`. The value of `OS_REGION_NAME` will be the master region (in FIWARE Lab this region is Spain2).

It is also possible to pass any configuration option using command line. For example, the following invocation runs a synchronisation taking from command line the parameters *master_region* in the *main* section and *metadata_set* in the *DEFAULT* section:

```
./sync.py --config main.master_region=Spain2 metadata_set=nid,type,sdc_aware,sdc_version
```

It is important to note that *–config* parameter expect any number of parameters separated by spaces. This is a problem if the list of regions are specified

after the *–config* parameter, because then the regions are parsed as part

of the *–config* parameter. The solution is passing the regions *before* the parameter or using the standard separator –:

```
# Wrong: region1 and region2 are interpreted as part of --config param
./sync.py --config main.master_region=Spain2 region1 region2
# Ok
./sync.py --config main.master_region=Spain2 -- region1 region2
# Ok
./sync.py region1 region2 --config main.master_region
```

The configuration file

The configuration used by the GlanceSync component is stored in the `/etc/fiware.d/glancesync.conf` file. However, this path may be changed with the environment variable *GLANCESYNC_CONFIG*.

The configuration file has a `main` section with some global configuration parameters and one section for each target (regions are grouped by targets, two regions are in the same targets if they use the same credential). The `master` section is the target where the master region is, that is, the region where are located the images to synchronise to the other regions.

Most of the configuration is defined at target level. If the same values are used in most or all the targets, an option is to set them in the DEFAULT section.

The only mandatory settings in the target sections, is the credential. It may be provided in two ways (in the case of `master` also it is possible to use the environment variables as explained in the previous section, even it is possible to combine both methods, for example to set only the password via environment variable):

- using the credential option. There are four values separated by commas: the first is the user, the second is the password encoded with base64, the third is the keystone URL and the fourth, the tenant name.
- using the options *user*, *password*, *tenant*, *keystone_url*.

If credentials are stored in the configuration file, it is convenient to make the file only readable by the user who invokes GlanceSync.

Example of a configuration file

The following is an example of a configuration file, with all the possible options auto explained in the comments. A configuration file like this can be generated invoking *fiwareglancesync/script/generated_config_file.py*

```
[main]

# Region where are the images in the "master" target that are synchronised to
# the other regions of "master" regions and/or to regions in other targets.
master_region = Spain

# A sorted list of regions. Regions that are not present are silently
# ignored. Synchronization is done also to the other regions, but first this
# list is revised and then the Regions are prefixed with "target:"
# This parameter is only used when running synchronisation without parameters
# or the region list includes a 'target' (e.g. 'master:' that is expanded to
# the regions in master but the specified in ignore_regions). When the full region
# list is provided explicitly via command line, the order of
# the parameters is used instead.
preferable_order = Trento, Lannion, Waterford, Berlin, Prague

# The maximum number of simultaneous children to use to do the synchronisation.
# Each region is synchronised using a children process, therefore, this
# parameter sets how many regions can be synchronised simultaneously.
# The default value, max_children = 1, implies that synchronisation is fully
# sequential. Be aware that you need also to invoke the sync tool with the
# --parallel parameter.
#
max_children = 1

# The folder where the master images are (the filename is the UUID of the
# image in the master region). The default value is the folder where the
# Glance server stores the images.
images_dir = /var/lib/glance/images

[DEFAULT]

# Values in this section are default values for the other sections.

# the files with this checksum will be replaced with the master image
# parameter may be any or a CSV list (or a CSV list with 'any' at the end)
# replace = 9046fd22131a96502cb0d85b4a406a5a

# the files with this checksum will be replaced with the master image,
# but the old image will be preserved renamed (using same name, but with
# .old extension) and made private.
# parameter may be any or a CSV list (or a CSV list with 'any' at the end)
# rename = any

# If replace or rename is any, don't update nor rename images with some of
# these checksums
# dontupdate =

# List of UUIDs that must be synchronised unconditionally.
#
# This is useful for example to pre-sync images marked as private

forcesyncs = 6e240dd4-e304-4599-b7d8-e38e13cef058
```

```
# condition to evaluate if the image is synchronised.
# image is defined, as well as metadata_set (see next parameter).
# Default condition is:
#  image.is_public and (not metadata_set or metadata_set.intersection(image.user_properties))

metadata_condition = image.is_public and\
 ('nid' in image.user_properties or 'type' in image.user_properties)

# the list of userproperties to synchronise. If this variable is undefined, all
# user variables are synchronised.
metadata_set = nid , type, sdc_aware, nid_version

# When the software asks for the list of images in a region, it gets both the
# images owned by the tenant and the public images owned by other tenants.
# If this parameter is true (the default and recommended value), only the
# tenant's images are considered. This implies that it can exist after the
# synchronisation a new image with the same name that a public one from other
# user. It could be very confusing (actually, a warning is printed when it is
# detected), but usually it is not recommend to work with images from other
# tenants. To find out more about this, see 'About UUIDs and image names' in
# the documentation.
#
# This parameter only affects to the list of images obtained from the regional
# servers. From master region only the tenant's images are considered.
only_tenant_images = True

# When this option is true (the default), the renaming and metadata updating of
# obsolete images is activate. See the documentation for details.
support_obsolete_images = True

# These are the properties that are synchronised (in addition to is_public
# and the name) in obsolete images, when support_obsolete_images is True.
obsolete_syncprops = sdc_aware

# Timeout to get the image list from a glance server, in seconds. Default
# value is 30 seconds.
list_images_timeout = 30

# API required to contact with the keystone server. If this parameter is True,
# then version 3 of the API is used. Otherwise, the version 2 is used
use_keystone_v3 = False

[master]

# This is the only mandatory target: it includes all the regions registered
# in the same keystone server than the master region.
#
# credential set: user, base64(password), keystone_url, tenant_name
# as alternative, options user, password, keystone_url and tenant can be used
# only with master target, it is possible also to set the credential using
# OS_USERNAME, OS_PASSWORD, OS_TENANT_NAME, OS_AUTH_URL (or even mixing this
# environment variables with parameters user, password, etc.)
credential = user,W91c2x5X2RpZF95b3VfdGhpbmtfdGhpc193YXNfdGhlX3JlYWxfcGFzc3dvcmQ/,http://server:4730/

# This parameter is useful when invoking the tool without specifying which
# images to synchronise or when the list includes a "target" without a region
# (e.g. master:). In this case it is expanded with the list of regions in that
# target except the included in ignore_regions
```

```
ignore_regions = Spain1

[experimental]

# Another
credential = user2,W91c2x5X2RpZF95b3VfdGhpbmtfdGhpc193YXNfdGhlX3JlYWxfcGFzc3dvcmQ/,http://server2:473
metadata_condition = image.is_public and image.user_properties.get('type', None) == 'baseimages'
```

This configuration file defines two *targets*: `master` and `experimental`. The first one synchronises all the public images with properties *nid* and/or *type* defined. The last one only synchronises images with `type=baseimages`

Security consideration

GlanceSync does not require *root* privileges. But at this version it requires read-only access to image directory `/var/lib/glance/images` (or making available a copy of all these files, or at least the subset that may be synchronised, in other path and then set the option *images_dir*)

It is strongly recommended:

- creating an account to run GlanceSync only
- creating a configuration file only readable by the GlanceSync account. This is because the credentials should not be exposed to other users.

Top

1.1.6 Client Running

Basic use

Once installed all the dependencies, there is a way to run GlanceSync manually from the command line invoking the `sync.py` tool inside the GlanceSync distribution.

When `./sync.py` is invoked without parameters, it synchronises the images from the master region to all the other regions with a glance endpoint registered in the keystone server (except the ones, if any, specified as a comma separated list in the `ignore_regions` parameter, inside the `master` section). The command can also receive as parameters the regions to synchronise. It is possible also to specify a target name and the suffix *:*; this way it is expanded to all the regions in that target (e.g. if there are two regions, *regionA* and *regionB* in target *target1*, then *target1:* is expanded with *target1:regionA target1:regionB*)

Advanced use

By default, GlanceSync synchronises regions one by one. When the command line option *–parallel* is passed, GlanceSync synchronised several regions in parallel. The number or regions synchronised at the same time is determined by the parameter max_children in the main section. Default value is 1 (no parallel). When synchronisation runs on parallel, a directory with the pattern *sync_<year><month>_<hour><minute>* is created. Inside this, it is a file for each region with the log of the synchronisation process.

The option *–dry-run* shows the changes needed to synchronise the images, but without doing the operations actually.

The option *–show-regions* shows all the regions available in all the targets defined in the configuration file.

The option *–make-backups* creates a backup of the medatada of the images in the regional Glance servers, instead of running the synchronisation.

It is possible to override any parameter of the configuration file, using the option *–config*. Be aware that the way of setting several parameters is separating them with spaces (e.g. *–config option1=value1 option2=value2*)

Finally, the option *–show-status* is to obtain a report about the synchronisation status of the regions. A more detailed information of this is provided in the *Checking status* section.

As pointed, GlanceSync can synchronised also from the master region to regions that do not use the same keystone server. A *target* is a namespace to refer to the regions sharing a credential. The `master` target is the one where the master region is. Each target has a section with its name in the configuration file, to specify the credential and optionally other configuration (most of the parameters are local to each target).

The way to synchronise to regions that are in other *target*, is to specified the region with the preffix `<target_name>:`. For example, to synchronise to region Trento and Berlin2, both in the same keystone server than the master region, but also to RegionOne and RegionTwo, registered in target *other* the following command must be invoked:

```
./sync.py Trento Berlin2 other:RegionOne other:RegionTwo
```

Note that the *master:* prefix may be omitted.

Making a backup of metadata

The option *–make-backups* create a backup of the metadata in the specified regions and in the master region. This is useful for example for debugging or testing, because GlanceSync supports the use of a mock that reads files likes these as input instead of contacting to the real servers. The mock is also used for testing real scenarios.

The backup is created in a directory named *backup_glance_* with the date and time as suffix. There is a file for each region (the name is backup_<region>.csv) and inside the file a line for each image. The following fields are included:

- the region name
- the image name
- the UUID of the image in the region
- the status of the image (the OK status is 'active')
- the size in bytes
- the checksum
- the tenant id of the owner (a.k.a. project id)
- a boolean indicating if the image is Public
- a dictionary with the user properties

Only the information about public images/ the images owned by the tenant, can be obtained. This is a limitation of the glance API: even the administrator does not get a list of private images of other users.

Using a mock with a backup

It is possible to use the result of a backup (optionally after changing the contents) for testing different scenarios.

Supposing the backup directory *backup_glance_2015-11-17T12:54:26.117838* is renamed to *scenario1*. After invoking this line, instead of operating with the real servers, a mock with metadata saved in *persistent_data* folder is used:

```
eval $(glancesync/glancesync_serverfacade_mock.py  --path persistent_data scenenario1)
export PYTHONPATH=glancesync
```

The created scenario is persistent, that is, is possible to invoke *sync.py –show-status* before and after running the synchronisation for checking that the state has changed.

The mock uses as tenant_id (this is important to compare the owner of the files) the paremeter *tenant_id* if defined in the configuration, otherwise *id* is added to the tenant_name as suffix.

To make test results deterministic, when a new image is created in the mock, the UUID is not random. The UUID's pattern is <*seq*>*$*<*image_name*> where seq is a number starting with 1 that guarantees the UUID uniqueness.

Checking status

In order to check the status of the synchronisation, use the following command:

```
./sync.py --show-status
```

This print the status of all the regions in the *master* target, that is, the region in the same keystone server than the master region. If `ignore_regions` is defined in the *master* configuration section, the specified regions are ignored.

Of course is also possible to check the status of any group of regions, for example, the call:

```
./sync.py --show-status Trento Mexico Gent target2:Region1 target2:Region2
```

It will show the status of the regions Trento, Mexico, Gent both in the *master* target, and the regions Region1 in Region2 defined in the *target2* target.

The output of command is a line for each image to be synchronised for each region. That is, in the last example, if 15 images are synchronised to the regions of *master* and 10 images to the regions of *target2*, then a total of 15*3 + 10*2 images are printed.

Each line is a CSV. The first field is the synchronisation status, the seconds is the region's name, and the third is the image name. This is an example:

```
ok,Prague,base_centos_6
ok,Prague,base_ubuntu_14.04
ok,Prague,base_ubuntu_12.04
ok,Prague,base_debian_7
ok,Prague,base_centos_7
pending_upload,experimental:Valladolid,base_centos_7
```

The synchronisation status can be classified in three categories: final status, error status and pending synchronisation status.

Final status

GlanceSync consider that there is no pending operations: the image is synchronised or marked as 'dontupdate'.

- ok: the image is fully synchronised
- ok_stalled_checksum: the image has a different checksum than master, but this checksum is included in parameter 'dontupdate'. Therefore the image will not be updated (content nor metadata)

Error status

There is an error condition that requires user intervention before trying again.

- error_checksum: there is an image, but with a different checksum and there is not a matching dontupdate, rename or replace directive. Action required: fill the checksum (or use any) with *dontupdate* or *rename* or *replace*.

- error_ami: the image requires a kernel or ramdisk that is not in the list of images to sync. Action required: ensure that the selection criteria include the kernel/ramdisk images.

Pending synchronisation status

The image needs synchronisation. Be aware that perhaps the image is on a pending status although GlanceSync execution has completed, because the glance server responded with an error. However, this is yet considered a pending status and not an error status, because it is not a problem that users must resolve by themselves.

- pending_metadata: there is an image with the right content (checksum), but metadata must be updated (this may include ramdisk_id and kernel_id)
- pending_upload: the image is not synchronised; it must be upload
- pending_replace: there is an image, but with different checksum. The image will be replaced
- pending_rename: there is an image, but with different checksum. The image will be replaced, but before this the old image will be renamed
- pending_ami: the image requires a kernel or ramdisk image that is in state *pending_upload*, *pending_replace* or *pending_rename*.

How use glancesync without access to images files

At the moment, GlanceSync is designed to run in the glance server of the master region, because it reads the images that are stored directly in the filesystem.

This may be an inconvenience, but a real issue is when the Glance backed does not use plain files (e.g. the Cepth backend) and therefore GlanceSync cannot read the files even when it is running at the glance server.

The following script can be used to pre-download the images required to synchronise the indicated regions to the folder specified by environment variable *GLANCE_IMAGES* (by default, `/var/lib/glance/images`) and then run the synchronisation:

```
#!/bin/bash

print_required_images_names() {
  ./sync.py --show-status $* | awk -v ORS=" " -F, \
   '/^pending_(upload|rename)/ {words[$3]++}
   END { for (i in words) print substr(i,1, length(i)-1) }'
}

get_id_from_name() {
 glance image-show $1 | awk -F\| \
 ' $2 ~ /^[ ]*id/ { sub(/[ ]+/,"",$3) ; print $3}'
}

GLANCE_IMAGES=${GLANCE_IMAGES:-/var/lib/glance/images}

# First, download the required images to $GLANCE_IMAGES
for name in $(print_required_images_names $*) ;
do
  id=$(get_id_from_name $name)
  echo $name $id
  if [ ! -f $GLANCE_IMAGES/$id ] ; then
    glance image-download --file $GLANCE_IMAGES/$id --progress $id
  fi
done
```

```
# run synchronisation
./sync.py $* --config images_dir=$GLANCE_IMAGES
```

Top

1.1.7 Server Running

There is several options to execute the GlanceSync server from the command line. You can obtain information of the different options running form the command line the following command:

```
$ python run.py

usage: run.py [-h] {gunicornserver,shell,db,runserver} ...

positional arguments:
  {gunicornserver,shell,db,runserver}
    gunicornserver      Run the GlanceSync server application within Gunicorn.
    shell               Runs a Python shell inside Flask application context.
    db                  Perform database migrations
    runserver           Runs the Flask development server i.e. app.run()

optional arguments:
  -h, --help            show this help message and exit
```

We go into details about the db, runserver and gunicornserver options.

Database configuration

If it is the first time that you use the component or if you need to make an upgrade of the database schemas, you will need to execute the commands related to the database management. You can obtain a help of the different operations just executing:

```
$ python run.py db -h
usage: Perform database migrations

Perform database migrations

positional arguments:
  {upgrade,heads,merge,migrate,stamp,show,current,edit,init,downgrade,branches,history,revision}
    upgrade             Upgrade to a later version
    heads               Show current available heads in the script directory
    merge               Merge two revisions together. Creates a new migration
                        file
    migrate             Alias for 'revision --autogenerate'
    stamp               'stamp' the revision table with the given revision;
                        don't run any migrations
    show                Show the revision denoted by the given symbol.
    current             Display the current revision for each database.
    edit                Edit current revision.
    init                Generates a new migration
    downgrade           Revert to a previous version
    branches            Show current branch points
    history             List changeset scripts in chronological order.
    revision            Create a new revision file.
```

```
optional arguments:
  -h, --help            show this help message and exit
```

This allows you to keep a revision of the database that we are using. The first time that you use the component you will need to create the database repository and initialize the revision of it. It can be done with the following commands (in order):

```
$ python run.py db init
$ python run.py db migrate
$ python run.py db upgrade
```

Run basic server

Once that we have initialized the database, we can launch the application, there is two possibilities. In this section, we see the easy way to launch the application running basically a python process. You can obtain help of the operation just executing:

```
$ python run.py runserver -h
usage: run.py runserver [-h] [-t HOST] [-p PORT] [--threaded]
                        [--processes PROCESSES] [--passthrough-errors] [-d]
                        [-r]

Runs the Flask development server i.e. app.run()

optional arguments:
  -h, --help            show this help message and exit
  -t HOST, --host HOST
  -p PORT, --port PORT
  --threaded
  --processes PROCESSES
  --passthrough-errors
  -d, --no-debug
  -r, --no-reload
```

How you can see almost all arguments are optionals, the HOST and PORT are defined in the `fiware-glancesync.cfg` file. You can execute the server just executing:

```
$ python run.py runserver
```

Run Gunicorn server

There is the possiblity to launch the service behind a Gunicorn HTTP Server. Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. You need to install this HTTP Server previously to execute the GlanceSync service. Take a look to the Gunicorn site to see how to install it. Keep in bear that you should use a version greater than 0.9.0.

After the installation of the HTTP server, you can execute the component. If you execute the following command you can obtain detailed information about the options that you have:

```
$ python run.py gunicornserver -h
usage: run.py gunicornserver [-h] [-H HOST] [-p PORT] [-w WORKERS]

Run the GlanceSync server application within Gunicorn.

optional arguments:
  -h, --help            show this help message and exit
  -H HOST, --host HOST  IP address or hostname of the Glancesync server.
```

```
  -p PORT, --port PORT  Port in which the GlanceSync server is running
  -w WORKERS, --workers WORKERS
                        Number of concurrent workers to be launched, usually
                        2*core numbers+1.
```

By default, HOST, PORT and WORKERS are defined in the configuration file, it is not necessary to specify them again here. So to run the service, just write the following line:

```
$ python run.py gunicornserver -h
```

Logging files

The current version of the GlanceSync service produce logging files that will be located in the sam directory of the application where you launch the application. It is defined with log rotate with allow to control the extension of the file. You can see details of the configuration of the log file in the `fiware-glancesync-logging.cfg` file. By default it will be named with **glancesync-api.log**. The sucessive rotate files will be numered adding a string from '.1' to '.3' to the previous file name (e.g. glancesync-api.log.1).

Top

1.1.8 API Overview

The GlanceSync offers a REST API, which can be used for synchronizing images in different regions. Please have a look at the API Reference Documentation section bellow.

API Reference Documentation

- FIWARE GlanceSync v1 (Apiary)

Top

1.1.9 Testing

Ent-to-end tests

To run the end-to-end tests, go to `test/acceptance` folder and run:

```
behave features/ --tags ~@skip
```

Please, be aware that this tests requires preparing a environment, including at least three glance servers and two keystone servers. Have a look to the `test/acceptance/README.rst` in order to get more information about how to prepare the environment to run the functional_test target.

Ent-to-end tests with Docker execution

Glancesync acceptance tests can be executed by Docker. To do that, firstly it is required to create the required docker images (fiware-glancesync and fiware-glancesync-acceptance). To do that:

```
docker build -t fiware-glancesync -f docker/Dockerfile docker
docker build -t fiware-glancesync-acceptance -f docker/AcceptanceTests/Dockerfile docker/AcceptanceTe
```

Once the images have been created, we can run the acceptance tests it by using docker-compose (to include the environment variables). To export then is required:

```
export OS_AUTH_URL = {the auth uri of the testbed agains the tests are going to be execute}
export OS_USERNAME = {the user name}
export OS_TENANT_NAME = {the tenant name}
export OS_PASSWORD = {the password}
export OS_REGION_NAME = {the region}
export OS_PROJECT_DOMAIN_NAME = {the project domain name}
export OS_USER_DOMAIN_NAME = {the user domain name}
export KEYSTONE_IP = {The keystone ip where testbed is deployed}
export Region1 = {The region name 1 for tests}
export Region2 = {The region name 2 for tests}
export Region3 = {The region name 3 for tests}
docker-compose -f docker/docker-compose.yml up
```

When docker has finished, you can obtain the tests results by .. code:

```
docker cp docker_fiwareglancesync-acceptance:/opt/fiware/glancesync/tests/acceptance/testreport .
```

Unit tests

To run the unit tests, you need to create a virtualenv using the requirements both contained in requirements.txt and test-requrirements.txt. You only need to execute the nosetests program in the root dorectory of the fiware-glancesync code. Keep in mind that it requires python2.7 or superior to execute the unit tests.

```
virtualenv -p <root to python v2.7> venv
source ./venv/bin/activate
pip install -r requirements.txt
pip install -r test-requirements.txt
nosetests --exe
deactivate
```

Eight tests are marked as skipped because they are more properly integration test. They are in the file ´´test_glancesync_serversfacade.py´´. The tested module contains all the code that interacts with Glance and the tests do some checks against a real glance server. To activate this eight tests, edit the file and change testingFacadeReal to True. It needs the usual OpenStack environment variables (*OS_USERNAME*, *OS_PASSWORD*, *OS_TENANT_NAME*, *OS_REGION_NAME*, *OS_AUTH_URL*)

Unit tests with Docker execution

Glancesync unit tests can be executed by docker. To do that, firstly it is required to create the docker image, with the following command:

```
docker build -t fiware-glancesync-build -f docker/UnitTests/Dockerfile docker
```

Once the fiware-glancesync-build image is created, we can run it by:

```
docker run --name fiware-glancesync-build fiware-glancesync-build
```

Finally, it is possible to obtain tests results and coverage information by:

```
docker cp fiware-glancesync-build:/opt/fiware/glancesync/test_results .
docker cp fiware-glancesync-build:/opt/fiware/glancesync/coverage .
```

Contributing new tests

It is possible to contribute new tests defining a scenario in *tests/resources* For a scenario 'new_scenario', the following folders must be created:

- **new_scenario: there are files for each region with the backup of the metadata** BEFORE invoking the synchronisation. These files can be generated with *sync.py –make-backup*
- **new_scenario.result: there are files for each region with the backup of the** metadata AFTER invoking the synchronisation
- **new_scenario.status_pre: there are files with the status of each region BEFORE invoking** the synchronisation. These files can be generated with the output of *sync.py –show-status*
- **new_scenario.status_post: there are files with the status of each region AFTER invoking** the synchronisation. These files can be generated with the output of *sync.py –show-status*

Inside the forlder *new_scenario*, optionally a *config* file may be included. If this file is not found, then the default configuration defined in the variable *config1* of the test file ´´tests/unit/test_glancesync.py´´ is used.

Then, a test class must be defined extending *TestGlanceSync_Sync*, for example:

```
class TestGlanceSync_AMI(TestGlanceSync_Sync):
    """Test a environment with AMI images (kernel_id/ramdisk_id)"""
    def config(self):
        path = os.path.abspath(os.curdir)
        self.path_test = path + '/tests/unit/resources/ami'
        self.regions = ['master:Burgos']
```

This class is provided in ´´tests/unit/test_glancesync.py´´.

More information about the mock: *mock*

Top

1.1.10 Support

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

Top

1.1.11 License

Top