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/

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'],
}
```

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:

https://ttboj.wordpress.com/2012/08/23/how-to-avoid-cluster-race-conditions-or-how-to-implement-a-distribut

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?

Not until vagrant properly supports libvirt/KVM. I have no desire to use VirtualBox for fun.

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::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.

`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::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.

`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.

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

`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.

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.

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

settle Do we want to run settle checks?

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.

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 has been tested on:

- CentOS 6.4

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

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

It will most likely work with other Puppet versions and on other platforms, but testing under other conditions has been light due to lack of resources. It will most likely not work on Debian/Ubuntu systems without modification. I would really love to add support for these operating systems, but I do not have any test resources to do so. Please sponsor this if you'd like to see it happen.

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.

Author

- github
- @purpleidea
- https://ttboj.wordpress.com/