Ansible-Powered Red Hat Storage One

A hands-on experience

Dustin Black / Marko Karg
Storage Solution Architecture
2018-05-08
About This Workshop

- Hybrid Presentation Format
  - Slides
  - Audience-Driven Demos
  - Hands-On Opportunity
  - Attempts at Humor

*Please jump in with questions at any time*
<table>
<thead>
<tr>
<th>DEVELOPMENT MODEL</th>
<th>APPLICATION ARCHITECTURE</th>
<th>DEPLOYMENT AND PACKAGING</th>
<th>APPLICATION INFRASTRUCTURE</th>
<th>STORAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall</td>
<td>Monolithic</td>
<td>Bare Metal</td>
<td>Data Center</td>
<td>Scale Up</td>
</tr>
<tr>
<td>Agile</td>
<td>N-Tier</td>
<td>Virtual Services</td>
<td>Hosted</td>
<td>Scale Out</td>
</tr>
<tr>
<td>DEVOPS</td>
<td>MICROSERVICES</td>
<td>CONTAINERS</td>
<td>HYBRID CLOUD</td>
<td>SOFTWARE-DEFINED STORAGE</td>
</tr>
</tbody>
</table>
"Simplicity on the Other Side of Complexity"
DIY Software-Defined Storage

1. Evaluate storage software
2. Evaluate storage servers
3. Optimize for target workload
4. Conduct proof of concept
5. Procure and license at scale
6. Install
7. Manually deploy
8. Multiple support contracts
DEMO 1 - MANUAL DEPLOYMENT
Foundational Storage Stack

- x86 Server
- HDD / SSD
- RAID
- LVM PV
- LVM VG
- LVM ThinP
- NVMe
- Cache
- Filesystem
- Mountpoint
Foundational Storage Stack
Gluster Storage Volume

Multiplying the Complexity
Foundational Storage Stack

Let's Focus Here
A volunteer from the audience for LVM configuration?
Demonstration

Manual LVM setup

Desired configuration:

- A separate LVM stack with a thin-pool for every backing-device
- Proper data alignment to a 256 KB RAID stripe at all layers
- Fast device configured as LVMcache in writethrough mode
- XFS filesystem with proper data alignment
- Filesystem mounted with appropriate parameters
- Repeat for 24 nodes and 288 backing devices!
Demonstration
Manual LVM setup

Thin Provisioning:

- Physical extents are assigned from the PV to the thin pool instead of to the LV directly
- LVs are created instead with logical extents and arbitrary sizes
- Logical extents are mapped to physical extents only as writes occur
- This enables near-instantaneous copy-on-write snapshots and over-provisioning
Demonstration
Manual LVM setup

LVMcache

- A "fast" device is configured as a LVM cache pool
- The cache pool is then associated with a thick LV or with a thin pool
- LVM then intelligently buffers writes and keeps hot blocks in the cache for reads
- High-transaction workloads can be greatly improved
- Both writethrough and writeback modes are supported
Demonstration

Manual LVM setup

Data Alignment

- At the lowest block level, bytes are written in chunks of a particular size (generally 512 bytes for HDDs)
- RAID typically has a larger fundamental block size that is a multiple of the disks' block size
- Aligning the LVM and filesystem layers above to the RAID stripe size ensures transactions at the file level efficiently propagate to the disks, reducing latency
A Good LVM Structure is Non-Trivial

```
# lsblk /dev/sda -o NAME,SIZE,TYPE
NAME     SIZE   TYPE
sda      5G      disk
├─DATA-cpool_cdata   5G  lvm
│ └─DATA-data_thinpool_tdata 29G  lvm
│    └─DATA-data_thinpool_tpool 29G  lvm
│        └─DATA-data_thinpool 29G  lvm
│            └─DATA-data_thinvolume 20G  lvm
└─DATA-cpool_cmeta   8M  lvm
└─DATA-data_thinpool_tdata 29G  lvm
    └─DATA-data_thinpool_tpool 29G  lvm
        └─DATA-data_thinpool 29G  lvm
            └─DATA-data_thinvolume 20G  lvm
```

```
# lsblk /dev/vda -o NAME,SIZE,TYPE
NAME     SIZE   TYPE
vda      30G     disk
├─DATA-data_thinpool_tmeta 512M  lvm
│ └─DATA-data_thinpool      29G  lvm
│    └─DATA-data_thinvolume 20G  lvm
├─DATA-data_thinpool_tdata 29G  lvm
│    └─DATA-data_thinpool_tpool 29G  lvm
│        └─DATA-data_thinpool 29G  lvm
│            └─DATA-data_thinvolume 20G  lvm
└─DATA-data_thinpool_tdata_corig 29G  lvm
    └─DATA-data_thinpool_tpool 29G  lvm
        └─DATA-data_thinpool 29G  lvm
            └─DATA-data_thinvolume 20G  lvm
```

A Good LVM Structure is Non-Trivial
DEMO 2 - AUTOMATION WITH ANSIBLE
```python
#!/usr/bin/python

from __future__ import absolute_import, division, print_function
__metaclass__ = type

ANSIBLE_METADATA =
{'metadata_version': '1.1', 'status':

Python Modules

YAML Playbooks

- hosts: gluster_nodes
  become: yes
  tasks:
    - name: Create Gluster vol
      volume:
        action: create
        volume: "{{ volname }}"
        bricks: "{{ bricks }}"
        hosts: "{{ play_hosts }}"
```
Automation!

More to learn!
Ansible

- Repetitive tasks are formulated and executed in parallel
- Agentless architecture means any system is a potential Ansible target
- Complicated actions fueled by simple YAML playbooks
- Modules do the heavy lifting and are (generally) designed for idempotence
  - **Does not** limit your need to be an expert in whatever you are automating
    - And you need to know Ansible, too!

Your tasks are simpler, not easier!
Demonstration
Ansible deployment for the LVM setup

Desired configuration:

- Automate the LVM stack and filesystem configuration from the first demo
- Add arbiter brick filesystems carved from the fast devices
- Use the remaining fast device space for LVMcache
- Set the tuned profiles
- Is your playbook idempotent?
Demonstration
Ansible deployment for the LVM setup

YAML

- YAML Ain't Markup Language
- Human-friendly data serialization
- Whitespace indentation used for structure denotation
- Lists and arrays easily interpreted between languages like python
- Ansible plays are relatively easy to construct and understand
Demonstration
Ansible deployment for the LVM setup

Arbiter Bricks

- Even replica geometries risk split-brain problems
- Adding an odd replica to prevent problems can be prohibitively expensive
- Gluster I/O operations are tracked in metadata with the files
- Arbiter bricks serve as metadata-only stores, providing a lightweight investment for split-brain protection
- Can be separate nodes or chained
"Tune-D" profiles

- Linux has always had a lot of knobs to turn for tuning
- Left on your own, tuning can be daunting if not near impossible
- With tuned, workload-based engineering knowledge has been codified
- Applying a pre-defined tuned profile can make dozens of on-the-fly adjustments
Anatomy of an Ansible Playbook

- hosts: gluster_nodes
  become: yes
  any_errors_fatal: True

tasks:
  - name: Create data volume group
    vg:
      action: create
disks: "/dev/vda"
vgname: "DATA"
diskcount: 10
disktype: RAID
stripesize: "256"
dalign: "256"

- name: Create data thin pools
  lv:
    vg: "DATA"
    lv: "data_thinpool"
    size: "100%FREE"
    opts: "--thin --chunksize 256k --poolmetadatasize 1G"

hosts to act on

Tasks to perform on hosts

Task module

Module parameters

Plays are ordered, and each play runs in parallel on all hosts

Play

Playbook
DEMO 3 - AUTOMATION WITH GDEPLOY
Gdeploy

- Ansible-backed
- Gluster-specific modules
- Order-less configuration file
- The power of Ansible with the context of Gluster
- No need for Ansible expertise
- With simplicity comes limited flexibility
Ansible vs. Gdeploy

Ansible YAML

- hosts: gluster_nodes
  become: yes
  tasks:
    - lvg:
      vg: DATA
      pvs: /dev/vda
    - lvol:
      vg: DATA
      thinpool: data_thinpool
      size: 29G
    - lvol:
      vg: DATA
      lv: data_lv

...
Demonstration
Using gdeploy frontend for Ansible

Desired configuration:
- Automate everything from demo 2
- Create a 4-node Gluster trusted storage pool
- Create a Gluster distribute-replicate volume with chained arbiter bricks
- Configure NFS-Ganesha with high-availability
- Are your arbiter bricks on the correct nodes?
Demonstration
Using gdeploy frontend for Ansible

Trusted Storage Pools

● Gluster nodes have a "peering" system to establish and modify pools of storage servers
● Peers must be established before volumes can be created
● Peers share status via TCP protocols
● Peers maintain on-disk "volfile" definitions of the translator stacks making up volumes
Demonstration
Using gdeploy frontend for Ansible

Chained Arbiters

- Instead of a dedicated arbiter node, we can use arbiter chaining for better efficiency
- In a 2x replica volume with more than 2 nodes, an arbiter brick for each subvolume is placed on a node that is not part of the replica set
- The arbiter brick should be as fast as your fastest device (including cache)
Demonstration
Using gdeploy frontend for Ansible

PCS High-Availability

- Pacemaker/Corosync Configuration System
- Used to enable VIP migration and session failover for NFS-Ganesha
- Configuration is non-trivial, but aided by built-in Gluster tooling
- High availability and load balancing are not the same thing
All of That Just to Get a Good Volume Config

```
# gluster vol info

Volume Name: myvol
Type: Distributed-Replicate
Volume ID: cc1b8e90-26c6-46c0-9302-58801b608263
Status: Started
Snapshot Count: 0
Number of Bricks: 2 x (2 + 1) = 6
Transport-type: tcp
Bricks:
Brick1: 192.168.122.19:/gluster/brick1/brick1
Brick2: 192.168.122.20:/gluster/brick1/brick1
Brick3: 192.168.122.21:/gluster/arbiter-brick1/arbiter-brick1 (arbiter)
Brick4: 192.168.122.21:/gluster/brick1/brick1
Brick5: 192.168.122.22:/gluster/brick1/brick1
Brick6: 192.168.122.20:/gluster/arbiter-brick1/arbiter-brick1 (arbiter)
Options Reconfigured:
transport.address-family: inet
nfs.disable: on
```
INTRODUCING

RED HAT
STORAGE ONE
WHAT IS RED HAT STORAGE ONE?
A hardware/software/support offering pre-configured for a target workload

- **4-24 servers**
  pre-configured for a workload personality

- **30 minutes or less**
  to get up-and-running

- **Fulfilled by Supermicro**
  or accredited reseller

- **Pre-loaded Red Hat® Gluster Storage®**
  and a workload-specific quick-deploy utility

- **Shipped and supported (L1/L2)**
  by Supermicro
SIMPLIFYING SDS DEPLOYMENT

Traditional “DIY” software-defined storage

Evaluate storage software  Evaluate storage servers  Optimize for target workload  Conduct proof of concept

Procure and license at scale  Install  Manually deploy  Multiple support contracts

Red Hat Storage One

Optimization-tested, self-configuring, and ready in minutes
120TB to 1.5PB (usable) of resilient Red Hat® Gluster Storage

Single part number for hardware software and support
CURRENT WORKLOAD IDENTITIES

General NAS and content repositories

General NAS
User directories, mix of small and large files in NFS, SMB, GlusterFS-native folders

Content repositories
Photos, rich images, and videos at large scale
RHS One Intro

- Software-defined storage isn’t simple
- Compare responsibilities with traditional storage:

<table>
<thead>
<tr>
<th></th>
<th>Traditional Storage</th>
<th>Software-Defined Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>Vendor</td>
<td>OS Admins?</td>
</tr>
<tr>
<td>Administration</td>
<td>Storage Admins</td>
<td>OS Admins? Storage Admins?</td>
</tr>
<tr>
<td>Day-to-Day Operation</td>
<td>Storage Admins</td>
<td>End-user? Customer?</td>
</tr>
</tbody>
</table>
SDS Isn't Simple?

- Optimal setup is tricky
  - A myriad of "compatible" hardware choices
  - LVM stack and data alignment is complicated
  - Multiple Gluster geometries to choose from
  - 311 volume options with Gluster
- Easier to define the expected workload
  - Large files
  - Video streams
  - Small files
  - Databases
RHS One is Built on Experience

- Endless test cycles to refine workload categories and performance characteristics
- Massive amounts of data collected on which to base architectural decisions
- Years of experience in critical enterprise deployments
- Extremely close feedback loop with engineering and support
What's in the Box?

The RHS One Quick-Deploy System is Built On:

- Ansible
- Gdeploy
- Python
- YAML
- gluster-zeroconf
DEMO 4 - RED HAT STORAGE ONE
Extra Challenges for RHS One

- Networking (bonding, device naming, subnets, hostnames)
- LVM stack with variable disk sizes and backends
- Node discovery
- Calculated arbiter sizes and locations
- Efficient fast device allocation
- Portability among hardware models
- Variable client access method
- Simplified step-by-step UI
The Gluster Colonizer Deployment Model

- User Input
- Gdeploy Modules
- OEMID Flavor File
- OEMID Verify File
- Ansible Templates
- Ansible Automation
- gluster-colonizer
- gluster-zeroconf
- Host Inventory
See the Code Upstream

- The Gluster Colonizer project is the technical basis for RHS One
- Currently handles:
  - Rep 2 + Chained Arbiter
  - Disperse 4+2 (erasure coding)
- New OEMID file sets can enable more hardware models, deployment types, and workloads

https://github.com/gluster/gluster-colonizer
Demonstration
Complete workload-based deployment automation

Desired configuration:

- 4-Node deployment
- Hostnames and IPs configured
- Proper foundational storage stack with data alignment
- Data bricks backed with lvmcache
- Gluster replica deployment with chained arbiter bricks on fast devices
- Key and password updates
- NFS-Ganesha with HA
Welcome to the Red Hat Storage One Gluster deployment tool!

This node will be configured as the deployment master for your Gluster storage pool. Before proceeding, please ensure that all RHS One Gluster nodes are connected to the management network infrastructure and are booted.

Do you wish to continue? [Y/n]
HANDS-ON OPPORTUNITY
Hands-On with RHS One

Get the simulation demo in this GitHub private branch:
https://github.com/dustinblack/gluster-colonizer/tree/demo

Requirements:
- python 2.7
- python netaddr
- python pyyaml
- asciinema (in $PATH)

Run from resources/demo/:
./gluster-colonizer-demo.py -f g1-demo.yml

Demo Inputs:
As this is a simulation, the inputs are arbitrary and up to you. A few selections, such as Client method, have been locked to one option. Validations are active, so entries like hostnames and IP addresses must be in correct formats.

Answering 'no' at most Y/N prompts will abort. Ctrl-c will also abort.

The simulation will not make any system changes.
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHat