redhat.

RED HAT GLUSTER TECHSESSION
CONTAINER NATIVE STORAGE
OPENSHIFT + RHGS

MARCEL HERGAARDEN
SR. SOLUTION ARCHITECT, RED HAT BENELUX
April 2017

AGENDA

- Why OpenShift?
- The Journey So Far for OpenShift Storage
- What does Red Hat ship today
- Future outlook

Why OpenShift?

Linux for App/Dev-Centric IT

- The “cluster” is the computer
- The scope of the OS is now the “cluster”- schedules workloads across clusters (unified compute substrate)
- Orchestrators (like Kubernetes, Marathon, Swarm) are the heart of this new OS

Welcome to OpenShift !

- Red Hat's new **OS** product:
  -> Kubernetes for enterprise + Add-ons

- Effectively a next generation RHEL!
- Why?
  - Consistent application run time across the hybrid cloud
  - Anti lock-in abstraction layer
  - Runs everywhere including proprietary clouds, Virtual and OSP! (similar like RHEL does)

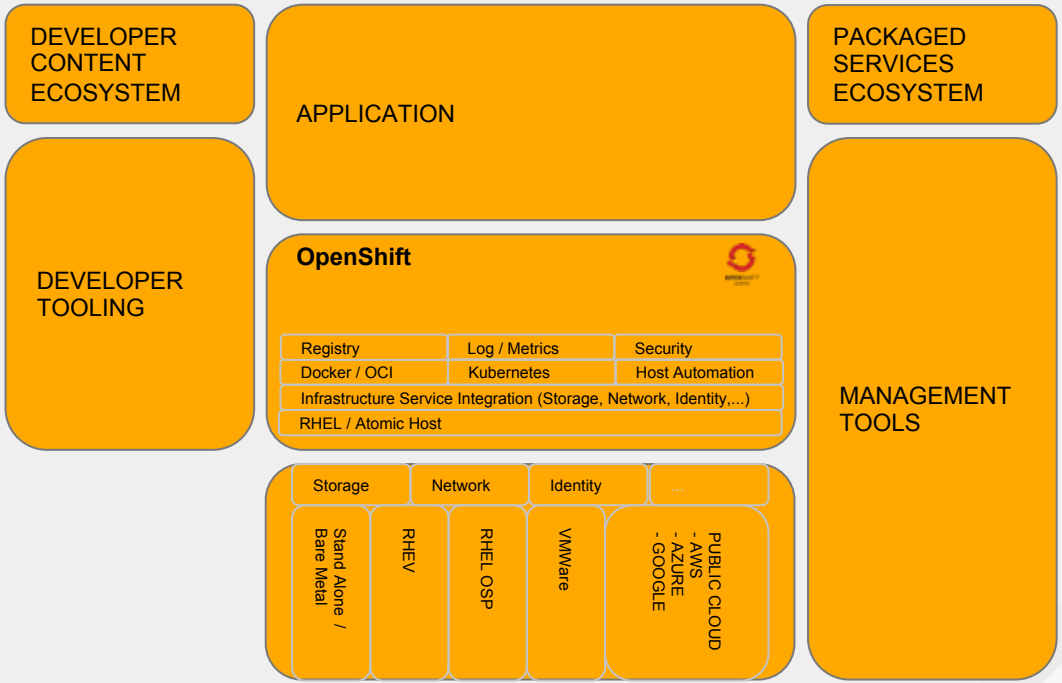
Abstraction Across Hybrid Cloud

OpenShift provides a scale-out, "**cluster is the computer platform"** to deploy fully-orchestrated multi-container applications.

Built on RHEL Atomic Host in the **immutable infrastructure paradigm**, the docker project, OCI, etcd, kubernetes, systemd.

Application is defined in abstraction from Infrastructure provider details, works across different cloud providers, integrates with infrastructure services.

Fully Open Source, Standards-based, **Pluggable**.

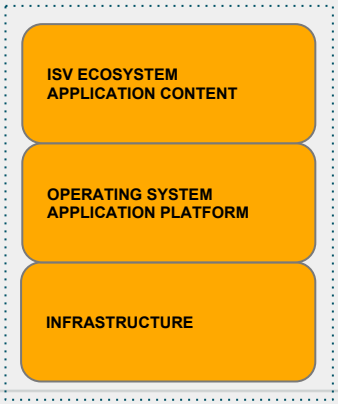


RHEL History
Breaking Vertical Integration

MAINFRAME

Complete vertical integration

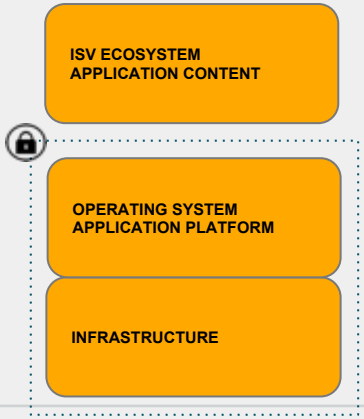
Vendor-controlled HW/OS/Ecosystem.



UNIX

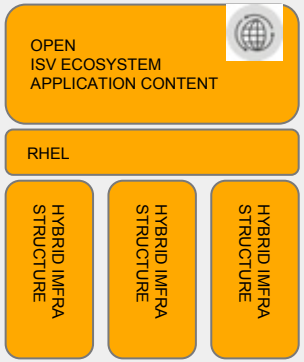
Vertical integration of infrastructure & app platform

Semi-open ecosystem.



RHEL

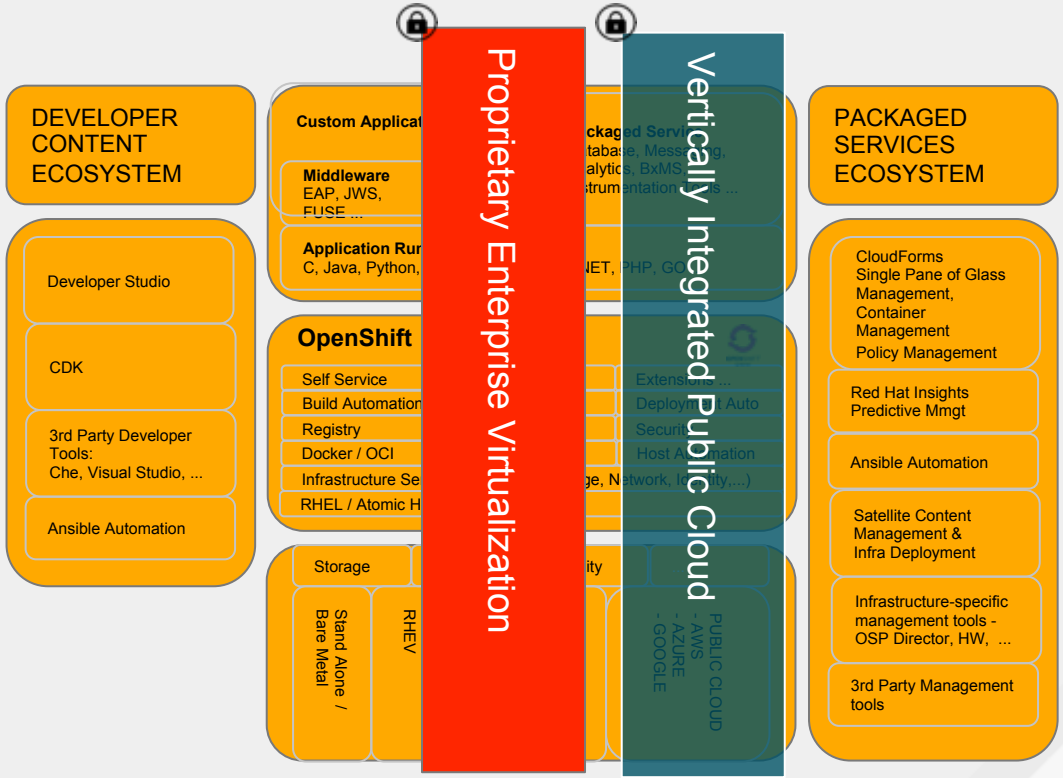
Completely Open HW and ISV ecosystem with RHEL as the neutral enterprise app platform



Breaking Vertical Integration with OpenShift

Public cloud & proprietary private cloud are driving vertical integration and lock-in with pseudo-standards.
- Just like UNIX.

Red Hat offers a neutral runtime for an open ecosystem on hybrid infrastructure, disrupting the vertical integration of proprietary vendors.
- Just like RHEL.



The Journey So Far

Container Storage Use Cases

Local Storage for Container Images

Registry

Persistent Storage

Registry

Container Image Repository

- Needs strong read-after-write consistency guarantees
  - Multiple registries write to the same back-end
- Few Terabytes suffice for most enterprise registries
  - Red Hat Container Registry ~ 3 TB
- OpenShift uses NFS as the default registry backend
  - Replace NFS with Red Hat Gluster Storage
- Registries often need to be federated for enterprises with multiple sites

Persistent Storage

- **Containers became popular with stateless apps**
  - This was only true for a few months
  - Enterprises adopting containers need persistent storage

- A new application packaging does not do away with the need for needing persistent storage for storing application state, data and config.

- When containers go away and come back they still need the data-store

Red Hat & Kubernetes Storage Orchestration

- Persistent Storage is key for enterprise container adoption
- Red Hat championed the cause for storage in upstream Kubernetes
    - Red Hat engineers worked through spring and summer of 2015 to add storage orchestration functionality to Kubernetes
    - Added drivers for a variety of storage layers
    - OpenShift 3.1 release in November 2015 had storage orchestration built-in and had the richest set of storage drivers.

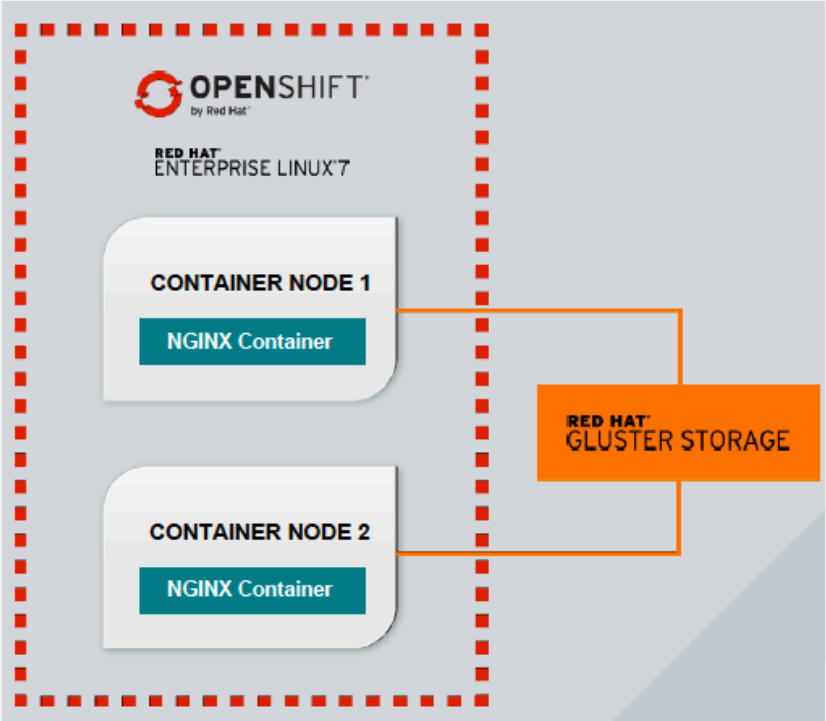
Nov 2015
OpenShift 3.1 - Container Ready Storage

Storage provided by a dedicated Red Hat Gluster Storage Cluster over the network

Storage consumed using Kubernetes PV and PVCs

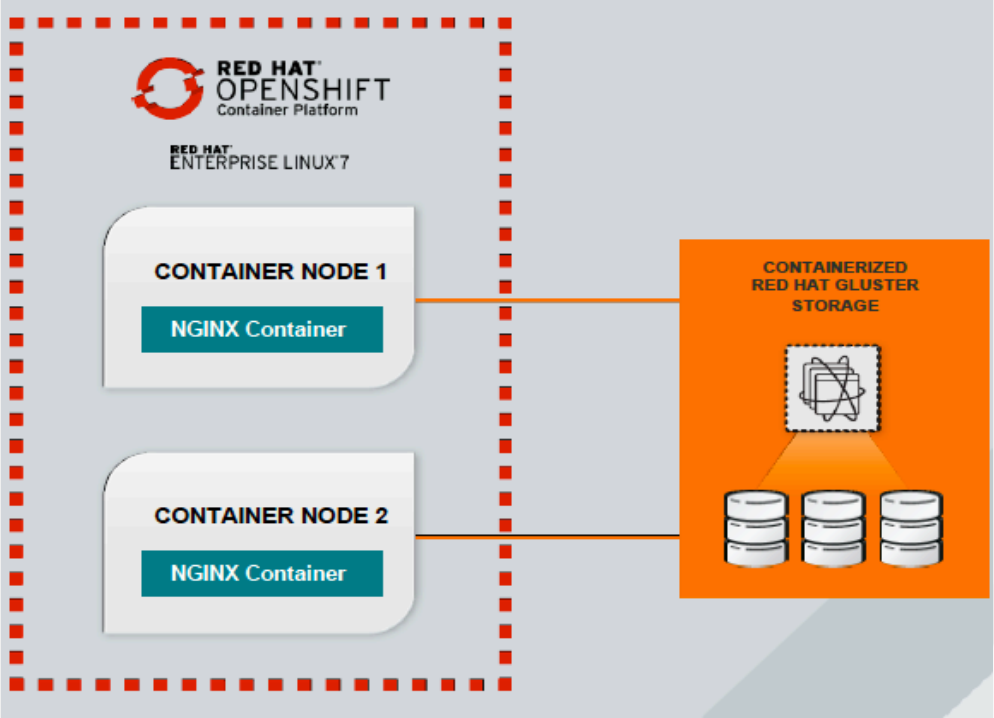
Red Hat Gluster Storage can run on bare metal or VMs

Enterprise Grade Storage in addition to in-box NFS, local storage.
Storage and compute can scale independently



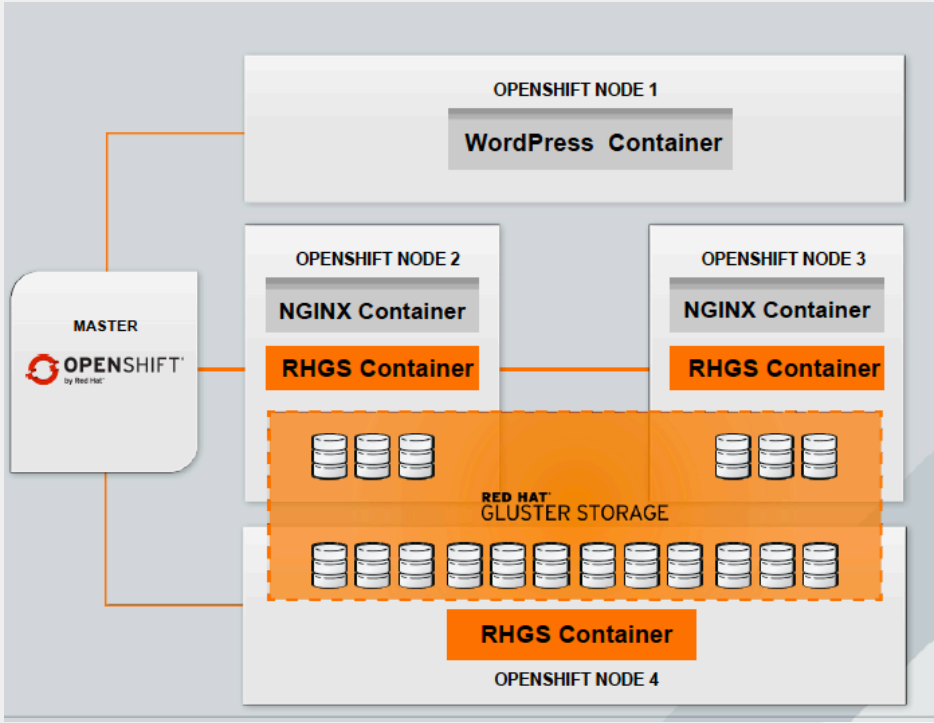
March 2016
Containerization of Red Hat Gluster Storage

- As a first-step in our journey to create the Container-Native Storage solution we containerized Gluster
- Pushed it to Red Hat Container Registry
- Validated that it could pool and serve out storage from local hosts



July 2016
Launch Container-Native Storage

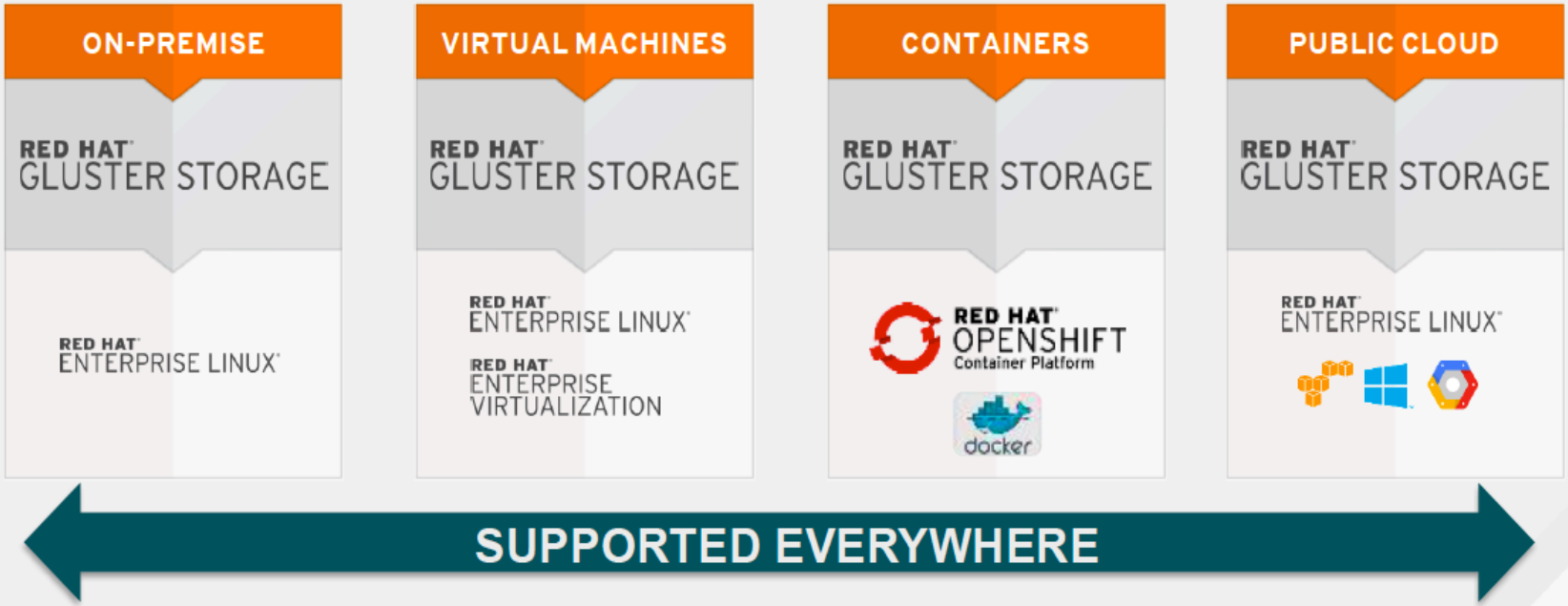
- Red Hat Gluster Storage runs inside OpenShift in a container (in Kubernetes pods)
- Application and storage containers can be co-located
- There's exactly one RHGS container per host
- Uses host networking, not overlay
- Use OpenShift templates to deploy
- Use a dynamic volume allocator (Heketi) to automate volume creation



What is available today

RHGS is supported Everywhere!

- Same software bits across on-prem, VMs, containers and all three public clouds
- Applications can be ported across deployments without expensive re-writes
- Close integration with RHEL, RHV, and OpenShift



Container Storage Deployments

- **Container Ready Storage, serving out storage to OpenShift**
  - RHGS in stand-alone bare-metal storage clusters
  - RHGS inside VMs on bare-metal hosts using local disks
  - RHGS inside VMs fronting Enterprise Storage Arrays
    - RHGS sits between OpenShift and Storage Arrays
    - LUNs are served out as RHGS bricks
  - RHGS in AWS, Azure, Google Cloud

- **Container-Native Storage**
  - RHGS runs containerized, inside OpenShift Container Platform
  - CNS runs anywhere OpenShift Container Platform runs!

Red Hat

OpenShift + Gluster Storage

How does it all work

What is CNS … and Heketi

- **CNS**: Container Native Storage
  Providing dynamic persistent storage for OpenShift with GlusterFS in a hyper-converged fashion

- **Heketi**: the high-level service interface to Gluster
  Manages the lifecycle of volumes in multiple Gluster clusters.

Openshift ⇔ Heketi ⇔ Gluster

Main components of CNS

- **OpenShift/Kubernetes**
  - dynamic glusterfs provisioner
  - glusterfs plugin
- **Heketi**
  - high-level service interface for Gluster volume lifecycle management
- **glusterFS**
  - one or more glusterfs clusters
  - running hyper-converged in OpenShift
- **cns-deploy**
  - tool to deploy gluster and heketi into an existing openshift cluster

Persistent Storage "*jargon*" in OpenShift

- **Pod**: group of one or more containers that form an entity
- **Persistent Volume** (PV): to be mounted by application pod
- **Provisioner**: to provide PVs upon request
- **Plugin**: mechanism to mount the PV, referenced in PV
- **Persistent Volume Claim** (PVC): mechanism for a user to request a PV
- **Access types** for volumes:
  - RWO - read write once (single node)
  - RWX - read write many (multiple nodes)
  - ROX - read only many (multiple nodes)
- **Flavors** of provisioning: dynamic and static

Static Provisioning *(pre-OCP 3.4)*

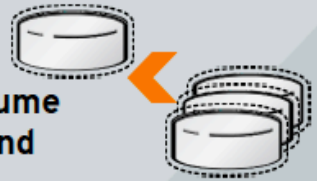
- Storage Admin pre-creates storage volumes
- OCP Admin pre-creates a set of persistent volumes
- Static provisioner choses existing PV based on type and size from PVC

Static Provisioning (pre OCP 3.4)

1 Multiple volumes created and registered with OpenShift

2 Submit Persistent Volume Claim

3 An available persistent volume is picked out of the pool and bound to the persistent volume claim



Flow of commands (SP)

```
# static provisioning

# create endpoints
oc create -f sample-gluster-endpoints.yaml
oc get endpoints

# create gluster service
oc create -f sample-gluster-service.yaml
oc get service

# create gluster volume
heketi-cli volume create --size=100 --persistent-volume-file=pv001.json

# create openshift pv using edited pv001 file:
oc create -f pv001-mod.json
oc get pv

# create PVC
oc create -f pvc.json
oc get pv
oc get pvc

# create app
oc create -f app.yml
```

PVC for Static Provisioner

```
obnox ~ cns cat sp-pvc.json
{
   "apiVersion": "v1",
   "kind": "PersistentVolumeClaim",
   "metadata": {
     "name":  "glusterfs-claim"
  },
   "spec": {
     "accessModes": [
       "ReadWriteMany"
    ],
     "resources": {
       "requests": {
         "storage": "100Gi"
      },
       "selector": {
       "matchLabels": {
         "storage-tier": "gold"
      }
    }
   }
  }
}
```

Dynamic Provisioning *(in general)*

- A Storage Class (SC):
  - Created by admin
  - Describes the storage properties
  - References a (dynamic) provisioner

Dynamic Provisioning *(in general)*

- A Storage Class (SC):
  - Created by admin
  - Describes the storage properties
  - References a (dynamic) provisioner

- PVC (created by user): references to a Storage Class

Dynamic Provisioning *(in general)*

- A Storage Class (SC):
  - Created by admin
  - Describes the storage properties
  - References a (dynamic) provisioner

- PVC (created by user): references to a Storage Class
- Provisioner (from SC): creates PV of the requested size

Dynamic Provisioning *(in general)*

- A Storage Class (SC):
  - Created by admin
  - Describes the storage properties
  - References a (dynamic) provisioner

- PVC (created by user): references to a Storage Class
- Provisioner (from SC): creates PV of the requested size

- User can then mount the PV within application pod

PV creation: GlusterFS dynamic provisioner workflow

- **PVC references to the glusterfs provisioner**
  - GlusterFS provisioner extracts details from PVC
  - Provisioner tells Heketi to create a volume of given size and class
    - Heketi looks for a Gluster cluster that can satisfy the request
    - If found, Heketi tells the Gluster instance to create the volume
      - Gluster creates the requested volume
    - Heketi hands Volume details back to the Provisioner
  - Provisioner creates PV and puts the Gluster volume details into it
  - Provisioner puts Glusterfs as the mount plugin into the PV
  - Provisioner returns PV to the caller/enduser

- **Caller receives the PV and can now use it within a pod**

Dynamic Provisioning (OCP 3.4)

1 Submit Persistent Volume Claim

2 OpenShift requests volume to be created

4 OpenShift binds persistent volume to persistent volume claim request

3 Persistent volume is created by storage system and registered with OpenShift

RED HAT
GLUSTER STORAGE

redhat.

GlusterFS mount plugin

- the OpenShift HOST has glusterfs-client installed
- the OpenShift HOST mounts the gluster volume
- the Gluster mount of the HOST is bind-mounted into the application container

Demo of Dynamic Provisioning

Details on Containerization

Gluster Cluster inside OpenShift

- Gluster is running as pods
- Aggregating HOST disk devices into volumes
- Gluster storage nodes are tied to OpenShift nodes (using the hosts disks devices...)
- Gluster nodes on some, not necessarily on all OpenShift worker nodes
- Application pods can be on the same OpenShift nodes

about Gluster containers

- **containerized systemd**
  - (running multiple processes: glusterd, brick servers, ...)
- **privileged container** (systemd, access host /dev, ...)
- **startup script**
- **host /dev bind-mounted**
- **bind-mount config**
  - /etc/glusterfs, /var/lib/glusterd, /var/log/glusterfs
- **uses host network**
  - gluster ip’s need to be constant
  - gluster config tied to the node
  - performance

about Heketi

- High-level service interface for managing the lifecycle of gluster volumes
- RESTful API and CLI ("heketi-cli")
- Manages either one or multiple Gluster clusters
- Capabilities: create, expand, delete volumes (more to come)
- Hides the “nitty gritty” details of volume creation from caller
- Simply takes the volume size and desired durability type
- Automatically finds cluster and disks to satisfy the request
- Stores its state within a database (currently Bolt DB)
- https://github.com/heketi/heketi

about the Heketi container

- Runs as a single container pod
- Heketi Pod can move within the cluster
- Database needs to be persistent

  ⇒ therefore stored within a Gluster volume ☺

Warning

Inside a Heketi managed cluster,

Never mess with Gluster Volumes manually !

CNS Deploy

Set it all up by using: cns-deploy

- New in OCP/CNS 3.4
- Set it all up by using a single command: “cns-deploy”
- Project / community: https://github.com/gluster/gluster-kubernetes
- Takes topology file to describe Disk devices, Gluster nodes and Heketi
- Deploys the Gluster Cluster (upon request)
  - Gluster is deployed as a DaemonSet
- Deploys the Heketi pod

Pre-requisites before running cns-deploy

- Openshift cluster installed
- One or more masters
- A cluster admin user must be created and logged in as that user
- Three or more worker nodes local storage to host gluster nodes
- The Gluster enabled nodes need:
  - firewall ports open: 24007,24008,2222,49152:49664
- All openshift nodes need to have the glusterfs-client installed
- The heketi pod can run anywhere, its logs should be persistent
- Need a management machine from where to run oc commands, heketi-cli, and cns-deploy
  - (e.g. a Master node), will need to have cns-deploy installed

Demo of cns-deploy

New: GID-security

- Originally only root UID==GID==0 could access a PV
- As of CNS 3.4, the dynamic provisioner automatically assigns a GID (otherwise unused) to a PV
- All users belonging to this group have write access
- The requesting user makes sure that his pod runs under a user belonging to this group
- Admin configures range from which the provisioner choses the GID in the SC

Scaling

- **Layout of gluster processes**
  - Glusterd
  - Brick daemons
  - Possibly more daemons (snapshot, quota, …)
- ⇒ **Memory Requirements**

  ~ 32GB of free/reserved memory per Gluster node (pod) for hosting 100 replica-3 Volumes
- Current glusterd has a limit compute resources at around 100 volumes

  Note: mounting gluster volumes also requires some memory
  - reserve similar amounts for the hosts to mount

Future Outlook

Key Asks from Customers

- Larger RHGS volume density per 3-node cluster
- Broader support for RWO workloads
- S3 access

Development Roadmap - Overview

- Scaling improvements (memory requirements/complexity) Gluster
- Improved day-2-day maintenance support
- Gluster as OpenShift registry
- RWO support with Gluster Block access (iscsi)
- S3-object access for application pods

Brick Multiplexing

- **Today there is one process per Gluster brick**
- **With BrickMux: run many bricks within one glusterfsd process**

Each process will consume one TCP port, instead of each brick doing so.

Proper RWO Support using iSCSI

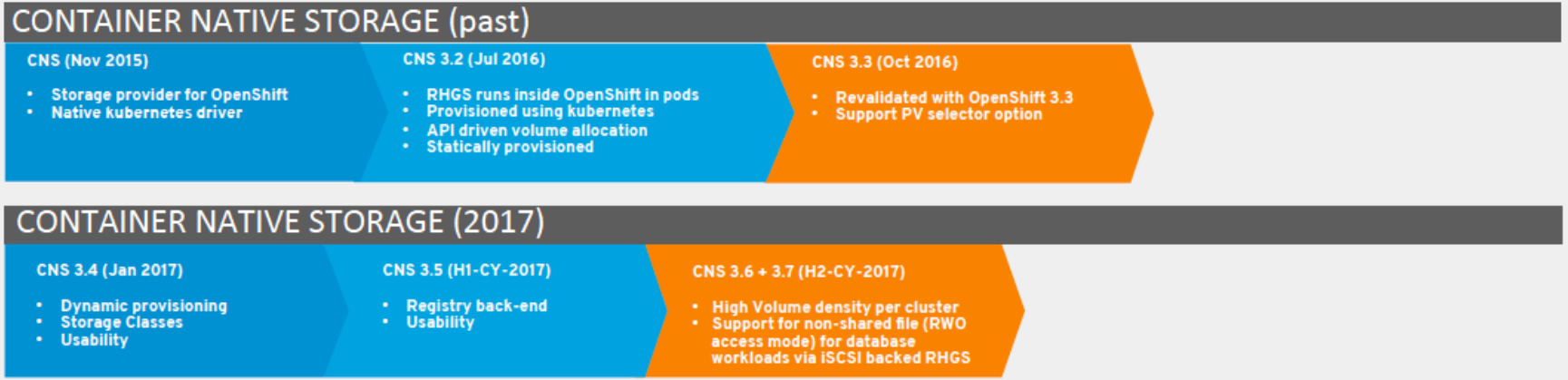
- A single file in a Gluster volume exported as an iSCSI target
- Host mounts iSCSI target
- iSCSI mount is bind-mounted into application pod
  - ⇒ For the pod it is no different than file!

- ⇒ better scalability in number of volumes
- ⇒ better fit for performance requirements from RWO workloads
- ⇒ better fit for access model

S3 Object Access

- Gluster Swift is a service sitting on top of a Gluster mount
- Provides an S3-plugin for S3-API access (put/get/delete)
- Directories in a Gluster vol will represent s3-buckets
- Via Gluster Swift/S3-plugin, these buckets will be made available to application pods

OpenShift + RHGS roadmap




THANK YOU
redhat.