XIFI BLOG What's happening in XIFI ?

Extending Fuel 3.2 to support nagios-based monitoring

30/12/2013 FI-OPS, FI-PPP, XIFI Federico Facca

g+1 0 Like 1 Tweet 9

Fuel by Mirantis is a powerful tool to support deployment of openstack based clouds. It provides a simple web interface through which it is possible to configure and install openstack services on bare-metal.

As matter of fact, Fuel is based on puppet, and its web interface is providing ways to "configure" puppet modules used to deploy openstack. Essentially the fuel distribution includes a puppet master that orchestrate the deployment of the different nodes part of the cloud set-up.

Fuel 3.2 puppet module library actually includes a module called "nagios" that supposedly is intended by Mirantis to allow installation of Nagios through Fuel. Nevertheless the module is not completed and not included in the puppet master files (e.g. *osnailyfacter/manifests/cluster_simple.pp*) that drive the openstack deployment in fuel.

First of all it is important to mention that Nagios, differently from other infrastructure monitoring services, does not support self-discovery. This requires some attention to the definition of puppet modules related to it. Usually puppet modules related to nagios uses the puppet export resource feature. This feature is quite powerful but requires to have a puppetdb on your puppet master node. Fuel 3.2 does not include this by default.

Install PuppetDB

To install it manually, following puppet documentation:

```
puppet resource package puppetdb ensure=latest
puppet resource service puppetdb ensure=running enable=true
```

you may need to increase memory available to puppetdb, if you deploy a large number of nodes, in case follow puppedb documentation.

Now, you need to edit *puppet.conf*, to enable PuppetDB for the inventory service and saved catalogs/exported resources, add the following settings to the *[master]* block of puppet.conf (or edit them if already present):

```
[master]
storeconfigs = true
storeconfigs_backend = puppetdb
```

Edit *routes.yaml*, the routes.yaml file will probably not exist yet. Create it if necessary, and add the following:

```
---
master:
facts:
terminus: puppetdb
cache: yaml
```

This will make PuppetDB the authoritative source for the inventory service.

On Fuel the port 8080 is already used, so we need to edit */etc/puppetdb/conf.d/jetty.ini* to use a different port (e.g. 8088)

```
port = 8088
```

We need to open the port on the firewall

```
iptables -A INPUT -m state --state NEW -p tcp --dport 8088 -j
ACCEPT
```

We can at this point start puppetdb

```
service puppetdb start
```

Now we can restart puppet master:

```
service puppetmaster restart
```

And test that everything is running correctly:

```
curl -H "Accept: application/json" 'http://localhost:8088/nodes'
```

Export nagios resources

Now we can export resources across nodes. The idea, is that when a new node is installed, the services, hostgroups and hosts related to that node are exported to the node that hosts the nagios server.
Given the way it works Fuel, there are two issues we need to manage:

- ensure that only the services, hostgroups and hosts related to the current environment are registered on the nagios server. by the default, if we use the same fuel set-up to install different environments all the services, hostgroups and hosts will recorded on the nagios server, regardless the environment they belong too
- ensure that nagios server re-collects exported resources periodically. given that fuel runs the agent only once on the different hosts, if we install nagios on the controller, that is the first host to be installed, only resources exported by the controller host will be retrieved.

we solve the first issue by tagging resources with the deployment_id generated by fuel, e.g.:

```
define nagios::host::hosts() {

  $deployment_id = $::fuel_settings['deployment_id']
  notify{ "**** called hosts() on host ${::hostname} tag
deployment_${deployment_id} *****": }

  @@nagios_host { "${name}_${deployment_id}":
    ensure => present,
    alias => $::hostname,
    use => 'default-host',
    address => $::fqdn,
    host_name => $::fqdn,
    target =>
"/etc/${nagios::params::masterdir}/${nagios::master_proj_name}/ho
sts.cfg",
    tag => "deployment_${deployment_id}"
  }
}
```

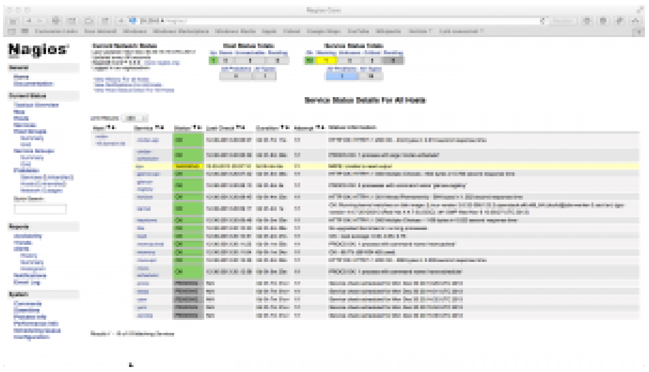
we solve the second issue by creating a cron entry on the nagios server hosts that periodically activate the puppet agent only for nagios resources.

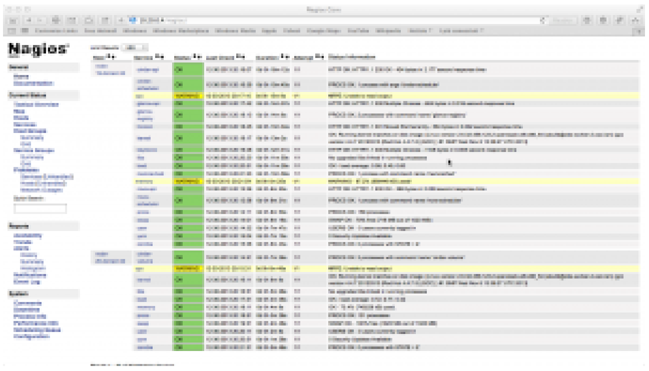
```
cron { puppet-agent:
  command => "puppet agent --onetime --tags=nagios",
  user => root,
  minute => '*/10'
}
```

this cron can be later disabled by editing the crontab in the server machine. alternatively we can manually run the command

```
  puppet agent --onetime --tags=nagios
```

on the nagios server when the deployment is completed (this allows for turning off the fuel host when done)


Nagios dashboard after the installation of the controller node


Nagios dashboard after the installation of the cinder node

you can find the source code of the modified version of the nagios puppet module here.
as regard, Fuel manifest, here you can find an example of cluster_simple.pp including the changes required to call nagios classes.

We are shortly going to extend the Fuel GUI to allow for the set-up of "monitor" role. This will allow for decoupling between the OpenStack controller and the nagios sever.

N.B.: the current version has been only tested on centos based deployments.

+1 0 Like 1 Tweet 9