Puppet-Gluster

A GlusterFS Puppet module by James

Available from:

https://github.com/purpleidea/puppet-gluster/

Also available from:

https://forge.gluster.org/puppet-gluster/

This documentation is available in: Markdown or PDF format.

Table of Contents

Overview

The Puppet-Gluster module installs, configures, and manages a GlusterFS cluster.

Module Description

This Puppet-Gluster module handles installation, configuration, and management of GlusterFS across all of the hosts in the cluster.

Setup

What can Puppet-Gluster manage?

Puppet-Gluster is designed to be able to manage as much or as little of your GlusterFS cluster as you wish. All features are optional. If there is a feature that doesn't appear to be optional, and you believe it should be, please let me know. Having said that, it makes good sense to me to have Puppet-Gluster manage as much of your GlusterFS infrastructure as it can. At the moment, it cannot rack new servers, but I am accepting funding to explore this feature ;) At the moment it can manage:

- GlusterFS packages (rpm)
- GlusterFS configuration files (/var/lib/glusterd/)
- GlusterFS host peering (gluster peer probe)
- GlusterFS storage partitioning (fdisk)
- GlusterFS storage formatting (mkfs)
- GlusterFS brick creation (mkdir)
- GlusterFS services (glusterd)
- GlusterFS firewalling (whitelisting)
- GlusterFS volume creation (gluster volume create)
- GlusterFS volume state (started/stopped)
- GlusterFS volume properties (gluster volume set)
- And much more...

Simple setup

include '::gluster::simple' is enough to get you up and running. When using the gluster::simple class, or with any other Puppet-Gluster configuration, identical definitions must be used on all hosts in the cluster. The simplest way to accomplish this is with a single shared puppet host definition like:

```
node /^annex\d+$/ {        # annex{1,2,..N}
        class { '::gluster::simple':
        }
}
```

If you wish to pass in different parameters, you can specify them in the class before you provision your hosts:

```
class { '::gluster::simple':
    replica => 2,
    volume => ['volume1', 'volume2', 'volumeN'],
}
```

Elastic setup

The gluster::elastic class is not yet available. Stay tuned!

Advanced setup

Some system administrators may wish to manually itemize each of the required components for the Puppet-Gluster deployment. This happens automatically with the higher level modules, but may still be a desirable feature, particularly for non-elastic storage pools where the configuration isn't expected to change very often (if ever).

To put together your cluster piece by piece, you must manually include and define each class and type that you wish to use. If there are certain aspects that you wish to manage yourself, you can omit them from your configuration. See the reference section below for the specifics. Here is one possible example:

```
class { '::gluster::server':
    shorewall => true,
}

gluster::host { 'annex1.example.com':
    # use uuidgen to make these
    uuid => '1f660ca2-2c78-4aa0-8f4d-21608218c69c',
}

# note that this is using a folder on your existing file system...
# this can be useful for prototyping gluster using virtual machines
# if this isn't a separate partition, remember that your root fs will
# run out of space when your gluster volume does!
gluster::brick { 'annex1.example.com:/data/gluster-storage1':
```

```
    areyousure => true,
}

gluster::host { 'annex2.example.com':
    # NOTE: specifying a host uuid is now optional!
    # if you don't choose one, one will be assigned
    #uuid => '2fbe6e2f-f6bc-4c2d-a301-62fa90c459f8',
}

gluster::brick { 'annex2.example.com:/data/gluster-storage2':
    areyousure => true,
}

$brick_list = [
    'annex1.example.com:/data/gluster-storage1',
    'annex2.example.com:/data/gluster-storage2',
]

gluster::volume { 'examplevol':
    replica => 2,
    bricks => $brick_list,
    start => undef, # i'll start this myself
}

# namevar must be: <VOLNAME>#<KEY>
gluster::volume::property { 'examplevol#auth.reject':
    value => ['192.0.2.13', '198.51.100.42', '203.0.113.69'],
}
```

Client setup

Mounting a GlusterFS volume on a client is fairly straightforward. Simply use the 'gluster::mount' type.

```
    gluster::mount { '/mnt/gluster/puppet/':
        server => 'annex.example.com:/puppet',
        rw => true,
        shorewall => false,
    }
```

In this example, 'annex.example.com' points to the VIP of the GlusterFS cluster. Using the VIP for mounting increases the chance that you'll get an available server when you try to mount. This generally works better than RRDNS or similar schemes.

Usage and frequently asked questions

All management should be done by manipulating the arguments on the appropriate Puppet-Gluster classes and types. Since certain manipulations are either not yet possible with Puppet-Gluster, or are not supported by GlusterFS, attempting to manipulate the Puppet configuration in an unsupported way will result in undefined behaviour, and possible even data loss, however this is unlikely.

How do I change the replica count?

You must set this before volume creation. This is a limitation of GlusterFS. There are certain situations where you can change the replica count by adding a multiple of the existing brick count to get this desired effect. These cases are not yet supported by Puppet-Gluster. If you want to use Puppet-Gluster before and / or after this transition, you can do so, but you'll have to do the changes manually.

Do I need to use a virtual IP?

Using a virtual IP (VIP) is strongly recommended as a distributed lock manager (DLM) and also to provide a highly-available (HA) IP address for your clients to connect to. For a more detailed explanation of the reasoning please see:

How to avoid cluster race conditions or: How to implement a distributed lock manager in puppet

Remember that even if you're using a hosted solution (such as AWS) that doesn't provide an additional IP address, or you want to avoid using an additional IP, and you're okay not having full HA client mounting, you can use an unused private RFC1918 IP address as the DLM VIP. Remember that a layer 3 IP can co-exist on the same layer 2 network with the layer 3 network that is used by your cluster.

Is it possible to have Puppet-Gluster complete in a single run?

No. This is a limitation of Puppet, and is related to how GlusterFS operates. For example, it is not reliably possible to predict which ports a particular GlusterFS volume will run on until after the volume is started. As a result, this module will initially whitelist connections from GlusterFS host IP addresses, and then further restrict this to only allow individual ports once this information is known. This is possible in conjunction with the puppet-shorewall module. You should notice that each run should complete without error. If you do see an error, it means that either something is wrong with your system and / or configuration, or because there is a bug in Puppet-Gluster.

Can you integrate this with vagrant?

Yes, see the vagrant/ directory. This has been tested on Fedora 20, with vagrant-libvirt, as I have no desire to use VirtualBox for fun. I have written an article about this:

Automatically deploying GlusterFS with Puppet-Gluster + Vagrant!

You'll probably first need to read my three earlier articles to learn some vagrant tricks, and to get the needed dependencies installed:

- Vagrant on Fedora with libvirt
- Vagrant vsftp and other tricks
- Vagrant clustered SSH and 'screen'

Puppet runs fail with "Invalid relationship" errors.

When running Puppet, you encounter a compilation failure like:

```
Error: Could not retrieve catalog from remote server:
Error 400 on SERVER: Invalid relationship: Exec[gluster-volume-stuck-volname] {
require => Gluster::Brick[annex2.example.com:/var/lib/puppet/tmp/gluster/data/]
}, because Gluster::Brick[annex2.example.com:/var/lib/puppet/tmp/gluster/data/]
doesn't seem to be in the catalog
Warning: Not using cache on failed catalog
Error: Could not retrieve catalog; skipping run
```

This can occur if you have changed (usually removed) the available bricks, but have not cleared the exported resources on the Puppet master, or if there are stale (incorrect) brick "tags" on the individual host. These tags can usually be found in the */var/lib/puppet/tmp/gluster/brick/* directory. In other words, when a multi host cluster comes up, each puppet agent tells the master about which bricks it has available, and each agent also pulls down this list and stores it in the brick directory. If there is a discrepancy, then the compile will fail because the individual host is using old data as part of its facts when it uses the stale brick data as part of its compilation.

This commonly happens if you're trying to deploy a different Puppet-Gluster setup without having first erased the host specific exported resources on the Puppet master or if the machine hasn't been re-provisioned from scratch.

To solve this problem, do a clean install, and make sure that you've cleaned the Puppet master with:

```
puppet node deactivate HOSTNAME
```

for each host you're using, and that you've removed all of the files from the brick directories on each host.

Puppet runs fail with "Connection refused - connect(2)" errors.

You may see a "*Connection refused - connect(2)*" message when running puppet. This typically happens if your puppet vm guest is overloaded. When running high guest counts on your laptop, or running without hardware virtualization support this is quite common. Another common causes of this is if your domain type is set to *qemu* instead of the accelerated *kvm*. Since the *qemu* domain type is much slower, puppet timeouts and failures are common when it doesn't respond.

Provisioning fails with: "Can't open /dev/sdb1 exclusively."

If when provisioning you get an error like:

"Can't open /dev/sdb1 exclusively. Mounted filesystem?"

It is possible that dracut might have found an existing logical volume on the device, and device mapper has made it available. This is common if you are re-using dirty block devices that haven't run through a *dd* first. Here is an example of the diagnosis and treatment of this problem:

```
[root@server mapper]# pwd
/dev/mapper
[root@server mapper]# dmesg | grep dracut
dracut: dracut-004-336.el6_5.2
dracut: rd_NO_LUKS: removing cryptoluks activation
dracut: Starting plymouth daemon
dracut: rd_NO_DM: removing DM RAID activation
dracut: rd_NO_MD: removing MD RAID activation
dracut: Scanning devices sda3 sdb  for LVM logical volumes myvg/rootvol
dracut: inactive '/dev/vg_foo/lv' [4.35 TiB] inherit
dracut: inactive '/dev/myvg/rootvol' [464.00 GiB] inherit
dracut: Mounted root filesystem /dev/mapper/myvg-rootvol
dracut: Loading SELinux policy
dracut:
dracut: Switching root
[root@server mapper]# /sbin/pvcreate --dataalignment 2560K /dev/sdb1
  Can't open /dev/sdb1 exclusively.  Mounted filesystem?
[root@server mapper]# ls
control  myvg-rootvol  vg_foo-lv
[root@server mapper]# ls -lAh
total 0
crw-rw----. 1 root root 10, 58 Mar  7 16:42 control
lrwxrwxrwx. 1 root root      7 Mar 13 09:56 myvg-rootvol -> ../dm-0
lrwxrwxrwx. 1 root root      7 Mar 13 09:56 vg_foo-lv -> ../dm-1
[root@server mapper]# dmsetup remove vg_foo-lv
```

```
[root@server mapper]# ls
control  myvg-rootvol
[root@server mapper]# pvcreate --dataalignment 2560K /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
[root@server mapper]# HAPPY_ADMIN='yes'
```

If you frequently start with "dirty" block devices, you may consider adding a *dd* to your hardware provisioning step. The downside is that this can be very time consuming, and potentially dangerous if you accidentally re-provision the wrong machine.

Provisioning fails with: "cannot open /dev/sdb1: Device or resource busy"

If when provisioning you get an error like:

"mkfs.xfs: cannot open /dev/sdb1: Device or resource busy"

It is possible that dracut might have found an existing logical volume on the device, and device mapper has made it available. This is common if you are re-using dirty block devices that haven't run through a *dd* first. This is almost identical to the previous frequently asked question, although this failure message is what is seen when *mkfs.xfs* is being blocked by dracut, where in the former problem it was the *pvcreate* that was being blocked. The reason that we see this manifest through *mkfs.xfs* instead of *pvcreate* is that this particular cluster is being build with *lvm => false*. Here is an example of the diagnosis and treatment of this problem:

```
[root@server mapper]# pwd
/dev/mapper
[root@server mapper]# dmesg | grep dracut
dracut: dracut-004-335.el6
dracut: rd_NO_LUKS: removing cryptoluks activation
dracut: Starting plymouth daemon
dracut: rd_NO_DM: removing DM RAID activation
dracut: rd_NO_MD: removing MD RAID activation
dracut: Scanning devices sda2 sdb  for LVM logical volumes vg_server/lv_swap vg_server/lv_ro
dracut: inactive '/dev/vg_bricks/b1' [9.00 TiB] inherit
dracut: inactive '/dev/vg_server/lv_root' [50.00 GiB] inherit
dracut: inactive '/dev/vg_server/lv_home' [383.26 GiB] inherit
dracut: inactive '/dev/vg_server/lv_swap' [31.50 GiB] inherit
dracut: Mounted root filesystem /dev/mapper/vg_server-lv_root
dracut:
dracut: Switching root
[root@server mapper]# mkfs.xfs -q -f -i size=512 -n size=8192 /dev/sdb1
mkfs.xfs: cannot open /dev/sdb1: Device or resource busy
```

```
[root@server mapper]# lsof /dev/sdb1
[root@server mapper]# echo $?
1
[root@server mapper]# ls
control       vg_server-lv_home  vg_server-lv_swap
vg_bricks-b1  vg_server-lv_root
[root@server mapper]# ls -lAh
total 0
crw-rw---- 1 root root 10, 58 May 20  2014 control
lrwxrwxrwx 1 root root      7 May 20  2014 vg_bricks-b1 -> ../dm-2
lrwxrwxrwx 1 root root      7 May 20  2014 vg_server-lv_home -> ../dm-3
lrwxrwxrwx 1 root root      7 May 20  2014 vg_server-lv_root -> ../dm-0
lrwxrwxrwx 1 root root      7 May 20  2014 vg_server-lv_swap -> ../dm-1
[root@server mapper]# dmsetup remove vg_bricks-b1
[root@server mapper]# ls
control  vg_server-lv_home  vg_server-lv_root  vg_server-lv_swap
[root@server mapper]# mkfs.xfs -q -f -i size=512 -n size=8192 /dev/sdb1
[root@server mapper]# echo $?
0
[root@server mapper]# HAPPY_ADMIN='yes'
```

If you frequently start with "dirty" block devices, you may consider adding a *dd* to your hardware provisioning step. The downside is that this can be very time consuming, and potentially dangerous if you accidentally re-provision the wrong machine.

I changed the hardware manually, and now my system won't boot.

If you're using Puppet-Gluster to manage storage, the filesystem will be mounted with *UUID* entries in */etc/fstab*. This ensures that the correct filesystem will be mounted, even if the device order changes. If a filesystem is not available at boot time, startup will abort and offer you the chance to go into read-only maintenance mode. Either fix the hardware issue, or edit the */etc/fstab* file.

I can't edit /etc/fstab in the maintenance shell because it is read-only.

In the maintenance shell, your root filesystem will be mounted read-only, to prevent changes. If you need to edit a file such as */etc/fstab*, you'll first need to remount the root filesystem in read-write mode. You can do this with:

```
mount -n -o remount /
```

Will this work on my favourite OS? (eg: GNU/Linux F00bar OS v12 ?)

If it's a GNU/Linux based OS, can run GlusterFS, and Puppet, then it will probably work. Typically, you might need to add a yaml data file to the *data/* folder so that Puppet-Gluster knows where certain operating system specific things are found. The multi-distro support has been designed to make it particularly easy to add support for additional platforms. If your platform doesn't work, please submit a yaml data file with the platform specific values.

Awesome work, but it's missing support for a feature and/or platform!

Since this is an Open Source / Free Software project that I also give away for free (as in beer, free as in gratis, free as in libre), I'm unable to provide unlimited support. Please consider donating funds, hardware, virtual machines, and other resources. For specific needs, you could perhaps sponsor a feature!

You didn't answer my question, or I have a question!

Contact me through my technical blog and I'll do my best to help. If you have a good question, please remind me to add my answer to this documentation!

Reference

Please note that there are a number of undocumented options. For more information on these options, please view the source at: https://github.com/purpleidea/puppet-gluster/. If you feel that a well used option needs documenting here, please contact me.

Overview of classes and types

- gluster::simple: Simple Puppet-Gluster deployment.
- gluster::elastic: Under construction.
- gluster::server: Base class for server hosts.
- gluster::host: Host type for each participating host.
- gluster::brick: Brick type for each defined brick, per host.
- gluster::volume: Volume type for each defined volume.
- gluster::volume::property: Manages properties for each volume.
- gluster::mount: Client volume mount point management.

gluster::simple

This is gluster::simple. It should probably take care of 80% of all use cases. It is particularly useful for deploying quick test clusters. It uses a finite-state machine (FSM) to decide when the cluster has settled and volume creation can begin. For more information on the FSM in Puppet-Gluster see: https://ttboj.wordpress.com/2013/09/28/finite-state-machines-in-puppet/

`replica` The replica count. Can't be changed automatically after initial deployment.

`volume` The volume name or list of volume names to create.

`path` The valid brick path for each host. Defaults to local file system. If you need a different path per host, then Gluster::Simple will not meet your needs.

`count` Number of bricks to build per host. This value is used unless *brick*params_ is being used.

`vip` The virtual IP address to be used for the cluster distributed lock manager. This option can be used in conjunction with the *vrrp* option, but it does not require it. If you don't want to provide a virtual ip, but you do want to enforce that certain operations only run on one host, then you can set this option to be the ip address of an arbitrary host in your cluster. Keep in mind that if that host is down, certain options won't ever occur.

`vrrp` Whether to automatically deploy and manage *Keepalived* for use as a *DLM* and for use in volume mounting, etc... Using this option requires the *vip* option.

`layout` Which brick layout to use. The available options are: *chained*, and (default). To generate a default (symmetrical, balanced) layout, leave this option blank. If you'd like to include an algorithm that generates a different type of brick layout, it is easy to drop in an algorithm. Please contact me with the details!

`version` Which version of GlusterFS do you want to install? This is especially handy when testing new beta releases. You can read more about the technique at: Testing GlusterFS during Glusterfest.

repo Whether or not to add the necessary software repositories to install the needed packages. This will typically pull in GlusterFS from *download.gluster.org* and should be set to false if you have your own mirrors or repositories managed as part of your base image.

brick_params This parameter lets you specify a hash to use when creating the individual bricks. This is especially useful because it lets you have the power of Gluster::Simple when managing a cluster of iron (physical machines) where you'd like to specify brick specific parameters. This sets the brick count when the *count* parameter is 0. The format of this parameter might look like:

```
$brick_params = {
    fqdn1 => [
        {dev => '/dev/disk/by-uuid/01234567-89ab-cdef-0123-456789abcdef'},
        {dev => '/dev/sdc', partition => false},
    ],
    fqdn2 => [{
        dev => '/dev/disk/by-path/pci-0000:02:00.0-scsi-0:1:0:0',
        raid_su => 256, raid_sw => 10,
    }],
    fqdnN => [...],
}
```

brick_param_defaults This parameter lets you specify a hash of defaults to use when creating each brick with the *brick*params_ parameter. It is useful because it avoids the need to repeat the values that are common across all bricks in your cluster. Since most options work this way, this is an especially nice feature to have. The format of this parameter might look like:

```
$brick_param_defaults = {
    lvm => false,
    xfs_inode64 => true,
    force => true,
}
```

brick_params_defaults This parameter lets you specify a list of defaults to use when creating each brick. Each element in the list represents a different brick. The value of each element is a hash with the actual defaults that you'd like to use for creating that brick. If you do not specify a brick count by any other method, then the number of elements in this array will be used as the brick count. This is very useful if you have consistent device naming across your entire cluster, because you can very easily specify the devices and brick counts once for all hosts. If for some reason a particular device requires unique values, then it

can be set manually with the *brick*params_ parameter. Please note the spelling of this parameter. It is not the same as the *brick*param_defaults_ parameter which is a global defaults parameter which will apply to all bricks. The format of this parameter might look like:

```
$brick_params_defaults = [
    {'dev' => '/dev/sdb'},
    {'dev' => '/dev/sdc'},
    {'dev' => '/dev/sdd'},
    {'dev' => '/dev/sde'},
]
```

setgroup Set a volume property group. The two most common or well-known groups are the *virt* group, and the *small-file-perf* group. This functionality is emulated whether you're using the RHS version of GlusterFS or if you're using the upstream GlusterFS project, which doesn't (currently) have the *volume set group* command. As package managers update the list of available groups or their properties, Puppet-Gluster will automatically keep your set group up-to-date. It is easy to extend Puppet-Gluster to add a custom group without needing to patch the GlusterFS source.

ping Whether to use *fping* or not to help with ensuring the required hosts are available before doing certain types of operations. Optional, but recommended. Boolean value.

again Do you want to use *Exec['again']* ? This helps build your cluster quickly!

baseport Specify the base port option as used in the glusterd.vol file. This is useful if the default port range of GlusterFS conflicts with the ports used for virtual machine migration, or if you simply like to choose the ports that you're using. Integer value.

rpcauthallowinsecure This is needed in some setups in the glusterd.vol file, particularly (I think) for some users of *libgfapi*. Boolean value.

shorewall Boolean to specify whether puppet-shorewall integration should be used or not.

gluster::elastic

Under construction.

gluster::server

Main server class for the cluster. Must be included when building the GlusterFS cluster manually. Wrapper classes such as gluster::simple include this automatically.

`vip` The virtual IP address to be used for the cluster distributed lock manager.

`shorewall` Boolean to specify whether puppet-shorewall integration should be used or not.

gluster::host

Main host type for the cluster. Each host participating in the GlusterFS cluster must define this type on itself, and on every other host. As a result, this is not a singleton like the gluster::server class.

`ip` Specify which IP address this host is using. This defaults to the *$::ipaddress* variable. Be sure to set this manually if you're declaring this yourself on each host without using exported resources. If each host thinks the other hosts should have the same IP address as itself, then Puppet-Gluster and GlusterFS won't work correctly.

`uuid` Universally unique identifier (UUID) for the host. If empty, Puppet-Gluster will generate this automatically for the host. You can generate your own manually with *uuidgen*, and set them yourself. I found this particularly useful for testing, because I would pick easy to recognize UUID's like: *aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa*, *bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb*, and so on. If you set a UUID manually, and Puppet-Gluster has a chance to run, then it will remember your choice, and store it locally to be used again if you no longer specify the UUID. This is particularly useful for upgrading an existing un-managed GlusterFS installation to a Puppet-Gluster managed one, without changing any UUID's.

gluster::brick

Main brick type for the cluster. Each brick is an individual storage segment to be used on a host. Each host must have at least one brick to participate in the cluster, but usually a host will have multiple bricks. A brick can be as simple as a file system folder, or it can be a separate file system. Please read the official GlusterFS documentation, if you aren't entirely comfortable with the concept of a brick.

For most test clusters, and for experimentation, it is easiest to use a directory on the root file system. You can even use a */tmp* sub folder if you don't care about the persistence of your data. For more serious clusters, you might want to create separate file systems for your data. On self-hosted iron, it is not uncommon to create multiple RAID-6 drive pools, and to then create a separate file system per virtual drive. Each file system can then be used as a single brick.

So that each volume in GlusterFS has the maximum ability to grow, without having to partition storage separately, the bricks in Puppet-Gluster are actually folders (on whatever backing store you wish) which then contain sub folders– one for each volume. As a result, all the volumes on a given GlusterFS cluster can share the total available storage space. If you wish to limit the storage used by each volume, you can setup quotas. Alternatively, you can buy more hardware, and elastically grow your GlusterFS volumes, since the price per GB will be significantly less than any proprietary storage system. The one downside to this brick sharing, is that if you have chosen the brick per host count specifically to match your performance requirements, and each GlusterFS volume on the same cluster has drastically different brick per host performance requirements, then this won't suit your needs. I doubt that anyone actually has such requirements, but if you do insist on needing this compartmentalization, then you can probably use the Puppet-Gluster grouping feature to accomplish this goal. Please let me know about your use-case, and be warned that the grouping feature hasn't been extensively tested.

To prove to you that I care about automation, this type offers the ability to automatically partition and format your file systems. This means you can plug in new iron, boot, provision and configure the entire system automatically. Regrettably, I don't have a lot of test hardware to routinely use this feature. If you'd like to donate some, I'd be happy to test this thoroughly. Having said that, I have used this feature, I consider it to be extremely safe, and it has never caused me to lose data. If you're uncertain, feel free to look at the code, or avoid using this feature entirely. If you think there's a way to make it even safer, then feel free to let me know.

`dev` Block device, such as */dev/sdc* or */dev/disk/by-id/scsi-0123456789abcdef*. By default, Puppet-Gluster will assume you're using a folder to store the brick data, if you don't specify this parameter.

`raid_su` Get this information from your RAID device. This is used to do automatic calculations for alignment, so that the:

```
dev -> part -> lvm -> fs
```

stack is aligned properly. Future work is possible to manage your RAID devices, and to read these values automatically. Specify this value as an integer number of kilobytes (k).

raid_sw Get this information from your RAID device. This is used to do automatic calculations for alignment, so that the:

```
dev -> part -> lvm -> fs
```

stack is aligned properly. Future work is possible to manage your RAID devices, and to read these values automatically. Specify this value as an integer.

partition Do you want to partition the device and build the next layer on that partition, or do you want to build on the block device directly? The "next layer" will typically be lvm if you're using lvm, or your file system (such as xfs) if you're skipping the lvm layer.

labeltype Only *gpt* is supported. Other options include *msdos*, but this has never been used because of it's size limitations.

lvm Do you want to use lvm on the lower level device (typically a partition, or the device itself), or not. Using lvm might be required when using a commercially supported GlusterFS solution.

lvm_thinp Set to *true* to enable LVM thin provisioning. Read 'man 7 lvmthin' to understand what thin provisioning is all about. This is needed for one form of GlusterFS snapshots. Obviously this requires that you also enable *LVM*.

lvm_virtsize The value that will be passed to *–virtualsize*. By default this will pass in a command that will return the size of your volume group. This is usually a sane value, and help you to remember not to overcommit.

lvm_chunksize Value of *–chunksize* for *lvcreate* when using thin provisioning.

lvm_metadatasize Value of *–poolmetadatasize* for *lvcreate* when using thin provisioning.

fsuuid File system UUID. This ensures we can distinctly identify a file system. You can set this to be used with automatic file system creation, or you can specify the file system UUID that you'd like to use. If you leave this blank, then Puppet-Gluster can automatically pick an fs UUID for you. This is especially useful if you are automatically deploying a large cluster on physical iron.

fstype This should be *xfs* or *ext4*. Using *xfs* is recommended, but *ext4* is also quite common. This only affects a file system that is getting created by this module. If you provision a new machine, with a root file system of *ext4*, and the brick you create is a root file system path, then this option does nothing.

xfs_inode64 Set *inode64* mount option when using the *xfs* fstype. Choose *true* to set.

xfs_nobarrier Set *nobarrier* mount option when using the *xfs* fstype. Choose *true* to set.

ro Whether the file system should be mounted read only. For emergencies only.

force If *true*, this will overwrite any xfs file system it sees. This is useful for rebuilding GlusterFS repeatedly and wiping data. There are other safeties in place to stop this. In general, you probably don't ever want to touch this.

areyousure Do you want to allow Puppet-Gluster to do dangerous things? You have to set this to *true* to allow Puppet-Gluster to *fdisk* and *mkfs* your file system.

again Do you want to use *Exec['again']* ? This helps build your cluster quickly!

comment Add any comment you want. This is also occasionally used internally to do magic things.

gluster::volume

Main volume type for the cluster. This is where a lot of the magic happens. Remember that changing some of these parameters after the volume has been created won't work, and you'll experience undefined behaviour. There could be FSM based error checking to verify that no changes occur, but it has been left out so that this code base can eventually support such changes, and so that the user can manually change a parameter if they know that it is safe to do so.

bricks List of bricks to use for this volume. If this is left at the default value of *true*, then this list is built automatically. The algorithm that determines this order does not support all possible situations, and most likely can't handle certain corner cases. It is possible to examine the FSM to view the selected brick order before it has a chance to create the volume. The volume creation

script won't run until there is a stable brick list as seen by the FSM running on the host that has the DLM. If you specify this list of bricks manually, you must choose the order to match your desired volume layout. If you aren't sure about how to order the bricks, you should review the GlusterFS documentation first.

transport Only *tcp* is supported. Possible values can include *rdma*, but this won't get any testing if I don't have access to infiniband hardware. Donations welcome.

replica Replica count. Usually you'll want to set this to *2*. Some users choose *3*. Other values are seldom seen. A value of *1* can be used for simply testing a distributed setup, when you don't care about your data or high availability. A value greater than *4* is probably wasteful and unnecessary. It might even cause performance issues if a synchronous write is waiting on a slow fourth server.

stripe Stripe count. Thoroughly unsupported and untested option. Not recommended for use by GlusterFS.

layout Which brick layout to use. The available options are: *chained*, and (default). To generate a default (symmetrical, balanced) layout, leave this option blank. If you'd like to include an algorithm that generates a different type of brick layout, it is easy to drop in an algorithm. Please contact me with the details!

ping Do we want to include ping checks with *fping*?

settle Do we want to run settle checks?

again Do you want to use *Exec['again']* ? This helps build your cluster quickly!

start Requested state for the volume. Valid values include: *true* (start), *false* (stop), or *undef* (un-managed start/stop state).

gluster::volume::property

Main volume property type for the cluster. This allows you to manage GlusterFS volume specific properties. There are a wide range of properties that volumes support. For the full list of properties, you should consult the GlusterFS documentation, or run the *gluster volume set help* command. To set a property you must use the special name pattern of: *volume#key*. The value argument is

used to set the associated value. It is smart enough to accept values in the most logical format for that specific property. Some properties aren't yet supported, so please report any problems you have with this functionality. Because this feature is an awesome way to *document as code* the volume specific optimizations that you've made, make sure you use this feature even if you don't use all the others.

`value` The value to be used for this volume property.

gluster::mount

Main type to use to mount GlusterFS volumes. This type offers special features, like shorewall integration, and repo support.

`server` Server specification to use when mounting. Format is *:/volume*. You may use an *FQDN* or an *IP address* to specify the server.

`rw` Mount read-write or read-only. Defaults to read-only. Specify *true* for read-write.

`mounted` Mounted argument from standard mount type. Defaults to *true* (*mounted*).

`repo` Boolean to select if you want automatic repository (package) management or not.

`version` Specify which GlusterFS version you'd like to use.

`ip` IP address of this client. This is usually auto-detected, but you can choose your own value manually in case there are multiple options available.

`shorewall` Boolean to specify whether puppet-shorewall integration should be used or not.

Examples

For example configurations, please consult the examples/ directory in the git source repository. It is available from:

https://github.com/purpleidea/puppet-gluster/tree/master/examples

It is also available from:

https://forge.gluster.org/puppet-gluster/puppet-gluster/trees/master/examples

Limitations

This module has been tested against open source Puppet 3.2.4 and higher.

The module is routinely tested on:

- CentOS 6.5

It will probably work without incident or without major modification on:

- CentOS 5.x/6.x
- RHEL 5.x/6.x

It has patches to support:

- Fedora 20+
- Ubuntu 12.04+
- Debian 7+

It will most likely work with other Puppet versions and on other platforms, but testing on those platforms has been minimal due to lack of time and resources.

Testing is community supported! Please report any issues as there are a lot of features, and in particular, support for additional distros isn't well tested. The multi-distro architecture has been chosen to easily support new additions. Most platforms and versions will only require a change to the yaml based data/ folder.

Development

This is my personal project that I work on in my free time. Donations of funding, hardware, virtual machines, and other resources are appreciated. Please contact me if you'd like to sponsor a feature, invite me to talk/teach or for consulting.

You can follow along on my technical blog.

To report any bugs, please file a ticket at: https://bugzilla.redhat.com/enter_bug.cgi?product=GlusterFS&component=puppet-gluster.

Author

- github
- [@purpleidea](https://twitter.com/#!/purpleidea)
- https://ttboj.wordpress.com/