Table of Contents

Introduction

This page aims to collect the documentation on how to use CloudInit. In particular it focuses on three modules developed in order to contextualize specific services often useful to run LHC experiments workflows into the cloud world. In here you will find how to use CloudInit to setup cvmfs, ganglia and condor, and also how to build or modify a module. We'll also introduce a simple module to configure the shoal-client service.

The tests and modifications that were conducted in order to achieve the following results were done on SLC and μCernVM images.

Before starting

NOTE: Legend of colors for the examples:

```
GRAY background for the commands to execute  (cut&paste)

GREEN background for the output sample of the executed commands

BLUE background for the configuration files (cut&paste)
```

Getting started

Basically CloudInit is a package that is used for early initialization of a cloud instance. This package is already installed in several cloud images like SLC5, SLC6 and soon in µCernVM.

It is possible to configure CloudInit through user-data during the instantiation, like with AMICONFIG, and there are several (eight to be precise) ways of building this user-data. Among all the user-data types that CloudInit can process, this document will focus on cloud-config. You can find useful information about CloudInit in the following websites:

- Official Documentation for CloudInit: https://help.ubuntu.com/community/CloudInit
- CERN Cloud Infrastructure User Guide: http://information-technology.web.cern.ch/node/4981
- Cloud-Init 0.7.2 documentation: https://cloudinit.readthedocs.org/en/latest/

Using Cloud Config

In this section you will see some cloud config examples and you'll also learn how to configure something through cloud config that is not known by the CloudInit package.

How does cloud config works

Cloud Config is a very user-friendly way of configuring services and issuing command through a user-data file. For instance, let's create a new user-data file with the following content:

```
#cloud-config
# Your cloud config file must always start with this first line, indicating that you want CloudIn

# We can run some bash script commands for example
runcmd:
 - [ echo, "Hello World. I am being run by CloudInit!" ]
```

Save this content into some file (let's assume the file is called 'myfile'). Now if you just boot an instance, passing this file, like this for example:

```
nova boot --image IMAGE-ID --flavor m1.small --key_name YOUR-KEY --meta cern-services=false --use
```

When the instance turns ACTIVE you can *ssh* into it and if you open the /var/log/boot.log file, you should get (among many other things):

```
Starting cloud-init-cfg: Hello World. I am being run by CloudInit!
```

During the boot, CloudInit will fetch the user-data from the metadata server and see what is inside. The first line of this file will indicate the input format.
"Cloud Config is the simplest way to accomplish some things via user-data". This syntax is built through modules (written in python). The default ones can be found in **/usr/lib/python-'version'/site-packages/cloudinit/CloudConfig/**. CloudInit knows what modules are there to be processed through the file **/etc/cloud/cloud.cfg** which has the list of all the python modules.

Building your own Cloud Config module

As said before, all Cloud Config modules are written in python.

For some enlightening you should first read some of the existing Cloud Config modules.
After writing yours, you just need to place the python file in **/usr/lib/python-'version'/site-packages/cloudinit/CloudConfig/** (note that this path can change according to CloudInit's version...this path is the default one in CloudInit 0.6.3). After doing this, you need to edit the **/etc/cloud/cloud.cfg** file and add the name of your module to the list. For example, let's assume that we wanted to add a new module called *mymodule*. We can either write the file *cc_mymodule.py* directly in the modules directory (inside the instance) or write it somewhere else (your personal computer for example) and copy it to the instance's modules directory. After doing this, we need to edit the "cloud.cfg" file and add our module's name. The resulting file should have the 'cloud_config_modules' section modified to look like this:

```
cloud_config_modules:
 - mounts
 - ssh-import-id
 - locale
 - set-passwords
 - timezone
 - puppet
 - disable-ec2-metadata
```

```
 - runcmd
 - mymodule
```

Snapshotting your instance

If you are planning to do some heavy testing on your own modules, probably the best thing to do it on-the-fly: boot your instance and make sure you provide your testing user-data; then log in and install your modules as explained above; the next thing to do is to erase CloudInit's previous contextualization data:

```
rm -rf /var/lib/cloud/instance/
```

and then reboot

```
reboot
```

After this, just log in again and your modules should have run.

Another way is to snapshot your instance (with the modifications from the last section of course).
You should be able to simply snapshot this instance and get a new image...and indeed you are but the problem is that when you try to boot this snapshot you'll get some network problems (the instance does not get network).

To solve this you must (before snapshotting) execute the following command in your modified instance:

```
[ -e /etc/udev/rules.d/70-persistent-net.rules ] && /bin/rm /etc/udev/rules.d/70-persistent-net.r
```

This will delete udev rules for vNIC and solve your networking issue. You can now snapshot your instance!

Read more about this in
http://information-technology.web.cern.ch/book/cern-cloud-infrastructure-user-guide/administering-vms/snapshotting-
.

CVMFS, Ganglia and Condor in CloudInit

CernVM File System, Ganglia and Condor are services that are used a lot by CERN users. Therefore having these services installed and configured in the experiments' workernodes is a strong asset.

Setup by scripting

The most obvious way to setup these services is to 'bash script our way into it'. There are instructions on how to do this in:

- The CernVM File System (technical report):
http://cvmrepo.web.cern.ch/cvmrepo/extras/cvmfstech-2.0-7.pdf
- Ganglia installation and configuration (wiki page):
http://sourceforge.net/apps/trac/ganglia/wiki/ganglia_configuration
- HTCondor YUM Repository (web page): http://research.cs.wisc.edu/htcondor/yum/

You could either log in into your instance and perform those instructions, or you could pass them as user-data during the boot (user-data scripts is one of the formats that CloudInit accepts).
Although it isn't hard to configure these services through scripting, it is a lot of coding to do every time you boot an instance or even to pass through user-data because it generates too much confusion in the user-data file.

Setup by Cloud Config

The easier and cleaner way of configuring CVMFS, Ganglia and Condor is to do it through Cloud Config. We want to be able to configure these services as easy as how it is done with AMICONFIG.

For this we wrote our own modules and baked them into an image.

Building the modules

All of the Cloud Config modules run during the boot and each one of them will search the user-data so they know if there is any configuration to be done or not. So in our case, we obviously have to guarantee that our modules will only setup CVMFS, Ganglia and/or Condor if there is any reference to these services on the user-data file.

If there is, the respective module will install the service (if asked to) and perform the necessary configurations so that the user has the minimal amount of effort when writing the user-data file.

You can find and download these python modules in the cloud-init-cern GitHub repository. Feel free to use, test and modify them and always remember what was told in the previous sections of this page in order to make them work properly.
Here's a little piece of bash code which you can just simply copy&paste on or instance in order to download, install and configure these CloudInit modules:

```
#!/bin/sh

wget https://raw.github.com/cjdcordeiro/cloud-init-cern/devel/cloudinit/config/cc_condor.py https
dest_dir='/usr/lib/python2.6/site-packages/cloudinit/CloudConfig/'     # or /usr/lib/python2.6/si

mv cc_condor.py $dest_dir
mv cc_cvmfs.py $dest_dir
mv cc_ganglia.py $dest_dir
```

```
current='cloud_config_modules:'
new='cloud_config_modules:\n - cvmfs\n - ganglia\n - condor'
sed -i.bak "s/${current}/${new}/g" /etc/cloud/cloud.cfg
```

You can also avoid all of these coding lines and go for the automatic installation from the RPM. Just download the repo file from https://cern-cloudinit-modules.web.cern.ch/cern-cloudinit-modules/, place it under */etc/yum.repos.d* and do:

```
sudo yum search cloudinit
# And then install whatever you want/need
```

Configuring CVMFS, Ganglia and Condor

After doing the above steps you are all set to configure these services through Cloud Config. Please note that these modules are regularly updated so that they can be faster, more configurable and easier to user. Therefore, frequent visits to this documentation are highly recommended for those who are working on or with these modules.

Always remember to respect the white-spaces and to *quote the double quotes* when you want the quotes to be a part of the value.

Note

By default, each module will only install their corresponding service from scratch if they are intentionally asked to do so. This can be achieved by using a special common parameter called ***install***. If you want to install the service, set it to *True*. Otherwise just don't mention it or assign it as *False*. For example:

```
service:
 install: True
 param_block:
   param1: value
```

Notice that this installation parameter can cause configuration issues like conflicts between multi installations of the same service, or even attempting to configure nonexistent services. It is user's responsibility to user this parameter correctly.

Configuring CVMFS

For the CVMFS service, you can pretty much configure whatever you want. This kind of freedom carries a lot of responsibility, in a way that everything, but everything that you define in the user-data will be written in the CVMFS configuration file. Therefore, it is entirely user's responsibility to ensure a correct parameters' configuration.

There are though some parameters which are very common and which are defined by default but that can be overwritten if the user decides so. They are:

- CVMFS_QUOTA_LIMIT = 8000
- CVMFS_CACHE_BASE = /cvmfs_cache
- CVMFS_MOUNT_RW = yes

Some other special parameters are also allowed, as the *CMS_LOCAL_SITE_* value which will end up being written in */etc/cvmfs/config.d/cms.cern.ch.local_* and the *CVMFS_SERVER_URL* values under the **domain** section, which will be written then in */etc/cvmfs/domain.d/cern.ch.local*.

Resuming, a simple and generic CVMFS configuration could be:

```
# Respect the white-spaces
cvmfs:
 CMS_LOCAL_SITE: value
 local:
  CVMFS_REPOSITORIES: value
  CVMFS_HTTP_PROXY: value
 domain:
  CVMFS_SERVER_URL: '"value"'
```

Configuring Ganglia

In Ganglia's case there are certain number of parameters that can be configured. For this service you are also allowed to install and configure a Ganglia node. Therefore, for a Ganglia node you can configure the following parameters:

globals section	**cluster** section	**udp_send_channel** section	**udp_recv_channel** section	**tcp_accept_channel** section
• daemonize • setuid • user • debug_level • max_udp_msg_len • mute • deaf • allow_extra_data • host_dmax • cleanup_threshold • gexec • send_metadata_interval • override_hostname • override_ip	• name • owner • latlong • url	• host • port • ttl	• port • bind	• port

These parameters will be used to modify the *gmond.conf* file, which is created with default values during the gmond installation. Your full Ganglia node configuration on the user-data file should look like the following:

```
ganglia:
 globals:
  daemonize: value
  setuid: value
  user: value
  debug_level: value
  max_udp_msg_len: value
  mute: value
  deaf: value
  allow-extra-data: value
  host-dmax: value
  cleanup-threshold: value
  gexec: value
  send_metadata_interval: value
  override_hostname: value
  override_ip:  value
 cluster:
  name: '"value"'
  owner: '"value"'
  latlong: '"value"'
  url: '"value"'
 udp_recv_channel:
  port: value
  bind: value
 tcp_accept_channel:
```

```
  port: value
 udp_send_channel:
  host: value
  port: value
  ttl: value
```

NOTE 1: There is a service triggering parameter that can be used to instruct CloudInit to either enable or disable *gmond*:

enabled: on/off # or true/false

NOTE 2: In some cases one might want to override_hostname according to some dynamically created string, only know after the contextualization. For this there is a special parameter to be used instead of the *override_hostname*, which will accept a bash instruction to be executed. The output of this instruction will be used to override hostname in Ganglia.
override_hostname_bash: echo "any bash instruction with an output"

Configuring Condor

For the Condor service, there is the same parameters' freedom as for CVMFS - everything you write is considered as being a valid configuration. Again, a good service setup is entirely user's responsibility.

Also as in CVMFS, there is a list of pre-configured parameters which can be overwritten. They are:

- DAEMON_LIST = MASTER, STARTD
- HIGHPORT = 24500
- LOWPORT = 20000
- START = TRUE
- SUSPEND = FALSE
- PREEMPT = FALSE
- KILL = FALSE
- QUEUE_SUPER_USERS = root, condor
- ALLOW_WRITE = condor@*.*
- STARTER_ALLOW_RUNAS_OWNER = False
- ALLOW_DAEMON = *
- RELEASE_DIR = /usr
- LOCAL_DIR = /var
- RANK = 0

There is though ONE special parameter which is not meant to be written in Condor's configuration. It is:

- pool-password

and is expects a string (normally base64 encoded) that will be written in the directory defined by the parameter SEC_PASSWORD_FILE. If this parameter is not specified, then this content will be written in */root/pool_password*.

There is also a special keywork, which is *IPADDRESS*. This keyword can either be a parameter value, or part of it, and it represents the machine's IP address. Obviously, the user doesn't know this IP address before hand, thence the special keyword, that will be replaced by the real value during contextualization.

Finally, another module feature is the parameter *SLOT*_USER* which normally is dynamically added (as well as the corresponding users in the machine), with no need to mention it in the user-data file. But if for some reason we want to fix this slot, independently of the core count, we just need to write it in the user-data file.

The parameter CONDOR_IDS is dynamically configured.

Configuring Shoal Client

"Shoal-client is a simple python script typically configured to run with cron to check for new squids periodically."

With this module one can configure (through contextualization) the Shoal-client. Full freedom is provided, meaning that whatever parameters are in this section, they will be written in "/etc/shoal/shoal_client.conf". It is then assumed that the user is completely aware of how *shoal-client* works and that the *shoal-client* is already installed in the VM. There is though one special keyword (**cron_shoal**) that won't be written in the configuration file, which allows the user to set a cronjob for *shoal-client*.

A custom default configuration is also provided, and can be overwritten in case its parameters are referenced in the userdata file. Otherwise, this is the default template:

- [general]
- cvmfs_config = /etc/cvmfs/default.local
- shoal_server_url = http://localhost:8080/nearest
- default_squid_proxy = http://chrysaor.westgrid.ca:3128;http://cernvm-webfs.atlas-canada.ca:3128;DIRECT

Here's an userdata example showing how can we configure *shoal-client*:

```
# Respect the white-spaces
shoal:
 shoal_server_url: value
 cron_shoal: '50 * * * * /usr/bin/shoal-client'     # just an example
```

After the contextualization, the VM will get a cronjob with the above specified characteristics and the file "/etc/shoal/shoal_client.conf" will be:

```
[general]

cvmfs_config = /etc/cvmfs/default.local
shoal_server_url = value
default_squid_proxy = http://chrysaor.westgrid.ca:3128;http://cernvm-webfs.atlas-canada.ca:3128;D
```

Working Examples

The following user-data example represents a basic configuration for CVMFS, Ganglia, Shoal-client and Condor using Cloud Config:

```
#cloud-config

# Let's start with cvmfs
cvmfs:
 install: True
 local:
  CVMFS_REPOSITORIES: cms.cern.ch,grid.cern.ch
  CVMFS_HTTP_PROXY: DIRECT

# Now let's install and configure a ganglia node
ganglia:
 install: True
 enabled: false
 udp_send_channel:
  host: cloudm.cern.ch

# Finally, Condor...let's connect it to a fictitious master called myinstancemaster where you can
# You can create this master also by using the CloudInit module. You would just need to have your
condor:
 LOWPORT: 20000
 HIGHPORT: 24500
 CONDOR-HOST: myinstancemaster.cern.ch
 DAEMON-LIST: MASTER, STARTD

shoal:
 shoal_server_url: http://shoal.heprc.uvic.ca/nearest
 cron_shoal: '0,30 * * * * /usr/bin/shoal-client'
```

To confirm that these services were correctly configured you can *ssh* into the instance and run:

```
cvmfs_config status
cvmfs2 -V
```

and

```
service gmond status
ganglia-config --version
```

and

```
service condor status
condor_version
```

You should get something like this (with different pid's and versions obviously):

```
automount (pid  4828) is running...
CernVM-FS mounted on /cvmfs/cms.cern.ch with pid 5022
CernVM-FS mounted on /cvmfs/grid.cern.ch with pid 5110
CernVM-FS version 2.0.19
```

and

```
gmond (pid 5107) is running...
Ganglia 3.1.7
```

and finally

```
Condor is running (pid 1704)
$CondorVersion: 7.8.7 Dec 12 2012 BuildID: 86173 $
$CondorPlatform: x86_64_rhap_6.3 $
```

Testing and Developing

For those who want to directly test and perform modifications on these modules, there is the already referenced GitHub repository where you can download and/or contribute to this work. Since the AI is now on Grizzly, we are still working to have these modules prepared to be incorporated in public images such as SLC5, SLC6 and µCernVM. Until then feel free to use whats was explained above in this documentation to create your own snapshots and play around with these modules. Enjoy!

This topic: LCG > CloudInit
Topic revision: r50 - 2015-10-20 - CristovaoCordeiro