Enabling High Availability Service with oVirt Virtualization and CephFS

M D Poat[1], J Lauret[1]
[1] Brookhaven National Laboratory, P.O Box 5000, Upton, New York 11973-5000, USA

Abstract. The online computing environment at STAR has generated demand for high availability of services (HAS) and a resilient uptime guarantee. Such services include databases, webservers, and storage systems that user and sub-systems tend to rely on for their critical workflows. Standard deployment of services on bare metal creates a problem if the fundamental hardware fails or loses connectivity. Additionally, the configuration of redundant failover nodes (a secondary Web service for example) requires constant syncing of configuration and content and sometimes manual interaction for switching hardware (DNS name comes to mind). For this project, we will focus on two tools: oVirt and Ceph. oVirt is an OpenSource virtualization management application that enables central management of hardware nodes, storage, and network resources used to deploy and monitor virtual machines. oVirt supports the deployment of a virtual environment for your data center leveraging automatic provisioning, live migration, and the ability to easily scale the number of hypervisors. oVirt enables the use of multiple storage technologies where you can store virtual machines, images, and templates within one or multiple storage systems. STAR's recent efforts focused on deploying a CephFS POSIX compliant distributed storage system, we would enable the ability to couple our Ceph storage with the oVirt virtualization management system. When designing an intricate system for hosting critical services, it is a requirement to circumvent single points of failure. This work will involve the testing and viability of such an approach along with test cases for high availability, live migration, and service on demand.

1. Introduction

The Relativistic Heavy Ion Collider (RHIC) is the only heavy ion collider in the USA. The Solenoidal Tracker at RHIC (STAR) experiment is the main running detector in operations since year 2000. During the RHIC running period, STAR operations have generated strong demand for high availability of services (HAS), guaranteed uptime, and overall strong reliability of our compute infrastructure. Currently many of our critical services such as our Webserver (serving all monitoring for the STAR experiment), Identity Provider (IDP) used for Single Sign On, and our database servers are all run on bare metal nodes with sometimes, a failover node providing a backup service. This setup creates a problem if the fundamental hardware does go down as the failover mechanisms typically require manual intervention and downtime usually occurs. Virtualization is a well-known established technology that enables portability to any working enclave. By moving towards a virtualized environment, STAR would gain the ability to easily take back-ups, clones, snapshots, and templates of our production virtualized systems. Virtual machines can be deployed on thin provisioned virtual disks and can be stored in a common shared storage such as CephFS.

2. CephFS

Ceph is a distributed storage system based on RADOS (Reliable Autonomic Distributed Object Store) [1]. Within STAR, we have deployed a 30 node, 240 TB raw storage CephFS cluster offering our users 80 TB of redundant safe storage (replication 3). In 2015, we published our first Ceph related research which was based on OpenStack Swift Object Storage and Ceph Object storage studying the architecture and IO performance of the two, the research was then followed by the deployment and study of CephFS which is the POSIX compliant layer on top of Ceph Object Storage, this work was presented at the CHEP 2015 conference [2]. The work shown at CHEP 2015 was then followed with a contribution for ACAT 2016 [3] where we studied the individual data placement techniques built into Ceph such as Primary Affinity, Journals on SSDs, and Cache Tiering. With this work we found that each technique had its own draw back and that Ceph would only perform as fast as its slowest component. Lastly, at the most recent CHEP conference (CHEP 2016) we published an additional research where we deployed SSD's in Ceph at a lower level in the software stack using disk caching techniques such as bcache and dm-cache [4]. We found that the performance of the SSDs laid on top of HDDs to act as a cache would outperform the bare HDDs when tested in a standalone environment. However, SSDs over HDDs tested as OSDs in Ceph the performance of the bcache/dm-cache devices performed below the plain HDD OSDs. We found that, in Ceph, drives that have Power Loss Protection (PLP) support will perform better in Ceph that drives without PLP support. The SSD drives that we used did not have PLP support but the HDDs did have PLP support. In the end, we concluded that the best configuration for our Ceph cluster was to use our 2 TB PLP supported HDDs as we gain maximum storage capacity with very fast write speed to Ceph.

3. oVirt

oVirt is a virtualization management application based on Red Hat Enterprise Virtualization (RHEV) [5]. oVirt has high availability features and a scalable architecture that may enable STAR to fulfil our niche requirements for our online enclave. Additionally, oVirt has support for multiple storage technologies such as our famed CephFS along with GlusterFS, NFS, iSCSI, and more. The oVirt framework consists of two main components, the oVirt engine and oVirt hypervisors. The oVirt engine provides the management interface and is responsible for managing hardware nodes, network resources, and is responsible for deploying and monitoring virtual machines that run on the hypervisor nodes. The hypervisor nodes are essentially "worker nodes" as they follow suite of requests from the oVirt engine.

Our initial plan for oVirt is to configure a resilient cluster with no single point of failure. In order to achieve this, we would reuse some of our over provisioned nodes to deploy our highly available virtual machines. We would enable the proper power management requirements on our nodes along with UPS backup power (IPMI or iDRAC) [6]. Our CephFS distributed storage system would be used as our main storage backend to store all virtual machines and content. Lastly, we would be deploying the self-hosted engine which will be discussed later in this paper.

4. Standard oVirt Deployment

A standard oVirt deployment typically requires 3 nodes at minimum in order to ensure some level of high availability. First you choose one node as the oVirt engine; you then install a required set of packages from the oVirt repository [5], followed by running the engine-setup initialization tool that is required for preparing the node. Once your oVirt engine is all setup you can then deploy your hypervisor nodes (in our case we call them ovirt1.star and ovirt2.star). The oVirt engine web interface enables you to deploy your hypervisor nodes right from it, this feature is intuitive especially if you have a large set of nodes to deploy. Once all of the physical nodes are setup, we can then add CephFS in as the main Data and Image storage domains. Lastly, we can then create virtual machines that will be managed by the oVirt engine and can run on either ovirt1.star or ovirt2.star.

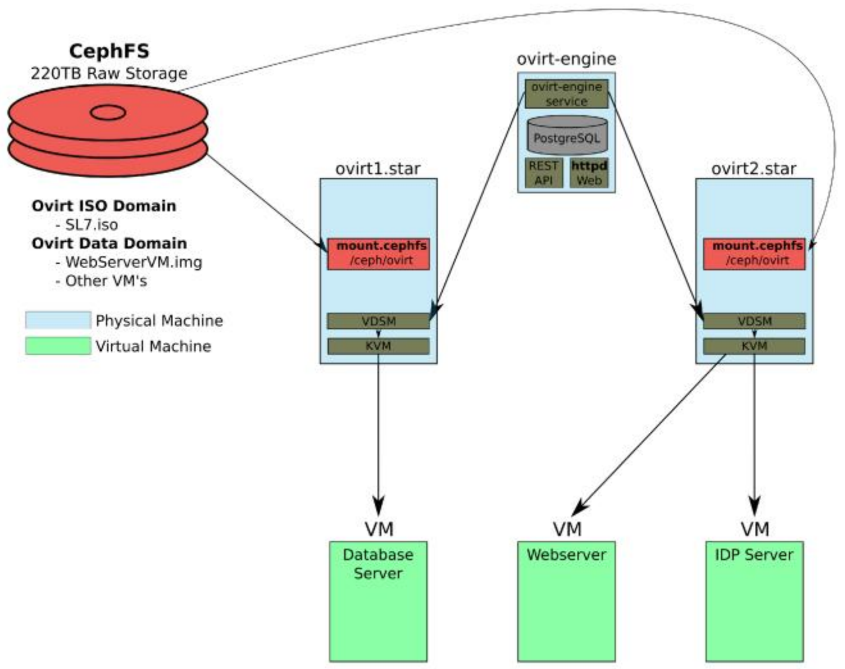

Figure 1. The diagram above is showing a Standard oVirt Deployment that is widely used within the oVirt community. The oVirt-engine is shown as the master node and the ovirt-engine service is shown managing the Virtual Desktop and Server Manager (VDSM) on the hypervisor nodes. CephFS is the main storage domain which is required to be mounted on all hypervisor nodes.

As shown in Figure 1, a Standard oVirt Deployment has been visually described and it can be seen how each service is implemented into the oVirt cluster. Within this deployment, if one of the hypervisor nodes (ovirt1.star or ovirt2.star) were to suffer a downtime, the running virtual machines on the failed hypervisor would failover to the other running hypervisor. The virtual machine may suffer a 1-2 minutes downtime, but it would soon recover. While a Standard oVirt Deployment may seem intuitive at first glance, there is a flaw in the setup. If, for example, the oVirt-engine machines were to suffer a failure and go down, the running virtual machines would become headless and there would be no control (i.e. no failover, no migration, no monitoring, etc.) over the hosts or virtual machines. If the oVirt-engine machine suffered a full catastrophic failure, recovering the oVirt environment would require downtime of the entire system and would not be pleasant.

5. Self-Hosted Engine Deployment

5.1 Deployment

The oVirt engine is the control center of the oVirt environment. The service talks directly to VDSM on the hosts to deploy, start, stop, migrate and monitor VM's [7]. Considering how imperative the oVirt engine is, it seems counterintuitive to run this service on a bare metal node as well. In oVirt, there is a feature called the Self-Hosted Engine. A self-hosted engine is a virtualized environment in which the oVirt Engine software runs on a virtual machine on the hosts managed by that engine [8]. Initially this may sound convoluted, but it works quite well. Essentially you have a physical host, that runs a virtual machine, the oVirt-engine software is installed on the virtual machine, the virtualized oVirt-engine then comes back to manage the initial host to run more highly available virtual machines. While the Self-Hosted engine may seem overly complex and unnecessary, the result in the end is worth it. The Self-Hosted engine needs at least two physical nodes that will both run software instances called 'ovirt-ha-agent' and 'ovirt-ha-broker'. The two nodes will negotiate using the ovirt-ha-agent to determine which host should run the self-hosted engine VM based on an internal scoring system, the host with the highest score will run the VM. The ovirt-ha-broker is responsible for monitoring the resources on the physical hosts themselves (CPU, load, memory, etc.).

5.2 Gluster

At the time of writing this paper the latest recommended version oVirt is version 4.1. oVirt 4.1 supports the Self-Hosted Engine feature, however there is no support for CephFS to act as the storage domain to store the Self-Hosted Engine virtual machine. When running through the deployment of the Self-Hosted Engine, the supported storage domains are GlusterFS, ISCSI, FC, NFSv3 or NFSv4. While CephFS can still be used as the storage domain for all other virtual machines, for our Self-Hosted oVirt deployment the next best thing would be a replicated GlusterFS storage system. GlusterFS is a scalable network filesystem suitable for data-intensive tasks such as cloud storage and media streaming [9]. GlusterFS is free open source software that can run on commodity hardware. Considering high availability as our main goal for our oVirt deployment, we decided to use our 3 physical hosts that will run our oVirt environment to also run a replication 3 GlusterFS storage system. We added 3 disks per machine, creating a 9 disks GlusterFS with replication 3. One out of three disks per machine is set as an arbiter volume; arbiter volumes only store file/directory names and metadata and do not store any actual real data. Enabling an arbiter volume will prevent a common issue in Gluster called "split-brains" [10]; a file is in split-brain when there is an inconsistency in either data or metadata and Gluster does not have enough information to ensure the integrity of the file (i.e. latest copy, correct metadata, etc.) . Split-brains can only really occur when there is a failure event or a simple reboot of one of the storage nodes without any arbiter volumes [11]. With proper configuration, we enabled our GlusterFS to seamlessly suffer a single node downtime with zero interruption of access to the file system. If we suffer a two nodes failure or downtime, the GlusterFS becomes read-only. After recovery of any downed hosts, GlusterFS will repair itself immediately to full replication 3.

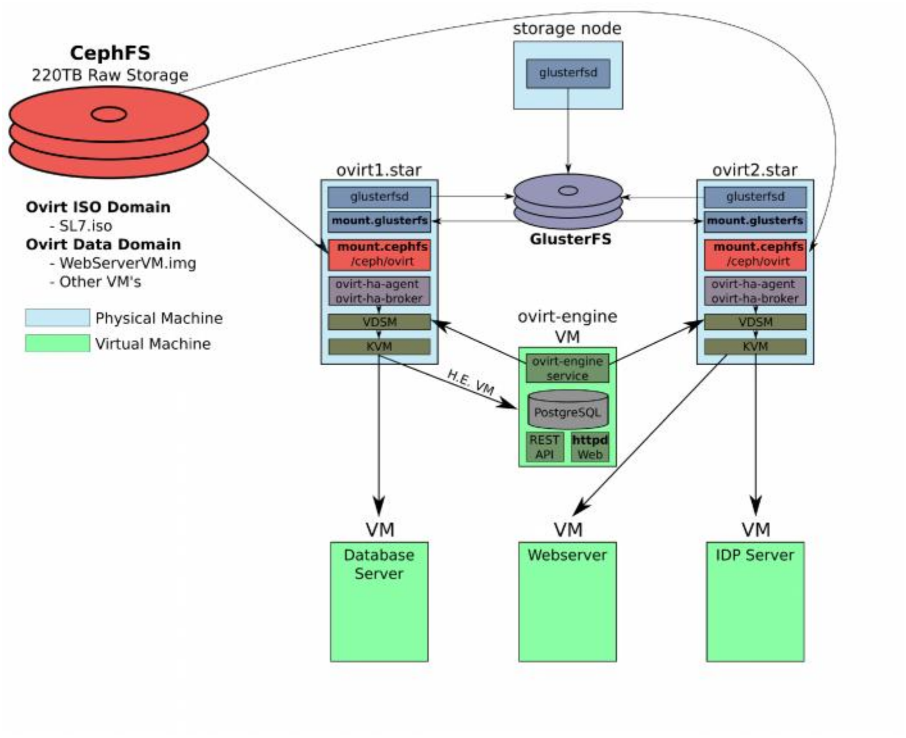

Figure 2. The diagram above is showing a Self-Hosted Engine oVirt Deployment where the oVirt engine is a highly available virtual machine itself. The GlusterFS filesystem is shown setup on ovirt1.star, ovirt2.star and a 3rd host titled storage node. The oVirt engine VM is stored in GlusterFS, the remaining VM's are all stored in CephFS.

As seen in Figure 2, we are showing a diagram of a Self-Hosted oVirt deployment. This setup first entails creating our 3 hosts replication 3 GlusterFS filesystem, followed by setting up our hypervisor nodes as GlusterFS clients. Once your initial storage is setup, we can move to the Self-Hosted engine deployment. First, we install ovirt-hosted-engine-setup and ovirt-engine-appliance packages from the oVirt repositories on our hypervisor nodes, once installed we then follow the simple deployment steps from running 'hosted-engine --deploy' which will take you through the setup to create the virtual machine and oVirt engine. The deployment setup also ensures the ovirt-ha-agent and ovirt-ha-broker packages are properly installed. Once the Self-Hosted oVirt engine is setup, the next steps are the same as any standard oVirt deployment. We simply deploy additional packages and configuration to any additional hypervisors nodes via the oVirt engine web interface, we re-add CephFS as our main storage domain, and from here we can create and deploy our highly available virtual machines for STAR production use case.

The Self-Hosted oVirt engine is a much more resilient setup compared to a standard oVirt engine deployment on bare metal. The reasons for this statement are as follows. When referring back to Figure 2, if we were to suffer a failure from our 'Storage Node' (which acts as one of the three GlusterFS hosts), then GlusterFS would continue working with replication 2 between ovirt1.star and ovirt2.star. In a different scenario, if we were to suffer a failure from ovirt2.star (which is running the two VM's titled Webserver and IDP server), then the oVirt-engine would immediately recognize the failed VM's and host, oVirt-engine would ensure ovirt2.star is actually down followed by restarting the downed VM's on ovirt1.star as a fail-over. Finally, in our last scenario if we were to suffer a failure from ovirt1.star (which is running the oVirt-engine VM itself and our Database VM), the ovirt-ha-agent and ovirt-ha-broker services running on ovirt2.star would recognize the oVirt-engine VM is down and would restart the VM on ovirt2.star. Once the oVirt-engine VM and services have recovered, any other downed VM's (Database VM) would then be restarted on ovirt2.star as well. With multiple passes of live testing of this configuration and successful fail-over mechanisms, it has been proven that a Self-Hosted oVirt engine setup truly is a highly available system.

6. Live Migration

Live migration is a feature of the KVM hypervisor that enables you to transfer live running virtual machines from one host to another with practically 0 down time [12]. With live migration, the virtual machine remains powered on and all user applications continue to run after the relocation/migration to the new host. The virtual machine RAM is actually copied from the source host to the destination host. In oVirt, live migration is a critical feature as it enables you to migrate production VM's to from one hypervisor to another hypervisor with no interruption. If a host needs to be taken out for maintenance or repair, it can be done without interrupting users or services.

To test live migration and to ensure that all acclaimed features work properly, we created a small test case demonstrating continuous functionality of a Web service. The test was to create a multi-stage CGI (a first stage passes to a second stage hidden parameters), making sure that between the two stages we a live migration and verify the information passed is preserved. In the CGI we made an entry in the first page, then migrated the virtual machine from one host to another, once migrated we would make the final entry and check the results. The result was that the entries from both the first page and second page are carried over properly. This test shows that virtual machine's RAM is properly copied during a live migration.

7. High Availability

High Availability is a feature in oVirt to ensure your virtual machines are always up [13]. To properly enable high availability there are few requirements that need to be met. First, your physical hypervisor nodes must have a power fencing interface such as an IPMI interface, iDRAC or the like [6]. An IMPI/iDRAC interface enables access to the baseboard management controller to manage the hosts low level controls such as the BIOS, hardware controls, and most importantly power management. The oVirt-engine requires access to this IPMI/iDRAC interface in order to power fence the host if needed, typically in cases of the host not responding or if the host power status needs to be verified. Furthermore, to enable high availability you must have equal available resources on your destination host for failover events. For example, if your highly available virtual machine has 4 CPU cores and 8 GB of memory, then there must be 4 CPU cores and 8 GB memory free on any destination host if you want to the virtual machine to successfully failover. Lastly, your hypervisors must be running the same CPU architecture (x86_64, etc.) and CPU family (Haswell, Sandy Bridge, etc.).

In STAR's oVirt environment we tested high availability in a live running oVirt cluster. The test was to simply pull the power chord for ~5 seconds on a hypervisor node running a highly available virtual machine to see how oVirt would react. Immediately after the power chord was removed, the oVirt-engine recognizes the downed VM and tries to restart it on another hypervisor. Additionally, the oVirt-engine checks the status of the downed hypervisor hosts via the IPMI/iDRAC interface, if the status of the machine returns that power is available but the host is not running, oVirt-engine will power fence the host to restart it. Meanwhile, while the hypervisor host is being power fenced, the virtual machine should be back up and running within a minute from the initial power chord removal. Lastly, within 3-5 minutes, the downed hypervisor host should be back up and running and re-added to oVirt as an available hypervisor host.

8. Conclusion

oVirt virtualization, a community free project branching from RHEV, will enable STAR to create a true highly available system helping to support our critical services. When deploying oVirt with the self-hosted engine feature, all single points of failure are eliminated. While the self-hosted engine does not support CephFS as its storage domain at this time, we have looked into this lack of support and have opened up a feature request with Red Hat to add CephFS as a self-hosted engine storage domain. However, apart from the self-hosted engine, all other virtual machines and content consequently can be stored in CephFS. For high availability to be enabled on our virtual machines in oVirt, there must be available resources on standby (2x CPU & Memory) in order for the VM to properly failover. This is a fair trade-off to ensure our critical nodes and services are always up. Virtualization will enable STAR to better use our over provisioned nodes to deploy a diverse set of virtual machines to meet our operation needs.

Acknowledgements
This work was supported by the Office of Nuclear Physics within the U.S. Department of Energy's Office of Science.

References

[1] Ceph http://www.ceph.com
[2] Poat M D, Lauret J and Betts W 2015 *POSIX and Object Distributed Storage systems – Performance Comparison Studies With Real-Life Scenarios in an Experimental Data Taking Context Leveraging OpenStack Swift & Ceph* J. Phys.: Conf. Ser. 664 042031
[3] Poat M D and Lauret J 2016 *Performance and Advanced Data Placement Techniques with Ceph's Distributed Storage System* J. Phys.: Conf. Ser. 762 012025
[4] CHEP 2016
[5] oVirt https://www.ovirt.org
[6] Integrated Dell Remote Access Controller 8 (iDRAC8) http://www.dell.com/support/manuals/us/en/19/idrac8-with-lc-v2.05.05.05/idrac8_2.05.05.05_ug-v1
[7] oVirt Engine Architecture https://www.ovirt.org/documentation/architecture/architecture/#engine
[8] oVirt Self-hosted engine deployment https://www.ovirt.org/documentation/self-hosted/chap-Introduction/
[9] Gluster https://www.gluster.org/
[10] Split Brain https://gluster.readthedocs.io/en/latest/Administrator%20Guide/Split%20brain%20and%20ways%20to%20deal%20with%20it/#split-brain
[11] Arbiter volumes and quorum options http://docs.gluster.org/en/latest/Administrator%20Guide/arbiter-volumes-and-quorum/
[12] Live Migration https://www.ovirt.org/documentation/vmm-guide/chap-Administrative_Tasks/#migrating-virtual-machines-between-hosts
[13] High Availability https://www.ovirt.org/develop/release-management/features/sla/ha-vm-reservation/