Table of Contents

* [Red Hat OpenShift Container Storage Deployment Lab](https://www.opentlc.com/labs/ocp_storage_cns/02_1_CNS_Deployment_Solution_Lab.html#_red_hat_openshift_container_storage_deployment_lab)
* [1. Add GlusterFS Nodes to OpenShift Cluster](https://www.opentlc.com/labs/ocp_storage_cns/02_1_CNS_Deployment_Solution_Lab.html#labexercises)
  + [1.1. Scale Cluster to Add Support Nodes](https://www.opentlc.com/labs/ocp_storage_cns/02_1_CNS_Deployment_Solution_Lab.html#_scale_cluster_to_add_support_nodes)
* [2. Deploy Red Hat OpenShift Container Storage in Existing Lab Environment](https://www.opentlc.com/labs/ocp_storage_cns/02_1_CNS_Deployment_Solution_Lab.html#_deploy_red_hat_openshift_container_storage_in_existing_lab_environment)

Red Hat OpenShift Container Storage Deployment Lab

MitziCom has an existing OpenShift deployment that they want you to reconfigure to take advantage of the advanced storage features of RHOCP.

**Goal**

* Deploy Red Hat OpenShift Container Storage in an existing HA lab environment

1. Add GlusterFS Nodes to OpenShift Cluster

Your Ansible Inventory file (**/etc/ansible/hosts**) contains definitions for three support servers that serve as GlusterFS servers. These servers are commented out by default. This means they are not part of the current OpenShift cluster. So it is necessary to "scale up" the cluster to add these nodes to it.

1. Edit your Ansible hosts file and add a **new\_nodes** child to your **[OSEv3:children]** section. Uncomment the **glusterfs** line.

vi /etc/ansible/hosts

**Sample**

[OSEv3:children]

masters

etcd

nodes

nfs

glusterfs

new\_nodes

* + The Ansible hosts file already contains three node definitions for the support nodes, which are commented out.

1. Add a **[new\_nodes]** heading above the support/CNS nodes and remove the comments in front of the three support nodes.
2. Remove the comments from the **[glusterfs]** section.

**Sample**

[...]

## These are regular nodes

node1.GUID.internal openshift\_hostname=node1.GUID.internal openshift\_node\_labels="{'env':'app', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

node2.GUID.internal openshift\_hostname=node2.GUID.internal openshift\_node\_labels="{'env':'app', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

node3.GUID.internal openshift\_hostname=node3.GUID.internal openshift\_node\_labels="{'env':'app', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

## These are CNS nodes

[new\_nodes]

support1.GUID.internal openshift\_hostname=support1.GUID.internal openshift\_node\_labels="{'env':'glusterfs', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

support2.GUID.internal openshift\_hostname=support2.GUID.internal openshift\_node\_labels="{'env':'glusterfs', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

support3.GUID.internal openshift\_hostname=support3.GUID.internal openshift\_node\_labels="{'env':'glusterfs', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

[nfs]

support1.GUID.internal openshift\_hostname=support1.GUID.internal

[glusterfs]

support1.GUID.internal glusterfs\_devices='[ "/dev/xvdd" ]'

support2.GUID.internal glusterfs\_devices='[ "/dev/xvdd" ]'

support3.GUID.internal glusterfs\_devices='[ "/dev/xvdd" ]'

1.1. Scale Cluster to Add Support Nodes

1. Because the configuration has changed, run the prerequisites check again:

ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

1. Run the **scaleup** playbook to add the support nodes to the cluster:

ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-node/scaleup.yml

1. Verify that the support nodes are now part of the cluster:

oc get nodes --show-labels

**Sample Output**

NAME STATUS ROLES AGE VERSION LABELS

infranode1.GUID.internal Ready <none> 57m v1.9.1+a0ce1bc657 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster=GUID,env=infra,kubernetes.io/hostname=infranode1.GUID.internal,logging-infra-fluentd=true,zone=eu-central-1a

master1.GUID.internal Ready master 57m v1.9.1+a0ce1bc657 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster=GUID,env=master,kubernetes.io/hostname=master1.GUID.internal,logging-infra-fluentd=true,node-role.kubernetes.io/master=true,openshift-infra=apiserver,zone=eu-central-1a

node1.GUID.internal Ready compute 57m v1.9.1+a0ce1bc657 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster=GUID,env=app,kubernetes.io/hostname=node1.GUID.internal,logging-infra-fluentd=true,node-role.kubernetes.io/compute=true,zone=eu-central-1a

support1.GUID.internal Ready compute 40s v1.9.1+a0ce1bc657 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster=GUID,env=glusterfs,kubernetes.io/hostname=support1.GUID.internal,node-role.kubernetes.io/compute=true,zone=eu-central-1a

support2.GUID.internal Ready compute 40s v1.9.1+a0ce1bc657 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster=GUID,env=glusterfs,kubernetes.io/hostname=support2.GUID.internal,node-role.kubernetes.io/compute=true,zone=eu-central-1a

support3.GUID.internal Ready compute 40s v1.9.1+a0ce1bc657 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster=GUID,env=glusterfs,kubernetes.io/hostname=support3.GUID.internal,node-role.kubernetes.io/compute=true,zone=eu-central-1a

1. Review the output and observe that the new support nodes are missing the **logging-infra-fluentd=true** label.
2. Add the missing label to the new nodes to enable log collection on the nodes.
3. oc label node support1.$GUID.internal logging-infra-fluentd=true
4. oc label node support2.$GUID.internal logging-infra-fluentd=true

oc label node support3.$GUID.internal logging-infra-fluentd=true

1. Verify that the **fluentd** pod is now running on all nodes.

oc get pod --all-namespaces -o wide|grep fluentd

**Sample Output**

logging logging-fluentd-8wbx9 1/1 Running 0 51m 10.131.0.3 master1.GUID.internal

logging logging-fluentd-czjrn 1/1 Running 0 51m 10.131.2.2 node1.GUID.internal

logging logging-fluentd-dcwj4 1/1 Running 0 11s 10.129.6.2 support1.GUID.internal

logging logging-fluentd-nbdmh 1/1 Running 0 51m 10.128.0.8 infranode1.GUID.internal

logging logging-fluentd-pldq7 1/1 Running 0 10s 10.131.4.2 support2.GUID.internal

logging logging-fluentd-rtsx7 1/1 Running 0 10s 10.128.6.2 support3.GUID.internal

1. After you have successfully scaled up the cluster, remove the **new\_nodes** line from the **[OSEv3:children]** section and remove the **[new\_nodes]** header above the support nodes.

**Sample**

[OSEv3:children]

masters

etcd

nodes

nfs

[...]

## These are regular nodes

node1.GUID.internal openshift\_hostname=node1.GUID.internal openshift\_node\_labels="{'env':'app', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

node2.GUID.internal openshift\_hostname=node2.GUID.internal openshift\_node\_labels="{'env':'app', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

node3.GUID.internal openshift\_hostname=node3.GUID.internal openshift\_node\_labels="{'env':'app', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

## These are CNS nodes

support1.GUID.internal openshift\_hostname=support1.GUID.internal openshift\_node\_labels="{'env':'glusterfs', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

support2.GUID.internal openshift\_hostname=support2.GUID.internal openshift\_node\_labels="{'env':'glusterfs', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

support3.GUID.internal openshift\_hostname=support3.GUID.internal openshift\_node\_labels="{'env':'glusterfs', 'cluster': 'GUID', 'zone': 'eu-central-1a'}"

[nfs]

support1.GUID.internal openshift\_hostname=support1.GUID.internal

[glusterfs]

support1.GUID.internal glusterfs\_devices='[ "/dev/xvdd" ]'

support2.GUID.internal glusterfs\_devices='[ "/dev/xvdd" ]'

support3.GUID.internal glusterfs\_devices='[ "/dev/xvdd" ]'

2. Deploy Red Hat OpenShift Container Storage in Existing Lab Environment

1. Run the Ansible Playbook to update the cluster to enable basic Gluster operations as Red Hat OpenShift Container Storage:

ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-glusterfs/config.yml

|  |  |
| --- | --- |
|  | In version 3.9.30, there is a known bug that can be fixed by running **sed -i s#rhgs3#registry.access.redhat.com/rhgs3#g /usr/share/ansible/openshift-ansible/roles/openshift\_storage\_glusterfs/defaults/main.yml** |

1. Verify that the Gluster pods are running:

oc get pod -n glusterfs -o wide

**Sample Output**

NAME READY STATUS RESTARTS AGE IP NODE

glusterblock-storage-provisioner-dc-1-tskq7 1/1 Running 0 2m 10.129.0.9 master2.GUID.internal

glusterfs-storage-5qv4k 1/1 Running 0 5m 192.199.0.222 support2.GUID.internal

glusterfs-storage-gt9s6 1/1 Running 0 5m 192.199.0.107 support3.GUID.internal

glusterfs-storage-jxml9 1/1 Running 0 5m 192.199.0.108 support1.GUID.internal

heketi-storage-1-mn5w5 1/1 Running 0 2m 10.131.0.6 master1.GUID.internal

Congratulations! You’ve finished the first lab.

Go on to read the next modules and do the next lab, "Dynamic Provisioning."

Build Version: 1344bb0858f0fd6710870b5d73a7fcf2f8769871 : Last updated 2018-07-31 01:29:20 EDT

Dynamic Provisioning Lab

MitziCom has an existing OpenShift deployment that they want you to reconfigure to take advantage of the advanced storage features of RHOCP.

|  |  |
| --- | --- |
|  | This lab is depenent upon the successful completion of the previous lab, "Red Hat OpenShift Container Storage Deployment" If you have not yet completed that lab successfully, stop now and go complete that lab. |

**Goal**

* Set up storage classes for dynamic provisioning for applications with storage classes in GlusterFS and static provisioning with legacy storage (NFS)

1. Set Up Dynamic Provisioning

In this section, you set up storage classes for dynamic provisioning with GlusterFS and static provisioning with legacy storage (NFS). By default, the GlusterFS installer creates a storage class called **glusterfs-storage**. You make that storage class the default. You assign a different storage class to legacy storage, and learn the special settings you need for the PVC to bind to a non-default PV. Finally, you deploy an application and note how it uses the default storage class, dynamically provisioning PVs and PVCs to assign storage. Then you review the actual storage on the remote volume.

1. On your bastion host, examine the **StorageClass** object that was created by the installer:

oc get sc

**Sample Output**

NAME PROVISIONER AGE

glusterfs-storage kubernetes.io/glusterfs 3h

1. Create a storage class for new applications called **development-storage** with a **volumenameprefix** parameter, and replacing **$GUID** with your GUID in the **resturl** parameter:
2. cat << EOF | oc create -f -
3. apiVersion: storage.k8s.io/v1
4. kind: StorageClass
5. metadata:
6. name: development-storage
7. parameters:
8. resturl: http://heketi-storage-glusterfs.apps.$GUID.example.opentlc.com
9. restuser: admin
10. secretName: heketi-storage-admin-secret
11. secretNamespace: glusterfs
12. volumenameprefix: "development"
13. provisioner: kubernetes.io/glusterfs
14. reclaimPolicy: Delete

EOF

1. Create a storage class called **old-infra** for the old NFS infrastructure and **uservol** storage:
2. cat << EOF > old-infra.yml
3. kind: StorageClass
4. apiVersion: storage.k8s.io/v1
5. metadata:
6. name: old-infra
7. provisioner: no-provisioning
8. parameters:
9. EOF

oc create -f old-infra.yml

1. Examine the storage classes in the cluster:

oc get sc

**Sample Output**

NAME PROVISIONER AGE

development-storage kubernetes.io/glusterfs 31s

glusterfs-storage kubernetes.io/glusterfs 4m

old-infra no-provisioning 22s

1. Replace all of the NFS PVs' storage classes with **old-infra**:

oc get pv -o yaml | sed '/persistentVolumeReclaimPolicy/a \ storageClassName: old-infra' | oc replace -f -

1. Examine the PVs:

oc get pv | head

**Sample Output**

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE

etcd-asb-volume 10G RWO Retain Bound openshift-ansible-service-broker/etcd old-infra 1d

logging-volume 10Gi RWO Retain Bound logging/logging-es-0 old-infra 1d

metrics-volume 10Gi RWO Retain Bound openshift-infra/metrics-cassandra-1 old-infra 1d

prometheus-alertbuffer-volume 10Gi RWO Retain Bound openshift-metrics/prometheus-alertbuffer old-infra 1d

prometheus-alertmanager-volume 10Gi RWO Retain Bound openshift-metrics/prometheus-alertmanager old-infra 1d

prometheus-volume 10Gi RWO Retain Bound openshift-metrics/prometheus old-infra 1d

pv1 5Gi RWO Recycle Available old-infra 9h

pv10 5Gi RWO Recycle Bound 3scale/backend-redis-storage old-infra 9h

pv11 5Gi RWO Recycle Available old-infra 9h

1. Set the development storage class to be the default by changing the value of the **storageclass.kubernetes.io/is-default-class** annotation to **true**:
2. oc patch storageclass development-storage -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'

oc get sc

1. Note that the **development-storage** storage class is now the default:

**Sample Output**

NAME PROVISIONER AGE

development-storage (default) kubernetes.io/glusterfs 6m

glusterfs-storage kubernetes.io/glusterfs 6h

old-infra no-provisioning 2m

1. Deploy a sample application and observe that it uses the default storage class:
2. oc new-project smoke-test
3. oc new-app nodejs-mongo-persistent

oc get pvc

**Sample Output**

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

mongodb Pending development-storage 2s

1. Wait a moment, then examine the PVC:

oc get pvc

**Sample Output**

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

mongodb Bound pvc-ae185317-58ef-11e8-a693-0256a2f4c3f0 1Gi RWO development-storage 26s

* + Note that it is the only one mounted on **development-storage**.

1. Examine the PV:

oc get pv | grep smoke-test

**Sample Output**

pvc-ae185317-58ef-11e8-a693-0256a2f4c3f0 1Gi RWO Delete Bound smoke-test/mongodb development-storage 53s

1.1. Examine Underlying Data

In this section, you connect to one of the Gluster storage pods and examine data.

1. Switch to the **glusterfs** project:

oc project glusterfs

1. Find the **glusterfs-storage** pods:

oc get pods | grep glusterfs-storage

**Sample Output**

glusterfs-storage-5qv4k 1/1 Running 0 12m

glusterfs-storage-gt9s6 1/1 Running 0 12m

glusterfs-storage-jxml9 1/1 Running 0 12m

1. Use the **rsh** command on any of the **glusterfs-storage** pods:

oc rsh glusterfs-storage-5qv4k

1. Examine the volumes:

gluster vol list

**Sample Output**

development\_smoke-test\_mongodb\_ae202150-58ef-11e8-a9f9-02e87850b77a

heketidbstorage

1. Display the information about the MongoDB volume:

gluster vol info development\_smoke-test\_mongodb\_ae202150-58ef-11e8-a9f9-02e87850b77a

**Sample Output**

Volume Name: development\_smoke-test\_mongodb\_ae202150-58ef-11e8-a9f9-02e87850b77a

Type: Replicate

Volume ID: 7103f2fb-3b6d-471f-92da-22be15c49cf3

Status: Started

Snapshot Count: 0

Number of Bricks: 1 x 3 = 3

Transport-type: tcp

Bricks:

Brick1: 192.199.0.108:/var/lib/heketi/mounts/vg\_ac825a9af4faf9597b42e40894ff8004/brick\_d0c5850c286a46615e4b3b7750c3c934/brick

Brick2: 192.199.0.107:/var/lib/heketi/mounts/vg\_5fe802c31a97b0890f6d0dbb6c142472/brick\_e1413e65bffb9a2ad27ebd9dc591ef18/brick

Brick3: 192.199.0.222:/var/lib/heketi/mounts/vg\_9c2e05afb1725d9c9d2db330cbf8abd3/brick\_739cd23b7aed8eca5225f1979c4c598c/brick

Options Reconfigured:

transport.address-family: inet

nfs.disable: on

cluster.brick-multiplex: on

1. Find the pod on which you used the **rsh** command by listing the contents of one of the bricks listed in your output—for example, **Brick3**:

|  |  |
| --- | --- |
|  | You may need to try multiple bricks to find the one on the host where the pod is located. |

1. ls -lah /var/lib/heketi/mounts/vg\_9c2e05afb1725d9c9d2db330cbf8abd3/brick\_739cd23b7aed8eca5225f1979c4c598c/brick
2. **Sample Output**
3. total 272K
4. drwxrwsr-x. 6 root 2000 8.0K May 16 10:43 .
5. drwxr-xr-x. 3 root root 19 May 16 09:58 ..
6. drw---S---. 195 root 2000 8.0K May 16 10:43 .glusterfs
7. drwxr-sr-x. 3 root 2000 25 May 16 09:58 .trashcan
8. -rw-r--r--. 2 1000140000 2000 46 May 16 09:58 WiredTiger
9. -rw-r--r--. 2 1000140000 2000 21 May 16 09:58 WiredTiger.lock
10. -rw-r--r--. 2 1000140000 2000 937 May 16 10:43 WiredTiger.turtle
11. -rw-r--r--. 2 1000140000 2000 40K May 16 10:43 WiredTiger.wt
12. -rw-r--r--. 2 1000140000 2000 4.0K May 16 09:59 WiredTigerLAS.wt
13. -rw-r--r--. 2 1000140000 2000 16K May 16 09:59 \_mdb\_catalog.wt
14. -rw-r--r--. 2 1000140000 2000 32K May 16 10:00 collection-0-5940396012458825912.wt
15. -rw-r--r--. 2 1000140000 2000 16K May 16 09:59 collection-2-5940396012458825912.wt
16. -rw-r--r--. 2 1000140000 2000 16K May 16 09:59 collection-4-5940396012458825912.wt
17. drwxr-sr-x. 2 1000140000 2000 113 May 16 10:43 diagnostic.data
18. -rw-r--r--. 2 1000140000 2000 32K May 16 10:00 index-1-5940396012458825912.wt
19. -rw-r--r--. 2 1000140000 2000 16K May 16 09:59 index-3-5940396012458825912.wt
20. -rw-r--r--. 2 1000140000 2000 16K May 16 09:58 index-5-5940396012458825912.wt
21. -rw-r--r--. 2 1000140000 2000 16K May 16 09:59 index-6-5940396012458825912.wt
22. drwxr-sr-x. 2 1000140000 2000 110 May 16 09:59 journal
23. -rw-r--r--. 2 1000140000 2000 2 May 16 09:59 mongod.lock
24. -rw-r--r--. 2 1000140000 2000 32K May 16 10:01 sizeStorer.wt
25. -rw-r--r--. 2 1000140000 2000 95 May 16 09:58 storage.bson
26. Press **Ctrl+D** or type **exit** to exit the pod.

Congratulations, you have finished the lab. Go on to the next modules and the next lab: "Registry Migration"

Build Version: 1344bb0858f0fd6710870b5d73a7fcf2f8769871 : Last updated 2018-07-31 01:29:20 EDT

Table of Contents

* [Registry Migration Lab](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#_registry_migration_lab)
* [1. Create Gluster Service and Endpoint](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#labexercises)
* [2. Manually Provision GlusterFS Volume](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#_manually_provision_glusterfs_volume)
* [3. Create GlusterFS PV and PVC](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#_create_glusterfs_pv_and_pvc)
* [4. Back Up Existing Container Images in Registry](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#_back_up_existing_container_images_in_registry)
* [5. Modify Volume Definition and Redeploy](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#_modify_volume_definition_and_redeploy)
* [6. Restore Backup and Inspect **smoke-test**](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#_restore_backup_and_inspect_code_smoke_test_code)
* [7. Test **docker push** into Registry](https://www.opentlc.com/labs/ocp_storage_cns/08_1_Registry_Migration_Solution_Lab.html#_test_code_docker_push_code_into_registry)

Registry Migration Lab

MitziCom has an existing OpenShift deployment that they want you to reconfigure to take advantage of the advanced storage features of RHOCP.

|  |  |
| --- | --- |
|  | This lab is dependent upon the previous two labs. If you have not completed them, go ahead now and complete "Red Hat OpenShift Container Storage Deployment" and "Dynamic Provisioning" |

**Goals**

* Migrate a registry from NFS to a GlusterFS storage class for hosted applications

1. Create Gluster Service and Endpoint

In this section, you create the Gluster service and endpoint objects in the **default** namespace. Because of the critical nature of the **docker-registry** integrated container registry, you need to use a statically provisioned volume to ensure that quota, access, and performance are not compromised for the service.

Every project/namespace that accesses GlusterFS as a static persistent volume—as opposed to the dynamic storage class above—must have OpenShift service and endpoint objects created in that namespace for GlusterFS access.

1. Get the IP addresses of the GlusterFS nodes:

dig support{1,2,3}.$GUID.internal|grep internal

**Sample Output**

; <<>> DiG 9.9.4-RedHat-9.9.4-61.el7 <<>> support1.GUID.internal support2.GUID.internal support3.GUID.internal

;support1.GUID.internal. IN A

support1.GUID.internal. 2 IN A 192.199.0.108

;support2.GUID.internal. IN A

support2.GUID.internal. 2 IN A 192.199.0.222

;support3.GUID.internal. IN A

support3.GUID.internal. 2 IN A 192.199.0.107

1. In the **default** namespace, create the Gluster service and endpoints, so they can be consumed by the registry, making sure to edit the **ip** values shown with their actual IP addresses:
2. cat << EOF >> /root/gluster-service-endpoints.yaml
3. apiVersion: v1
4. kind: Service
5. metadata:
6. name: gluster-registry-service
7. spec:
8. ports:
9. - port: 1
10. ---
11. apiVersion: v1
12. kind: Endpoints
13. metadata:
14. name: gluster-registry-endpoints
15. subsets:
16. - addresses:
17. - ip: <support1.$GUID.internal IP>
18. ports:
19. - port: 1
20. - addresses:
21. - ip: <support2.$GUID.internal IP>
22. ports:
23. - port: 1
24. - addresses:
25. - ip: <support3.$GUID.internal IP>
26. ports:
27. - port: 1

EOF

1. Edit the generated **/root/gluster-service-endpoints.yaml** file and replace the IP address placeholders with the IP addresses you previously obtained.
2. Create the Gluster service and endpoint objects from the **default** namespace:
3. oc project default

oc create -f /root/gluster-service-endpoints.yaml

1. Verify that the service and endpoints were created:

oc get svc,ep

**Sample Output**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

svc/docker-registry ClusterIP 172.30.124.135 <none> 5000/TCP 2h

svc/gluster-registry-service ClusterIP 172.30.143.193 <none> 1/TCP 5s

svc/kubernetes ClusterIP 172.30.0.1 <none> 443/TCP,53/UDP,53/TCP 2h

svc/registry-console ClusterIP 172.30.245.64 <none> 9000/TCP 2h

svc/router ClusterIP 172.30.34.199 <none> 80/TCP,443/TCP,1936/TCP 2h

NAME ENDPOINTS AGE

ep/docker-registry 10.128.0.3:5000 2h

ep/gluster-registry-endpoints 192.199.0.107:1,192.199.0.108:1,192.199.0.222:1 5s

ep/kubernetes 192.199.0.126:8053,192.199.0.212:8053,192.199.0.8:8053 + 6 more... 2h

ep/registry-console 10.130.0.2:9090 2h

ep/router 192.199.0.188:1936,192.199.0.221:1936,192.199.0.188:80 + 3 more... 2h

2. Manually Provision GlusterFS Volume

In this section, you use **heketi-cli** to manually provision a GlusterFS storage volume for the container image registry. You need to create this volume for the same **fsGroup** that was used for the NFS registry.

1. Examine the already running **docker-registry** pod to determine the proper **fsGroup** to be used in the GlusterFS volume:
2. REGISTRY\_GID=$(oc get po --selector="docker-registry=default" -o go-template --template='{{printf "%.0f" ((index .items 0).spec.securityContext.fsGroup)}}' -n default)

echo $REGISTRY\_GID

|  |  |
| --- | --- |
|  | Alternatively, you can use **oc get pod <docker-registry podname> -o yaml** and search for the **fsGroup**value. |

1. Find the Heketi pod:
2. oc project glusterfs

oc get pods

**Sample Output**

NAME READY STATUS RESTARTS AGE

glusterblock-storage-provisioner-dc-1-tskq7 1/1 Running 0 1h

glusterfs-storage-5qv4k 1/1 Running 0 1h

glusterfs-storage-gt9s6 1/1 Running 0 1h

glusterfs-storage-jxml9 1/1 Running 0 1h

heketi-storage-1-mn5w5 1/1 Running 0 1h

1. Use **rsh** to access the Heketi pod to use **heketi-cli** and set the environment variables:
2. oc rsh heketi-storage-1-mn5w5 /bin/bash
3. export HEKETI\_CLI\_KEY=$HEKETI\_ADMIN\_KEY

export HEKETI\_CLI\_USER=admin

1. List the existing volumes:

heketi-cli volume list

**Sample Output**

Id:02b988bd8f2cd6a4724b81fc0b67a3ae Cluster:b84ef50165fc6a3dc36a206807b164ec Name:development\_smoke-test\_mongodb\_ae202150-58ef-11e8-a9f9-02e87850b77a

Id:05fb82a9787d686b80aae026f19f8535 Cluster:b84ef50165fc6a3dc36a206807b164ec Name:heketidbstorage

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | |  | The volume you created with your **smoke-test** **nodejs-mongo-persistent** application. | |  | The Heketi API’s database. (To be removed in later versions of the product.) | |

1. Create a new volume for the statically provisioned registry using the **REGISTRY\_GID** you determined above:

heketi-cli volume create --size=20 --gid=<REGISTRY\_GID> --name=gluster-registry-volume

**Sample Output**

Name: gluster-registry-volume

Size: 20

Volume Id: 509f443ee6e8f4184027a0cf80caa80a

Cluster Id: b84ef50165fc6a3dc36a206807b164ec

Mount: 192.199.0.222:gluster-registry-volume

Mount Options: backup-volfile-servers=192.199.0.107,192.199.0.108

Block: false

Free Size: 0

Block Volumes: []

Durability Type: replicate

Distributed+Replica: 3

1. Exit the Heketi pod by pressing **Ctrl+D** or typing **exit**.

3. Create GlusterFS PV and PVC

1. Use SSH to access one of your infra nodes and switch to **root** to test mounting the newly created volume:
2. ssh infranode1.$GUID.internal

sudo -i

1. On the infra node, mount one replica of the registry volume to make sure the group ID matches what was requested:
2. export GUID=`hostname|awk -F. '{print $2}'`
3. echo $GUID
4. mkdir -p /mnt/glusterfs/gluster-registry-volume

mount -t glusterfs support1.$GUID.internal:/gluster-registry-volume /mnt/glusterfs/gluster-registry-volume/

1. Verify the contents of the mounted directory and verify that the correct **fsGroup** is displayed:

ls -lnZ /mnt/glusterfs/

**Sample Output**

drwxrwsr-x. 0 1000000000 system\_u:object\_r:fusefs\_t:s0 gluster-registry-volume

1. Unmount the volume and clean up the directories:
2. umount /mnt/glusterfs/gluster-registry-volume
3. rmdir /mnt/glusterfs/gluster-registry-volume

rmdir /mnt/glusterfs

1. Disconnect from the infra node.
2. Create a PV for the volume by indicating the proper **accessModes** and **persistentVolumeReclaimPolicy** for the integrated registry, and the proper **endpoints** and **path** for GlusterFS.

|  |  |
| --- | --- |
|  | You need a **claimRef** of the **gluster-registry-claim** and its namespace so that the PVC does not bind to a different PV. |

1. oc project default
2. cat << EOF | oc create -f -
3. apiVersion: v1
4. kind: PersistentVolume
5. metadata:
6. name: gluster-registry-volume
7. spec:
8. storageClassName: reg-gluster
9. capacity:
10. storage: 20Gi
11. accessModes:
12. - ReadWriteMany
13. glusterfs:
14. endpoints: gluster-registry-endpoints
15. path: gluster-registry-volume
16. readOnly: false
17. claimRef:
18. name: gluster-registry-claim
19. namespace: default
20. persistentVolumeReclaimPolicy: Retain
21. EOF
22. Verify that the PV was created:

oc get pv gluster-registry-volume

**Sample Output**

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE

gluster-registry-volume 20Gi RWX Retain Available default/gluster-registry-claim reg-gluster 11s

1. Create a PVC for the registry that does not bind to the default **StorageClass**:
   * This PVC’s **name** and **namespace** matches the **name** and **namespace** in **claimRef** in the PV above.
   * You also need to match the PVC to the PV by indicating the proper **volumeName** and **storageClassName**.
   * cat << EOF | oc create -f -
   * apiVersion: v1
   * kind: PersistentVolumeClaim
   * metadata:
   * name: gluster-registry-claim
   * spec:
   * accessModes:
   * - ReadWriteMany
   * volumeName: gluster-registry-volume
   * storageClassName: ""
   * resources:
   * requests:
   * storage: 20Gi

EOF

1. Verify that the **gluster-registry-claim** PVC was created and bound to the correct PV:

oc get pvc

**Sample Output**

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

gluster-registry-claim Bound gluster-registry-volume 20Gi RWX 16s

registry-claim Bound registry-volume 20Gi RWX 2h

* + Observe that it did not attach to *any* storage class.
  + The PVC **registry-claim** remains bound to the legacy (NFS) volume, **registry-volume**.
  + The PVC **registry-claim** does not have a storage class, because it was created before the legacy PVs were configured to have storage classes.

4. Back Up Existing Container Images in Registry

Now you can back up the existing container images from the existing NFS-based registry volume.

1. Use **oc rsync** to copy the contents of the registry file system to a directory on the bastion, replacing the Docker registry pod name with the name of your actual pod:
2. mkdir /root/registry

oc rsync docker-registry-1-9tfnm:/registry/ /root/registry/

5. Modify Volume Definition and Redeploy

In this section, you modify the volume definition in the registry deployment configuration with new storage and then redeploy the registry.

1. Replace the legacy (NFS) volume with the new GlusterFS-backed volume, and allow the registry to redeploy with the new storage:

oc volume dc/docker-registry --add --overwrite --name=registry-storage -m /registry -t pvc --claim-name=gluster-registry-claim

1. Delete the old NFS-backed persistent volume claim:

oc delete pvc registry-claim

1. Use **oc describe dc/docker-registry** to validate that the **registry-storage** volume is now using the **gluster-registry-claim** persistent volume claim.
2. Find the host name of the node where the **docker-registry** pod is running:

oc get pods -o wide

**Sample Output**

NAME READY STATUS RESTARTS AGE IP NODE

docker-registry-2-r5cgl 1/1 Running 0 3m 10.130.2.13 infranode1.GUID.internal

registry-console-1-4thqf 1/1 Running 0 2h 10.130.0.2 node1.GUID.internal

router-1-jjz8n 1/1 Running 0 2h 192.199.0.188 infranode1.GUID.internal

* + Note that **glusterfs** is running on the node to provide the mount into the container.

1. Connect to the node where your **docker-registry** pod is running (**infranode1.GUID.internal** in the example output above):
2. ssh infranode1.$GUID.internal

sudo -i

1. Validate that the GlusterFS volume is mounted on this node:

mount | grep gluster

**Sample Output**

192.199.0.107:gluster-registry-volume on /var/lib/origin/openshift.local.volumes/pods/380a641f-58fa-11e8-a1e8-02e87850b77a/volumes/kubernetes.io~glusterfs/gluster-registry-volume type fuse.glusterfs (rw,relatime,user\_id=0,group\_id=0,default\_permissions,allow\_other,max\_read=131072)

1. Disconnect from the infra node:
2. exit *#the root shell*

exit *#return to bastion*

6. Restore Backup and Inspect **smoke-test**

In this section, you restore the trivial registry backup and inspect the **smoke-test** with **skopeo**.

1. Use **oc rsync** to restore the previous contents of the registry into the new persistent volume, again replacing the name of the **docker-registry** pod with the actual name of the pod in your environment:

oc rsync /root/registry/ docker-registry-2-r5cgl:/registry/

|  |  |
| --- | --- |
|  | Ignore the file permissions change error that results from the command. |

1. Grant **cluster-admin** permissions to the **karla** user if they are not already granted:

oc adm policy add-cluster-role-to-user cluster-admin karla

1. Log in as **karla** and use **skopeo** to validate the changes:
2. oc login -u karla -p r3dh4t1!
3. yum -y install skopeo

skopeo --debug inspect --creds karla:$(oc whoami -t) --tls-verify=false docker://docker-registry-default.apps.$GUID.example.opentlc.com/openshift/dotnet

**Sample Output**

{

"Name": "docker-registry-default.apps.15cd.example.opentlc.com/openshift/dotnet",

"Digest": "sha256:5d600e2c1ee45ed82e08639591a32ee9845295692110a8162591131eff34b4c3",

"RepoTags": [

"latest",

"1.0",

"1.1",

"2.0"

],

"Created": "2018-05-03T23:44:10.471871Z",

"DockerVersion": "1.12.6",

"Labels": {

"architecture": "x86\_64",

"authoritative-source-url": "registry.access.redhat.com",

"build-date": "2018-05-03T23:37:48.223803",

"com.redhat.build-host": "ip-10-29-120-145.ec2.internal",

"com.redhat.component": "rh-dotnet20-container",

"com.redhat.deployments-dir": "/opt/app-root/src",

"com.redhat.dev-mode": "DEV\_MODE:false",

"description": "Platform for building and running .NET Core 2.0 applications",

"distribution-scope": "public",

"io.k8s.description": "Platform for building and running .NET Core 2.0 applications",

"io.k8s.display-name": ".NET Core 2.0",

"io.openshift.expose-services": "8080:http",

"io.openshift.s2i.scripts-url": "image:///usr/libexec/s2i",

"io.openshift.tags": "builder,.net,dotnet,dotnetcore,rh-dotnet20",

"io.s2i.scripts-url": "image:///usr/libexec/s2i",

"name": "dotnet/dotnet-20-rhel7",

"release": "17",

"summary": "Provides the latest release of Red Hat Enterprise Linux 7 in a fully featured and supported base image.",

"url": "https://access.redhat.com/containers/#/registry.access.redhat.com/dotnet/dotnet-20-rhel7/images/2.0-17",

"vcs-ref": "63f4d9a45434667d70e570c0a97a186010149f40",

"vcs-type": "git",

"vendor": "Red Hat, Inc.",

"version": "2.0"

},

"Architecture": "amd64",

"Os": "linux",

"Layers": [

"sha256:d1fe25896eb5cbceebe612f6f820f433997d6748700d05f2029e8e077c02fd16",

"sha256:001d79f6847029de668d289c5b602d004b2bf1691d29f31c643481e0821f03d2",

"sha256:25f1a85ae481757e357d76ceeab27cee37eccc743e225ee53b84628915e796bb",

"sha256:3ccba9f8379d311de9b2890cd58aa84ed0bf5840f3fb6e26abf2b728918ae23c"

]

}

1. Switch back to the **system:admin** user:

oc login -u system:admin

7. Test **docker push** into Registry

1. On **master1**, use the Docker command-line tools to test if the registry is writable:
   * Make sure to do this as a user with **cluster-admin** privileges, like **karla**, and *not* the default **system:admin** user, because that user does not have a token.
   * ssh master1.$GUID.internal
   * sudo -i
   * oc login -u karla -p r3dh4t1!
   * docker login -u openshift -p $(oc whoami -t) docker-registry.default.svc:5000
   * docker pull docker.io/busybox
   * docker tag docker.io/busybox docker-registry.default.svc:5000/openshift/busybox
   * docker push docker-registry.default.svc:5000/openshift/busybox
   * oc login -u system:admin *# Switch back to system:admin before logging out*
   * exit *# from root shell*

exit *# to bastion*

|  |  |
| --- | --- |
|  | You need to do this from a node within the OpenShift cluster in order to be able to access **docker-registry.default.svc**. The **bastion** host is not a member of the OpenShift cluster. |

Congratulations, you’ve finished all the labs!

Build Version: 1344bb0858f0fd6710870b5d73a7fcf2f8769871 : Last updated 2018-07-31 01:29:20 EDT