Gluster and OpenShift

Sayan Saha and Michael Adam

OpenShift Commons Briefing
2017-03-09

Agenda

- Gluster
- Storage use cases in OpenShift
- The way to Container Native Storage
- How it all works
- Roadmap

Gluster

Gluster

- Software-defined storage
- Scale-out file storage
- Highly available
- Easy to set up
- Easy to administer
- Very flexible
- Access:
  - Native fuse POSIX file system
  - NFS
  - SMB
  - iscsi (on file) (new)
  - Object: S3 / swift via gluster-swift (new)

Gluster

- https://gluster.org
- https://github.com/gluster

Gluster

- Volumes composed of local FS directories (bricks)
- Different “durability” types: *replicate*, *distribute*, *disperse* (ec), …
- Flexibility and feature-richness due to architecture of a stack of translators
- Example of features:
  - Encryption
  - Snapshots (user-serviceable)
  - Geo-replication
  - Quota
  - …
- Layout of multiple daemons for bricks, glusterd, quota, ...

Storage use cases in OpenShift

- Persistent storage for application containers
- Registry

Persistent Storage

- Containers are stateless, ephemeral animals
  - Can be brought down and up again arbitrarily
- But the apps running in containers *need* persistent storage for their
  - State
  - Config
  - Data
- ⇒ Gluster seem like a natural fit

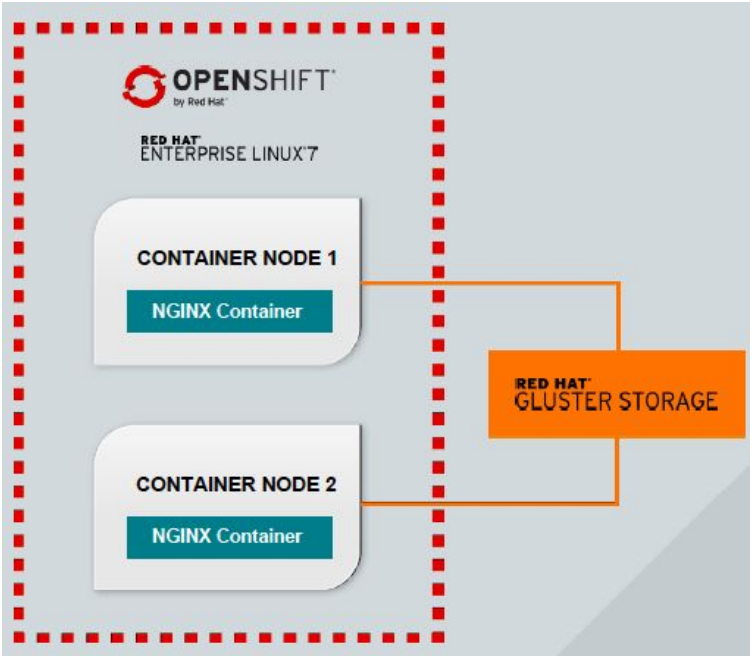
Registry

- Container image repository
- OpenShift uses NFS as registry backend by default
- ⇒ Replace with Gluster backend?

Towards Container Native Storage

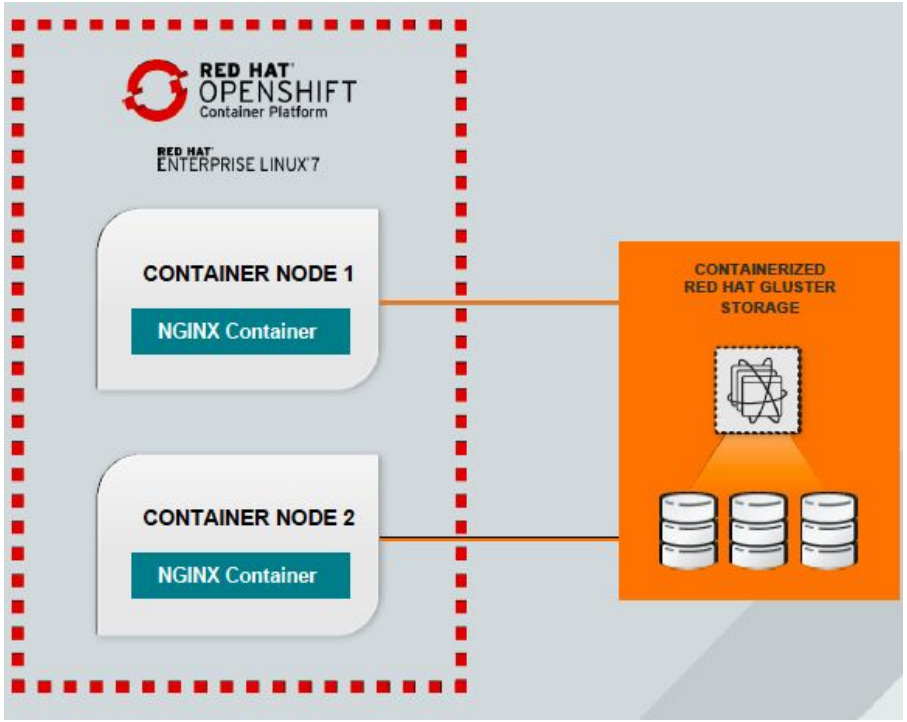
Step 1: Container Ready Storage (Nov 2015)

- Container Orchestration added in 3.1
- Container Ready Storage made use of external Gluster:



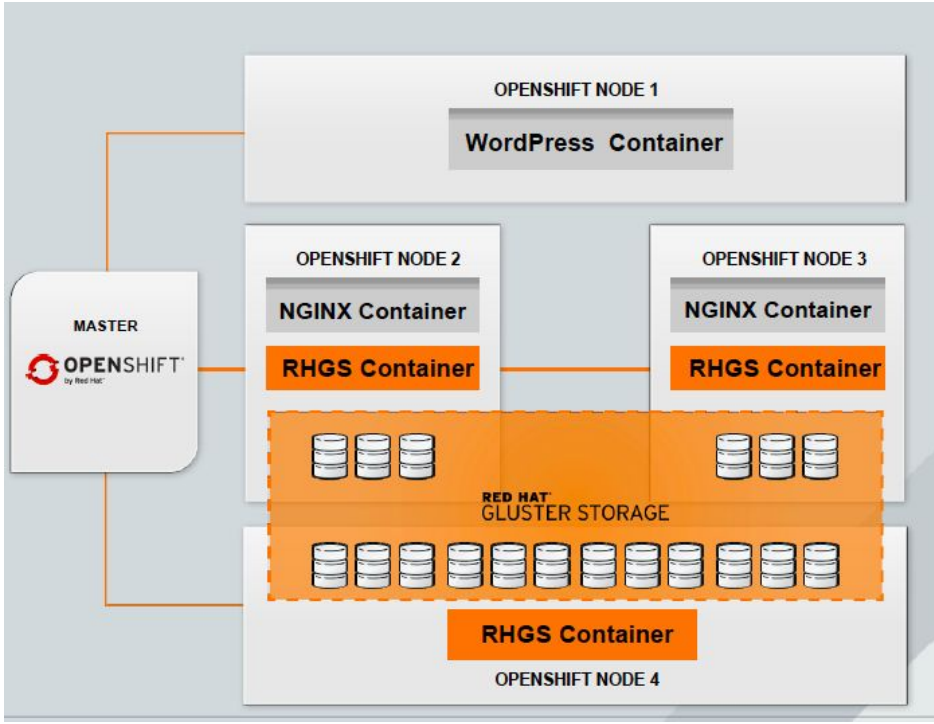
Step 2: Containerized Storage (March 2016)

- Put Gluster into containers
- Containerized Gluster outside of the openshift cluster running the aps



Step 3: Container Native Storage (July 2016)

- Gluster Storage runs inside OpenShift in containers
- Apps and Storage *can* be co-located
- Uses host network (not overlay)
- Management interface (heketi) is used for dynamic volume creation



How it all works

Components

- OpenShift/kubernetes
  - dynamic GlusterFS provisioner
  - GlusterFS mount plugin
- Heketi
  - high-level service interface for gluster volume lifecycle management
- GlusterFS:
  - one or more glusterfs clusters
  - running hyper-converged in OpenShift
- Gk-deploy / cns-deploy:
  - tool to deploy gluster and heketi into an existing OpenShift cluster

Persistent Storage in OpenShift today

- 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 OpenShift 3.4)

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

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 (since 3.4) - in general

- a storage class (SC):
  - Created by admin
  - describes the storage
  - references a (dynamic) provisioner
- PVC (by user): references SC
- provisioner from SC: creates PV of requested size / type / ...
- user can mount the PV in application pod

PV Creation: glusterfs dynamic provisioner

- PVC (created by user) references the glusterfs provisioner
  - glusterfs provisioner extracts details from PVC
  - provisioner tells heketi to create a volume of given size and type
    - heketi looks for a gluster cluster that can satisfy this request
    - if found, heketi tells the gluster instance to create the volume
      - gluster creates a volume
    - Heketi hands volume back to 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
- PVC is bound to the PV and can later be used in a pod by the user

Example of a StorageClass

```
obnox ~ cat glusterfs-sc.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-container
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://127.0.0.1:8081"
  restuser: "admin"
  secretNamespace: "default"
  secretName: "heketi-secret"
obnox ~
```

Example of PVC

```
obnox ~ cns cat glusterfs-pvc-claim1.yaml
{
  "kind": "PersistentVolumeClaim",
  "apiVersion": "v1",
  "metadata": {
    "name": "claim1",
    "annotations": {
        "volume.beta.kubernetes.io/storage-class": "gluster-container"
    }
  },
  "spec": {
    "accessModes": [
      "ReadWriteOnce"
    ],
    "resources": {
      "requests": {
        "storage": "4Gi"
      }
    }
  }
}
obnox ~ cns
```

Example of APP using PVC

```
obnox ~ cns cat app.yml
apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
    - image: busybox
      command:
        - sleep
        - "3600"
      name: busybox
      volumeMounts:
        - mountPath: /usr/share/busybox
          name: mypvc
  volumes:
    - name: mypvc
      persistentVolumeClaim:
        claimName: claim1
obnox ~ cns
```

Flow of commands (DP)

```
# dynamic provisioning

# creating a storage class

oc create -f glusterfs-storageclass.yaml
oc describe storageclass gluster-container

# detail: creating a secret
oc create -f glusterfs-secret.yaml

# creating a PVC (and thereby PV)

oc create -f glusterfs-pvc-claim1.yaml

oc describe pvc claim1
oc get pv,pvc
oc get endpoints,service

# using claim in a pod:

oc create -f app.yml
oc get pods

# delete claim
oc delete pvc claim1
```

GlusterFS mount plugin

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

Demo of Dynamic Provisioning

About heketi

- high-level service interface for managing the lifecycle of gluster volumes
- RESTful API and cli ("heketi-cli")
- manages one or several gluster clusters
- can create, expand, delete volumes (more coming)
- hides nitty gritty details of volume creation from caller
- just takes size and desired durability type
    - (currently only replicate is supported in CNS)
- automatically finds cluster and disks to satisfy the request
- stores its state in a database (currently Bolt)
- https://github.com/heketi/heketi

About the heketi container

- single container
- can move in the cluster
- database needs to be persisted
  - ⇒ currently stored in a gluster volume

WARNING

In a heketi-managed cluster, don’t mess with the volumes manually!

How to set it all up? Cns-deploy / gk-deploy

- New in CNS 3.4
- Set it all up in a single command
- 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 heketi pod

Demo of cns-deploy

Roadmap

Roadmap

- 3.5
  - GlusterFS as registry backend
  - Improved day-2-day maintenance (remove disk …)
- 3.6
  - Improved RWO support with gluster-block provisioner (iscsi)
  - Scalability improvements
- 3.7+
  - Support for S3-object access from pods
  - Possibly Gluster with S3 as improved backend for registry

Join The Conversation

redhat.com/ContainerStorage

- Demos
- Analyst Opinion

redhatstorage.redhat.com

- Thought Leadership
- Announcements

@RedHatStorage

- Storage Days
- Promotions