Praxisarbeit - Clustering with Linux in the year 2015

Noel Kuntze - 175280

September 11, 2015

Abstract

As Linux is the most widely spread operating system in the data center environment, implementing high availability for services on that platform is an important aspect to critical network services. This paper lists and asesses the modern implementations of software for high availability on the Linux platform.

List of Figures

List of Abbreviations

API Application Programming Interface

BMC Base Management Controller

CIB Cluster Information Base

CIFS Common Internet File System

CRMd Cluster Resource Management daemon

DRBD Dynamic Redundant Block Devices

ERP Enterprise Resource Planning

GFS Global File System

GFS2 Global File System 2

HA High Availability

HMAC Hashed Message Authentication Code

KDC Key Distribution Center

LRMd Local Resource Management daemon

MITM Man In The Middle

NAS Network Access Storage

NFS Network File System

NLM Network Lock Manager

NONCE Number used ONly onCE

NSS Network Security Services

OCFS Oracle Cluster File System

OCFS2 Oracle Cluster File System 2

pcs Pacemaker/Corosync configuration system

PE Policy Engine

RPC Remote Procedure Call

SAN Storage Area Network

SHA Secure Hash Algorithm

SMB Server Message Block

SPOF Single Point Of Failure

STONITHd Fencing daemon

STONITH Shoot The Other Node In The Head

TCP Transmission Control Protocol

TTL Time To Live

UUID Universally Unique IDentifier

VLAN Virtual Local Area Network

VM Virtual Machine

Contents

1 Introduction

High availability is an important aspect of network services in the corporate sector, as it decreases the time a service is unavailable to users. Such services might be business critical, like ERP[1] systems. Therefore it is necessary to evaluate and implement high availability in those systems.

1.1 The scope of this document

The purpose of this document is to explain and promote the modern implementations of high availability clustering on Linux, as well as judge the current state of them and availability on different Linux distributions. For this purpose, different tools for this purpose will be explained in depth with their features and services to the cluster environment. Furthermore, this document contains guidance on optimizing cluster performance, failure detection and guidance on configuring a high availability cluster scenario on three CentOS 7 hosts in a virtualized environment.

[1] Enterprise Resource Planning

2 High Availability

High Availability is usually implemented by duplicating the service over several distinct physical or virtual hosts. The hosts are called a ̈cluster:̈ The software on the hosts and other resources that are duplicated and necessary for the service are managed by what is called a ̈cluster manager:̈ Said manager handles cluster communication and failover action over several hosts if a local service, that is protected by High Availability (HA) goes down.

3 Cluster Types

Naturally, there exist different types of clusters for different purposes.

- High Availability Cluster to increase the availability of services.
- High Performance Clusters to increase the performance of services.
- Load Balancing Cluster to balance the load of a service.

In this work, we will only look at High Availability Clusters. High Availability Clusters are usually used for critical infrastructure, which must be available most of the time. Examples for such a case are DNS servers, routers and firewalls, as well as Enterprise Resource Planning (ERP) systems.

4 Data Storage

The hosts in the cluster usually provide a service for which persistent shared data storage is necessary. If those files are static or are rarely modified, locally storing them on each node can be a solution. However, caution is advised, as makeshifts have a bad habit of sticking around. There are two different feasible ways to implement this in a cluster environment:

- Using a storage area network
- Using a network access storage
- using shared storage on the cluster nodes

Using a storage area network in a cluster environment is good choice as it provides highly available access, if the Storage Area Network (SAN) is layed out for high availability. That is the case in enterprise environments. As such, it can be relied upon. Using network access storage is a more common solution for providing access to data in an environment where concurrent access is needed, but cluster file systems are not an option. In such a case, the cluster members would access the data over network protocols like NFS or CIFS. For concurrent access to shared storage in a cluster, file systems like Global File System (GFS), Oracle Cluster File System (OCFS) or CephFS have to be used, as those provide a lock manager, which is needed for data integrity. In the following sections, we will go over some cluster file systems and shared storage providers, as well as Ceph in an extra section.

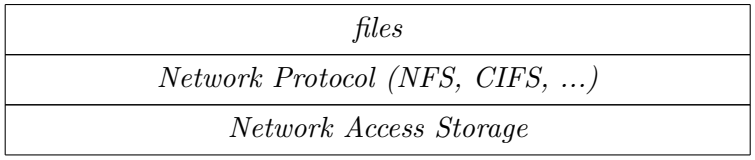
4.1 Storage Area Network

files
Cluster File System
Block Device on the cluster member
Network Protocol (iSCSI, ...)
Storage Area Network

Storage Area Networks are a common concept in enterprises and are mostly implemented with appliances from different vendors, which cluster and provide high availability. The nodes in need of shared storage import the shared storage with iSCSI or

other network protocols that the own host's kernel can display as block devices. Over that block device, a cluster file system must be layed over the shared medium to handle concurrent access. Obviously, Single Points Of Failure (SPOFs) in the connection to the SAN must be avoided, for which different solutions can be used.

4.2 Network Access Storage



Network Access Storage (NAS) are basically network hard drives, which provide file storage to other hosts on the network. They do not provide distributed storage like a SAN and are usually not layed out redundantly. NAS export their data using network file systems like NFS or CIFS, which provide their own lock manager. This solves concurrency issues. Some NAS are also capable of using other network file systems like iSCSI to provide access to data on the block layer, which enables the use of cluster file systems on top of those block devices.

4.2.1 NFS

NFS is a distributed file system protocol, which is deployed widely on Linux and UNIX. The latest version is 4.1 and is openly standardized in RFCs, so anyone can implement it.

protocols The NFS itself only specifies the Remote Procedure Call (RPC) calls, not the transport protocol. NFS from version 1 to 3 only works over **UDP!** (**UDP!**), but version 4 and onwards also support Transmission Control Protocol (TCP).

performance NFS can be faster than CIFS, if implemented correctly.

security NFS has several different security mechanisms

lock manager NFS from version 1 to 3 has no intrinsic lock manager in the protocol. However, there is the Network Lock Manager (NLM) protocol, which can work together with the NFS protocol to provide a lock manager. It is a separate service of NFS.

Version 4 and later of NFS provide a native lock manager, which enables concurrent access to the files. NFS currently does not provide a distributed lock manager, therefore a breakdown of communication between the NFS client and the NFS server will result in stale locks on the server until the connection times out. Also, there is no possibility for the deployment of redundant servers. As such, the loss of a single server prevents access to the shared resources. This makes it grossly unsuitable for usage in an environment where high availability is needed.

4.2.2 CIFS

CIFS or Server Message Block (SMB) is a protocol developed by Microsoft, but originally made by IBM, for use in the Windows operating systems to access files on remote hosts over an IP based network. The protocol is readily available on Linux as part of the kernel and can be used to access remote file systems, which are either provided by a native Windows host or the smbd and nmbd services for Linux, which are provided by the SAMBA project.

protocols CIFS can use TCP or NetBIOS as underlying transport protocols. Nowadays, TCP is used over port 445.

performance CIFS isn't exactly famous for its performance. Version 1 has severe performance problems, which arise from the use of NetBIOS as transport protocol and the chattiness of the protocol. Newer versions, starting with 2.0, have increased performance through the use of TCP window scaling and decreased chattiness, as well as client side caches and a redesign of the protocol.

security CIFS is capable of providing native authenticity through signing of the packets and confidentiality through encryption of the packets. The key exchange is done through Kerberos, for which a Key Distribution Center (KDC) is needed.

lock manager CIFS has a native lock manager, which runs on the CIFS server. This prevents concurrent access to single files, but can not coordinate concurrent and fast access to files, as the information about the held locks is not distri- buted to the clients.

4.3 Shared Storage On The Cluster Nodes

files
Cluster File System
Shared Storage Implementation of your choice
Block Device on the cluster member
Specific Implementation of Shared Storage

Shared storage on the cluster nodes is a similiar topic to storage are network. The difference is, that the block devices are stored locally on each cluster node and not in a Storage Area Network. In such a scenario, data replication has to be taken care of by the system's kernel or another application. A solution for this on Linux is Dynamic Redundant Block Devices (DRBD), which replicates data over the network using TCP.

DRBD stands for Dynamic Redundant Block Devices and is an open source technology developed by LINBIT. It provides data replication over a cluster of two hosts. As of the time of the writing, DRBD only works good for two nodes, but scales bad beyond it and is not suitable for HA clustering in such scenarios. In the next version of DRBD, support vor arbitrary numbers of cluster members will be added, which enables cheap shared storage for multiple cluster members. The current version of DRBD implements replication between more than two nodes using stacked resources, which explicitly define the replication between two nodes. If the node, that is part of the normal and the stacked resource dies, there is no replication between the remaining two nodes.

files
Cluster File System
Block Device on the cluster member
DRBD block device on the cluster member
DRBD metadata and data layer
Hard drive partition
Kernel
...

4.3.1 Cluster File Systems

GFS2 Global File System 2 (GFS2) is the second iteration of a cluster file system of Red Hat, Inc. It provides different lock managers:

distributed lock manager

nolock manager

GlusterFS

OCFS2 Oracle Cluster File System 2 (OCFS2) is a cluster file system developed by Oracle. Its original purpose was to serve as cluster file system for Oracle's database software. It provides POSIX semantics and is part of the Linux kernel since kernel version 2.6.16.

Its current version is 1.6. Support for it is very limited, because Oracle only provides any for it as part of Oracle Linux. SUSE provides support, too, but only if it is used on Novell's SUSE Linux Enterprise Server. The tool chain to manage OCFS2 file systems is available on Oracle's web site[2], however the tool chain needs patches to compile correctly. Therefore, it is advised to look at the rpm source of the working packages for CentOS or other distributions and use those as a basis to get a working tool chain.

cLVM2 cLVM2 is the clustered version of LVM2, which is a volume manager for Linux. The clustered version enables the synchronization of the status and metadata of the cLVM volumes over the network.

Performance Considerations Distributed locking is a lot worse performance wise than purely local locking, because the process of locking inodes includes more operations and even the network layer. This causes big performance problems when data is written to the file system, therefore cluster file systems should only be used when changes need to be propagated immediately and only few I/O operations are required.

Usually, for High Availability clusters, it is sufficient for a cluster file system to have at least some performance. In numbers, this means the processing of hundreds of transactions, not thousands or more. A trade off of performance for availability is usually

[2] `https://oss.oracle.com/projects/ocfs2-tools/`

wanted, as a highly performing cluster is useless, if it easily breaks. Judging from existing performance benchmarks, OCFS2 performs better than GFS2 and better than NFS, however, OCFS2 has not been developed any further publicly since 2012 and version 1.4. Oracle Linux 5 offers version 1.6 of OCFS2, but is a commercial product of Oracle and only available together with Oracle's Unbreakable Enterprise Kernel, which is in Oracle Linux 5, which prevents the widespread usage of this solution. The common public image of Oracle further discourages any use of the file system.

High Availability Considerations For high availability, it is necessary to have a distributed lock manager to prevent a loss of availability if the host which currently has the lock manager, goes down or is unreachable for any reason. A distributed lock manager avoids this by existing distributed on all hosts, possible with identical copies, which survive a

4.3.2 Shared Storage Providers

4.3.3 Ceph

Ceph has a special position in the cluster world, as it provides all aspects of shared storage:

- block devices
- cluster file systems
- object storage

Ceph is an open source cluster storage platform to provide block, file and object storage to other hosts. It is constructed in a manner that prevents SPOFs, scales well to exabyte levels and provides redundancy to prevent data loss. It is capable of distributing the data across the cluster members and maintaining performance over an unlimited number of cluster members.

Ceph can be used as shared storage provider, over which an arbitrary cluster file system can be layed, but also completely with CephFS as complete shared storage solution. Furthermore, it can be used as storage backend for libvirt and qemu to store virtual machines in. It also provides a gateway for object storage over a REST API and is compliant with the APIs provided by Amazon S3 and Swift.[unk]

Users authenticate themselves to the cluster using a mechanism that utilises shared keys and provides authenticity to prevent Man In The Middle (MITM) attacks. However, no secrecy or replay protection is provided[unk][Car06].

Ceph is capable of distributing copies of stored objects over the cluster, as well as striping and dynamically moving the objects to balance the load on the different hosts in the storage cluster. A such, it is a perfect solution for large storage clusters.

5 Running clusters on VMs

5.1 Why it is a bad idea

Running a cluster on VMs is neither encouraged, nor widely supported. The reason herein is, that the added complexity of the hypervisor decrease the availability of the service. Because of the nature of the token protocol used by corosync, Corosync is scheduled as a realtime process. This is necessary to ensure the quick processing of the tocken messages. If that is not ensured, other nodes in the cluster could believe the node is dead and will fence it to ensure the safety and consistency of the shared storage. As the processing capability and network is shared with other VMs, overprovisioning of the host **will** cause problems with the cluster. The cluster nodes must be guaranteed to be able to run corosync in time. This will cause problems, because the VM itself isn't scheduled for real time, as is required by corosync. In rare cases, the timeout values for the token protocol have to be increased, because the hypervisor does not schedule the guest for execution early enough. This will case corosync to log warnings over syslog:

```
1 Aug 04 02:10:10 c7-arch-mirror-1 corosync[1058]: [MAIN ] Corosync main
     process was not scheduled for 1217.9677 ms (threshold is 800.0000 ms)
     . Consider token timeout increase.
```

6 Fencing/STONITH

6.1 Purpose

Fencing and STONITH basicly describe the same thing: The prevention of I/O-Operations of a disconnected/misbehaving host in the cluster. For this purpose, two different

key words have been established: Fencing and Shoot The Other Node In The Head (STONITH).

6.2 Usage

STONITH or Fencing is an optional feature of a cluster, but needed to prevent unwanted I/O operations on a shared medium or a database by an unreachable cluster member in a split-brain situation. Usually, this is needed and desired in a cluster environment. STONITH or Fencing can be implemented through network capable power switches, which can cut the power to the split off hosts or if VMs are used, by telling the hypervisor to kill the VM. Ususally, a reboot of the node is sufficient to fix the problem.

6.3 Fencing classes

There are difference ways that fencing can be implemented in.

6.3.1 Power Fencing

The first and probably best solution is power fencing. Power fencing means, that access is prevented by cutting power to the cluster member. For physical hosts, this can mean accessing a networked powerboard and switching off the power of a specific outlet or accessing the Base Management Controllers (BMCs) of the host and powering it off that way. Fencing the hosts by *reseting* the power, not *cutting* the power enables the host to recover from an error, if it is related to the operating system and can be fixed by a power cycle. This can increase the durability of a cluster. Any device that shares power with a node is unsuitable for fencing, because the cluster resource manager can not determine if the node was fenced.

6.3.2 Network Fencing

The second type of fencing is network fencing. It works by disconnecting the host from the shared resource on the network layer. This makes sense if a SAN or NAS is used to host shared storage.

6.3.3 Virtual Fencing

Virtual fencing is a term for fencing by using the hypervisor of the cluster members, which are guests. This is commonly done either through multicast communication with all VM hypervisors, which host the cluster members, or virtual serial connections from the guest through the hypervisor's Application Programming Interface (API). Hosting HA clusters as VMs is not encouraged for production, as the added complexity of the hypervisor and the host **OS!** (**OS!**) increase the susceptibility of the guest **OS!** to failure or hanging.

6.4 Fence-Virt

fence-virt is an ongoing effort from ClusterLabs to build an open source solution for virtual fencing and is the successor to fence_xvm. The upstream of the software is at Github[3].

6.4.1 Components

fence-virt consists of a daemon that is running on the hypervisor and an agent, that is run by the guest. If a cluster node needs to be fenced, a fencing agent contacts a fencing server and asks the other node to be rebooted or powered off.

6.4.2 features

The feature set of fence-virt is currently quite narrow, but more features are listed in the to do list. The client or fencing agent running on the cluster node, which wants to fence another node, can contact the server over either multicast IP, serial connection or tcp. It currently only supports libvirt. The fencing agent can send different operations through the established connection, for example a request for the list of running VMs, to shutdown or reboot a VM or no operation (null), that is only there to test the connection.

6.4.3 Fencing

fence-virt already supports contacting a fencing daemon or server over multicast UDP, TCP, serial connection or VMchannel. The latter two obviously are only of use, if all cluster nodes are guests on the same host.

[3] `https://github.com/ClusterLabs/fence-virt`

6.4.4 Frontends

multicast If multicast is used, the agent sends out a multicast UDP packet to the configured multicast group, in which the desired operation, the Universally Unique IDentifier (UUID) of the VM and its domain are contained, as well as other things needed for communication. But those are the most significant in this paragraph. The servers which received the packet then check the signature of the packet, as well as check the list of locally running VMs against the desired UUID and domain, which are contained in the multicast packet. If the VM runs on the local host, then a TCP connection is initiated to the sender of the multicast packet. The agent and the server then authenticate each other using a challenge response protocol. Then the server executes the operation the host sent in the multicast packet, sends back the result to the agent and closes the connection. Fencing over multicast IP addresses is needed in an environment, where the cluster members might be distributed over different VM hosts. Addressing the fencing server over multicast enables fencing of cluster members whose location is not known. For fencing to work, all virtual machine hosts need to run an instance of the fencing daemon.

TCP If the agent is configured to use TCP to contact the server, it will connect to it using the aforementioned protocol, then authenticate itself and the server, followed by the request. Afterwards, it will get the response from the server. It then closes the connection.

Serial If a serial connection is used, the agent opens the serial device and initiates it with the configured settings (serial speed, word length, partity and stop bits, ...). It then transmits its request and receives the response. Afterwards, it cloes the serial port by hanging up. There is no authentication, instead, the folder the serial devices are in on the host is used to restrict access to the fencing daemon.

VMchannel VM channel works over a virtual serial device, which is virtualized using libvirt and virtio und Linux. The client connects to the daemon using Vmchannel and connects to the VM channel IP address, as well as the channel port. It then speaks over the channel as over a serial connection. There is no authentication, instead, the folder the VM channel devices are in on the host is used to restrict access to the fencing daemon.

6.4.5 Backends

fence-virt supports a variety of backends, with which fencing can be implemented.

libvirt libvirt is the standard fencing backend and can be used to fence guests, which reside on the same host as the recipient of the fencing request.

libvirt-qmf The libvirt-qmf backend uses an Apache QPID message broker to route fencing requests to the host, which houses the node, that should be fenced. This enables the use of fence-virt on networks, which are not multicast friendly.

checkpoint fence-virt has an additional backend called "checkpoint", which uses CMAN, CPC and OpenAIS checkpoints to track virtual machines across a cluster, which uses CMAN and OpenAIS as cluster stack. The information is stored in checkpoints in OpenAIS and is used to route fencing requests to the host, which runs the VM to be fenced, over the OpenAIS communication layer. Checkpoint can fence VMs based on the name or the UUID.

6.4.6 Configuration of the daemon

The fencing daemon (fence_virtd) takes its configuration largely from a configuration file, whose syntax is oriented around curly brackets and sections. The syntax is described in fence_virt.conf(8) Every instance of fence_virtd needs its own configuration file. Every instance of the daemon can only listen on one frontend and one backend. Additionally, there is no possibility to use different credentials on a cluster. Therefore, each cluster needs its own instance. To prevent a cluster node from fencing other guests, a list of allowed fencing targets can be defined, based on the UUID of the guest.

6.4.7 Configuration of the client

fence_virt can be configurd either through parameters given in stdin or program arguments. fence_virt is usually called through pacemaker or stonithd. As expected, all the communication methods and necessary settings can be configured, as well as the Time To Live (TTL) for the multicast packets, the retransmit time for those, and the VM's name or UUID.

```
fence_virtd {
        listener = "serial";
        backend = "libvirt";
        foreground = "yes";
}

listeners {
        multicast {
                key_file = "/etc/cluster/fence_xvm.key";
                address = "225.0.0.12";
                # Needed on Fedora systems
                interface = "virbr5";
        }
        serial {
                path = "/etc/cluster/mirror/";
                mode = "serial";
        }
}

backends {
        libvirt {
                uri = "qemu:///system";
        }
}

groups {
        group {
                ip = "192.168.181.66";
                ip = "192.168.181.67";
                uuid = "28b9d5e8-f28a-4c5d-9853-e098d2a2a5d1";
                uuid = "9385d78a-872b-4657-8ae7-7e5ba8b637be";
        }
}
```

Figure 1: Example configuration file for serial frontend and libvirt backend

6.4.8 Security

The multicast packets are authenticated using the shared key, 6 byte of randomness from */dev/urandom* and a hash function, not a Hashed Message Authentication Code (HMAC). In the subsequent TCP connection, the two participants authenticate each other using two seperate challenge response exchanges, where in each exchange, a predetermined side of the connection issues a challenge and checks the response. The authentication scheme employed by the software is flawed, as the software uses 6 byte of randonness from */dev/urandom* as Number used ONly onCE (NONCE), with the sequence number being the time of day. Also, the digest function is not a HMAC, but a regular hash function.

The way virt-fenceuse the function does not introduce any weaknesses the simple construct usually introduce in a cryptographic system, as the input is of fixed length. the agent authenticates itself to the daemon, if multicast is used, over a challenge response protocol, which uses a shared key and a NONCE to authenticate the client to the server.

The NONCE consists of 6 byte taken from */dev/urandom* on the fencing server. The shared key has to be known to all nodes in the cluster and the fencing daemons, therefore the security of them has to be ensured. A successful attack on a cluster node completely subverts the security of the cluster and enables an attacker to carry out a DOS attack on the cluster by shutting down all cluster nodes. The software uses Network Security Services (NSS) to hash the messages and currently only supports the Secure Hash Algorithm (SHA) family of hash algorithms.

The code for the authentication is available at `https://github.com/ClusterLabs/fence-virt/blob/master/common/simple_auth.c`.

6.4.9 Developement

As of the time of the writing, fence-virt is available in version 0.4.0 after 6 years of development. This is rather poor, considering the rather narrow feature set. It is not expected, that new features are introduced in the near future. The small amount of code that fence-virt consists of contain some jumps and has some comments that clearly state that the code is not satisfactory or needs to be refactored. Therefore, the state of the code is not satisfactory.

7 Quorum

Quroum (plural quora) in the cluster world describes the minimum number of hosts in a group that the cluster requires to function. If a split-brain situation occurs where the cluster split in two, quorum is important to figure out which part of the cluster should continue to operate. A quorum is achieved when more than half of the nodes are reachable.

8 Software

To implement High Availability, different types of software for different purposes are used. As mentioned before, a cluster manager is used for communication between the different hosts that provide services in the cluster. Services are commonly called "resources" in the cluster world. Another software is the resource manager, which manages the resources themselves. This includes starting, stopping and monitoring the resources.

8.1 Pacemaker

Pacemaker is a ressource manager for HA clusters. Every node in a cluster runs an instance of it. It detects failures of system daemons and the physical or virtual host itself. Also, it handles the complex relationship between the different services and resources, which are formulated by the administrator. Pacemaker manages resources using resource agents, which are binaries or scripts, that are run with different parameters and return standardized return codes, from which Pacemaker figures out the status of the resource. There exist many resource agents for different purposes and based on different standards. One such standard is "ocf". Others are "lsb", "service", "systemd" or "stonith". "lsb" and "systemd" are simple wrappers around **lsb!** (**lsb!**) compliant init scripts, and hence the normal init system, or systemd. "stonith" are scripts for STONITH resources and "ocf" are an extension of the "lsb" standard. Special resource agents are ususally much superior to "lsb" or "systemd" resource agents, because they can perform checks on the actual functionality of the service instead of the reported status. One example for such a special resource agent is the one for "Apache" Web server, which follows the "ocf" standard. It checks the status page of the web server by loading it. In contrast, a resource agent following the "lsb" standard would only be able to check if the "Apache" process is running, not if it actually functional. Pacemaker uses the communication facilities, which are provided by Corosync to communicate with the Pacemaker instances on the other nodes. It is primarily written in C.

8.1.1 History

Pacemaker is primarily a joint effort between Red Hat and Novell, but also some other, smaller companies and the open source community. The project exists since 2004.

8.1.2 Components

- Cluster Information Base (CIB)
- Cluster Resource Management daemon (CRMd)
- Local Resource Management daemon (LRMd)
- Policy Engine (PE)
- Fencing daemon (STONITHd)

8.2 Corosync

Corosync is a descendent of the older OpenAIS project and is run as a system daemon. The software is open source and licensed under the BSD licenses. It provides group membership communication and enables the cluster members to communicate with each other on the application layer to communicate service outages, corruption and scheduled shutdowns of hosts other hosts. The daemon is written in C and is commonly used on the Linux platform together with pacemaker or cman.

The latest version of Corosync as of the time of the writing was 2.3.5.

8.2.1 History

8.2.2 Services

Corosync takes up the whole task of managing the cluster and distributing the services. For this purpose, it provides 4 different service engines that can be used with the native C API:

confdb Configuration and Statistics database

cpq Closed Process Group

quorum Provides notifications of gain or loss of quorum

sam Simple Availability Manager

The native C API of Corosync uses shared memory for high throughput and low latency and enables the efficient use of the API for mass sending of messages to other cluster

members. Corosync can use **UDP!** over IP or Infiniband for communication and supports different rings of communication for redundancy and prevention of false positives in the detection of unresponsive or dead hosts.

8.2.3 Locality Constraints

Corosync can keep services together on a single host and express locality constraints in general, like pinning services to a certain host or bundling services together, which should always be together, like a cluster IP and a web service.

8.2.4 Failover

Corosync uses totems[AMMS+95] to check the availability of the hosts in the cluster. It can use different "rings"[4] to detect a failure in the network infrastructure between the host, but not actually causing any failover action on the host themselves.

8.3 PCS

Pacemaker/Corosync configuration system (pcs) is a modern program to configure and maintain clusters made of pacemaker and Corosync. It provides a unified command line interface to the configuration options of Corosync and pacemaker, which enables the easy configuration of the aforementioned software. pcs manages the hosts using a daemon called "pcsd", to which the other nodes authenticate using a Unix account. The daemon runs as root user and manages the CIB and the status of pacemaker and Corosync on the host. When setting up a cluster using pcs, it does not generate an authkey file by default to secure the cluster traffic. This must be done to the user, because it takes several minutes. The upstream url is `https://github.com/feist/pcs`. The software is primarily maintained by Chris Feist of Red Hat.

8.4 The whole picture

8.5 Availability of the software in different Linux distributions

[4] Different communication paths between the hosts, e.g: physical networks

Distribution	pcs	Pacemaker	Corosync	fence-virt	fence-agents
Debian 7	No	1.1.7	1.4.2	No	3.1.5
Debian 8	No	No	1.4.6	No	3.1.5
Fedora 22	0.9.139	1.1.12	2.3.5	0.3.2	4.0.16
Fedora 23	0.9.141	1.1.12	2.3.5	0.3.2	4.0.16
CentOS 6	0.9.123	1.1.12	1.4.7	0.2.3	3.1.5
CentOS 7	0.9.137	1.1.12	2.3.4	0.3.2	4.0.11
SLES 11 HA Extension	No	Yes, version unknown	Unknown	Unknown	Unknown
SLES 12 HA Extension	No	Yes, version unknown	Unknown	Unknown	Unknown
Arch Linux	No	Yes	Yes	No	Yes

Table 1: Software Availability

8.5.1 Plausibility of building clusters on different distros

The content of subsection 8.5 clearly shows, that different distros have different support for clustering. For example, Debian has basically no support for clustering, whereas all distros that are affiliated with Red Hat have very good support for it. SLES is a different topic. I could not find out what software exactly is used in the different versions, only that clustering is supported. As S.U.S.E Linux Gmbh, the developer of SLES, invested money around the year 2000 in the development of clustering software, it is plausible that, as usual, pacemaker and Corosync is used, probably with crm shell. Building clusters on Debianoid systems is not possible or advised. This includes Ubuntu and Linux Mint.

9 Setting up a HA cluster on Linux

This part is about the real world installation of an HA cluster on Linux using the tools introduced in the former sections.

9.1 Cluster architecture

For the cluster, you will use 3 CentOS 7 VMs, which are connected to three Virtual Local Area Networks (VLANs):

- 192.168.178.0/24 for public cluster resources
- 192.168.180.0/24 for the management of the clusters
- 192.168.181.64/27 for the cluster communication

The services you are going to set up are the following:

- A cluster IP
- A lighttpd web server to host an example web site
- A storage built with a shared virtual hard drive and xfs

Fencing will be implemented using fence-virt over a serial device on the guests. The obvious options for resources are not explained, because it is clear what they do.

9.2 First steps

Changes to the file system must always be done on each node. Changes to the cluster configuration must only be done once.

9.2.1 Installation of the distribution and the packages

The first steps to building the cluster is to install the operating system on the nodes. For that, you need to download the CentOS 7 ISO from the official website[5]. Afterwards, install the distribution together with the High Availability extension and the guest agents in the infrastructure server part. If you did not choose that option, you need to install the packages manually:

[5] `https://www.centos.org/download/`

```
#
# /etc/hosts: static lookup table for host names
#

#<ip-address> <hostname.domain.org> <hostname>
127.0.0.1     localhost.localdomain localhost
::1           localhost.localdomain localhost

192.168.181.180 c7-testcluster-1
192.168.181.181 c7-testcluster-2
192.168.181.182 c7-testcluster-3
```

Figure 2: Example hosts file

```
yum install corosync pacemaker pcs
```

You need to install the **epel!** (**epel!**) repository to get lighttpd package

```
yum install epel-release
```

Now install lighttpd:

```
yum install lighttpd
```

9.2.2 Network setup

eth0 Public network

eth1 Cluster network

eth2 Management network

During the setup, use a suitable hostname for your purpose and set up networking. Afterwards, you should set up the firewall rules on the nodes that allow cluster traffic on the cluster network adapter and access to the normal services on the public interface. So you can use nice names in the cluster setup, set up the host files on the cluster nodes to contain entries for the hostnames. The entries must point to the cluster IPs on the cluster internal network. Configure the network interfaces using "nmtui" and enable them to be configured and brought up on boot. Deactivate and reactivate the interface to apply the changes.

```
1  # sample configuration for iptables service
2  # you can edit this manually or use system-config-firewall
3  # please do not ask us to add additional ports/services to this default
       configuration
4  *filter
5  :INPUT DROP [0:0]
6  :FORWARD ACCEPT [0:0]
7  :OUTPUT ACCEPT [0:0]
8  -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
9  -A INPUT -p icmp -j ACCEPT
10 -A INPUT -i lo -j ACCEPT
11 -A INPUT -i eth0 -p tcp -m conntrack --ctstate NEW -m tcp --dport 80 -j
       ACCEPT
12 -A INPUT -i eth1 -p tcp -m conntrack --ctstate NEW -m tcp --dport 22 -j
       ACCEPT
13 -A INPUT -i eth2 -p tcp -m conntrack --ctstate NEW -m multiport --dports
        2224,21064 -j ACCEPT
14 -A INPUT -i eth2 -p udp -m conntrack --ctstate NEW -m udp --dport 5405 -
       j ACCEPT
15 -A INPUT -i eth2 -m addrtype --dst-type MULTICAST -j ACCEPT
16 -A INPUT -j REJECT --reject-with icmp-host-prohibited
17 -A FORWARD -j REJECT --reject-with icmp-host-prohibited
18 COMMIT
```

Figure 3: Example firewall rules

9.2.3 Security

Afterwards, firewall rules must be set up that allow SSH from the management network and the cluster traffic on the cluster network. Install the package"iptables-services" with yum and enable starting the iptables service with "systemctl enable iptables.service" Place the rules in "/etc/sysconfig/iptables". Afterwards, disable firewalld, the native firewall manager of CentOS 7 with "systemctl disable firewalld.service" and stop it with "systemctl stop firewalld.service". Make sure the rules are compliant with the iptables-save format and test it using "systemctl start iptables.service". If you use IPv6, use ip6tables instead of iptables or additionally to it. It is advised to disable SSH access over IPv6 by removing the "-A INPUT -p tcp -m conntrack –ctstate NEW -m tcp –dport 22 -j ACCEPT" line from /etc/sysconfig/ip6tables. pcsd uses a special Unix user account to start, stop and manipulate Pacemaker and Corosync. By default it uses the

```
base64 < /etc/corosync/authkey
base64 -d | /etc/corosync/authkey <<EOF
Foobar
EOF
```

Figure 4: Command lines to transport authkey

"hacluster" user, which is correctly configured on CentOS 7 to allow for correct usage by pcsd. The account needs a password, which needs to be set with the "passwd' utility. To secure the cluster traffic, it is possible and advised to encrypt and authenticate traffic using a shared RSA key called 'authkey", that is in "/etc/corosync". To generate the key, "corosync-keygen" must be executed. All nodes need to use the same key. Only root must be able to access the file, therefore the owner and group must be "root" and the access rights must the 600. The content of the file is confidential. If it is leaked, an attacker could hijack the cluster. To transport the file to the other nodes, you can get creative. I base64ed the file and then copy and pasted it over ssh and my terminal window to other hosts.

9.2.4 lighttpd configuration

The configuration of lighttpd must be changed to be able to serve the content. It is in "/etc/lighttpd/". In the setup, I simply stored some files in the web server's directory and enabled directory listing to be able to see them all. I stored the configuration file as "/etc/lighttpd/mirror.conf" and rewrote the systemd service file for lighttpd to use the new configuration file by copying "/usr/lib/systemd/system/lighttpd.service" to "/etc/systemd/system/lighttpd.service" and changing the "-F" parameter in the "ExecStart" line to my new configuration file.

9.2.5 Cluster setup

To make pcsd start automaticly, run "systemctl enable pcsd.service" Afterwards, use pcs to set up the cluster on all nodes. In the example, the udpu (unicast udp) transport is used for cluster communcation, because it is more resilient. It is also possible to use udp multicast in networks, which support it. Using multicast instead of unicast takes off load from the nodes when they communicate, because packet duplication is done by the network hardware. The first step is to auth pcsd to all other nodes: Ten you can setup

```
var.log_root   = "/var/log/lighttpd"
var.server_root = "/var/www/mirror"
var.state_dir  = "/var/run"
var.home_dir   = "/var/lib/lighttpd"
var.conf_dir   = "/etc/lighttpd"

server.modules = (
       "mod_access",
       "mod_accesslog"
       )

server.port = 80
server.use-ipv6 = "disable"
server.username = "lighttpd"
server.groupname = "lighttpd"
server.document-root = server_root + "/mirror"
server.pid-file = state_dir + "/lighttpd.pid"
server.errorlog          = log_root + "/error.log"

accesslog.filename       = log_root + "/access.log"

dir-listing.activate    = "enable"
dir-listing.hide-dotfiles = "disable"
dir-listing.exclude     = ( "~$" )
dir-listing.encoding = "UTF-8"
dir-listing.hide-header-file = "disable"
dir-listing.show-header = "disable"
dir-listing.hide-readme-file = "disable"
dir-listing.show-readme = "disable"

server.event-handler = "linux-sysepoll"
server.network-backend = "linux-sendfile"
server.stat-cache-engine = "simple"
server.max-connections = 1024
index-file.names += (
  "index.xhtml", "index.html", "index.htm", "default.htm", "index.php"
)
url.access-deny          = ( "~", ".inc" )
$HTTP["url"] =~ "\.pdf$" {
  server.range-requests = "disable"
}
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi", ".scgi" )
include "conf.d/mime.conf"

server.follow-symlink = "enable"
server.upload-dirs = ( "/var/tmp" )
```


Figure 5: lighttpd configuration

```
1 pcs cluster auth c7-testcluster-1 c7-testcluster-2 c7-testcluster-3 -u
    hacluster
```

Figure 6: pcs cluster auth example

```
1 pcs cluster setup --enable --name c7-testcluster c7-testcluster-1 c7-
    testcluster-2 c7-testcluster-3 --transport udpu
```

Figure 7: pcs cluster setup example

the cluster Then you need to enable encryption and authentication in corosync.conf with the crypto_cipher and crypto_hash options in the "totem" section. Afterwards, run "pcs cluster sync" to make pcs synchronize the corosync.conf file to all other nodes and reload the configuration. Then the cluster must be started with "pcs cluster start". The cluster will initally complain about missing stonith. This is something you will do at the end before you test it.

9.3 Setting up the resources

storage To set up the shared storage, the secondary hard drive must be partitioned and formatted first. CentOS 7 provides "fdisk" and "mkfs.xfs", which can be used to do that. Create a maximum size partition on the secondary hard drive and partition it with xfs. For the shared virtual hard drive, I disabled caching in the options of it on the hypervisor. Afterwards, mount the device to "/mnt", fill it with data and change the SElinux context, so the webserver can access it to "system_u:object_r:httpd_sys_content_t:s0" with "chcon". Then unmount it.

Cluster IP The cluster IP is an IP that is bound to the public interface of the currently active node. It is defined using the "ocf:heartbeat:IPaddr2" resource agent. "pcs resource show ClusterIP" shows more information about the resource.

file system The file system is a simple resource of type "ocf:heartbeat:filesystem". "pcs resource" shows the resource:

lighttpd The resource standard for lighttpd here is "systemd", because there is no resource agent available for it that. Instead, the status that systemd reports is used to

```
totem {
version: 2
crypto_cipher: aes192
crypto_hash: sha256
cluster_name: c7-testcluster
transport: udpu
}

nodelist {
  node {
        ring0_addr: c7-testcluster-1
        nodeid: 1
       }
  node {
        ring0_addr: c7-testcluster-2
        nodeid: 2
       }
  node {
        ring0_addr: c7-testcluster-3
        nodeid: 3
       }
}

quorum {
provider: corosync_votequorum

}

logging {
to_syslog: yes
}
```

Figure 8: corossync example configuration

```
pcs cluster sync
```

Figure 9: pcs cluster sync example

```
pcs cluster start --all
```

Figure 10: pcs cluster start example

```
1 fdisk /dev/vdb
2 mkfs.xfs /dev/vdb1
```

Figure 11: disk formatting

```
1 mount /dev/vdb1 /mnt
2 touch /mnt/foobar
3 chcon -R system_u:object_r:httpd_sys_content_t:s0 /mnt
4 umount /mnt
```

Figure 12: mounting of the shared storage, SElinux adjustement

```
1 pcs resource create ClusterIP ocf:heartbeat:IPaddr2 ip=192.168.178.190
      cidr_netmask=32 nic=eth0 op monitor interval=5s
```

Figure 13: pcs resource ClusterIP

```
1  Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
2   Attributes: ip=192.168.178.190 cidr_netmask=32 nic=eth0
3   Operations: start interval=0s timeout=20s (ClusterIP-start-timeout-20s
        )
4               stop interval=0s timeout=20s (ClusterIP-stop-timeout-20s)
5               monitor interval=5s (ClusterIP-monitor-interval-5s)
```

Figure 14: resource description

```
1 pcs resource create cluster-data ocf:heartbeat:filesystem device=/dev/
      vdb1 directory=/var/www/mirror/ fstype=xfs op monitor interval=20
```

Figure 15: pcs resource shared data

```
1  cluster-data  (ocf::heartbeat:Filesystem): Stopped
```

Figure 16: pcs resource cluster-data

```
1 pcs resource create lighttpd systemd:lighttpd op monitor interval=6s
```

Figure 17: pcs resource lighttpd

```
1 pcs resource group add lighty ClusterIP cluster-data lighttpd
```

Figure 18: pcs resource group

figure out if the resource is still alive.

grouping and ordering the resources The resources must be grouped and ordered to ensure that they are always run on the same host and run in the correct order. The latter is done with order constraints. Creating a group is fairly straight forward and done with "pcs resource group add". "pcs resource" shows the new group. What remains for this paragraph is to define the correct order using "pcs constraint". The program returns the understood order. "pcs constraint" then shows the orders it has:

stonith STONITH in the cluster is done using fence-virt on the guests and fence-virtd on the host. This is the host configuration: The guest configuration is done in pcs. After starting the daemon, it must be tested if communication works. This cone be done by using "fence_virt": The command should print a list of running VMs: After manually testing the functionality, the STONITH resource can be configure using pcs. Fencing is done based on the UUID of the VM, not based on the name. Therefore, a list needs to be configured, which tells pacemaker what cluster node has what UUID. That list is the parameter "pcmk_host_map". The parameter "pcmk_host_check" tells the resource how to figure out what VM can be fenced by the resource. That is not necessary here, because it is explicitely configured using the host map. Therefore it is set to "none".

```
1 Resource Group: lighty
2     ClusterIP (ocf::heartbeat:IPaddr2):    Stopped
3     cluster-data     (ocf::heartbeat:Filesystem): Stopped
4     lighttpd  (systemd:lighttpd):   Stopped
```

Figure 19: pcs resource group view

```
1 pcs constraint order start cluster-data then start lighttpd
```

Figure 20: pcs constraint

```
1 Adding cluster-data lighttpd (kind: Mandatory) (Options: first-action=
    start then-action=start)
```

Figure 21: pcs constraint output

cluster test The last thing to test are lighttpd, failover and STONITH. The first thing is tested by simply accessing the website over the configured IP. If the files show up and can be downloaded, it's fine. STONITH can be tested by running "pcs stonith fenc <node>", where "<node>" stands for the node name. This command also tests for functional failover, if the node of lighttpd is fenced. c7-testcluster-1 was fenced correctly and the resources are now all started on c7-testcluster-2. Looks like everything is finished. Fin!

```
1 Location Constraints:
2 Ordering Constraints:
3   start cluster-data then start lighttpd (kind:Mandatory)
4 Colocation Constraints:
```

Figure 22: pcs constraint view

```
fence_virtd {
        listener = "serial";
        backend = "libvirt";
        foreground = "yes";
}

listeners {
        serial {
                path = "/etc/cluster/clusters/test/";
                mode = "serial";
        }
}

backends {
        libvirt {
                uri = "qemu:///system";
        }
}

groups {
        group {
                uuid = "248d2268-2512-4a75-8398-cafbab580529";
                uuid = "c1e36261-a86b-42f5-89f7-8bf78d936d0c";
                uuid = "653aa17a-d766-455a-b7a0-bdd997ee1d71";
        }
}
```

Figure 23: fence_virtd host config

```
$ fence_virt -D /dev/ttyS0 -o list
```

Figure 24: fencing test

```
c7-testcluster-1   248d2268-2512-4a75-8398-cafbab580529 on
c7-testcluster-2   c1e36261-a86b-42f5-89f7-8bf78d936d0c on
c7-testcluster-3   653aa17a-d766-455a-b7a0-bdd997ee1d71 on
```

Figure 25: fencing list

```
1 $ pcs stonith create fencing fence_virt serial_device=/dev/ttyS0
    pcmk_host_map='c7-testcluster-1:248d2268-2512-4a75-8398-cafbab580529;
    c7-testcluster-2:c1e36261-a86b-42f5-89f7-8bf78d936d0c;c7-testcluster
    -3:653aa17a-d766-455a-b7a0-bdd997ee1d71;' pcmk_host_check=none
```

Figure 26: pcs stonith

```
1 $ pcs stonith fence c7-testcluster-1
```

Figure 27: pcs stonith fencing test

```
$ pcs stonith fence c7-testcluster-1
Node: c7-testcluster-1 fenced
$ pcs status
Cluster name: c7-testcluster
Last updated: Sun Aug 30 19:04:59 2015
Last change: Sun Aug 30 19:04:31 2015
Stack: corosync
Current DC: c7-testcluster-3 (3) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
4 Resources configured


Online: [ c7-testcluster-2 c7-testcluster-3 ]
OFFLINE: [ c7-testcluster-1 ]

Full list of resources:

 Resource Group: lighty
     ClusterIP (ocf::heartbeat:IPaddr2):    Started c7-testcluster-2
     cluster-data     (ocf::heartbeat:Filesystem): Started c7-testcluster
         -2
     lighttpd  (systemd:lighttpd):   Started c7-testcluster-2
 fencing       (stonith:fence_virt): Started c7-testcluster-2

Failed actions:
    fencing_start_0 on c7-testcluster-3 'unknown error' (1): call=18,
        status=Error, exit-reason='none', last-rc-change='Sun Aug 30
        18:56:13 2015', queued=0ms, exec=1040ms


PCSD Status:
  c7-testcluster-1: Offline
  c7-testcluster-2: Online
  c7-testcluster-3: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

Figure 28: pcs stonith test and output

10 Maintaining a Cluster

Maintaining a cluster is a completely different story from setting one up.

maintenance While doing maintenance, the node should be put into standby mode, so it is not suspect to fencing action. A fencing action can occur when the maintenance process takes up too much CPU time for Corosync to work correctly. When a node is put into standby mode, all resources are moved away from it. The syntax for pcs is "pcs cluster standby <node>". Putting nodes into production is done using "pcs cluster unstandby <node>".

adding a node From time to time, a cluster needs to be expanded. This is done with "pcs cluster node add <node>". Everything besides the pure Corosync conf needs to be moved to the new node manually. This includes the authkey.

removing a node If a node needs to be removed, use "pcs cluster node remove <node>".

11 Conclusion

Based on my last experiences, clustering on Linux has adulterated in the last 15 years, at least for small clusters. The complexity is not overwhelming and the concepts are easy to grasp. The configuration is made easier by using "pcs", which eases many tasks. It still lacks support for clustering of resources in different geographical locations like colocations. This can be implemented using "booth"[6]. Easy and cheap replication of data over different nodes is not available, but should be in the near future through DRBD. The current lack of cheap replication makes building a cluster with shared storage, but without a SAN, not suitable for production. Building clusters is only supported on platforms provided by Red Hat (RHEL, CentOS, Fedora, ...) or SUSE (openSUSE, SLES). All in all, clustering on Linux is suitable for businesses now, even without paying experts for integration.

[6] `https://github.com/ClusterLabs/booth`

References

[AMMS+95] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The totem single-ring ordering and membership protocol, 1995.

[Bee15a] Andrew Beekhof. Pacemaker 1.1 clusters from scratch, 2015.

[Bee15b] Andrew Beekhof. Pacemaker 1.1 configuration explained, 2015.

[Car06] Sage A. Weil Scott A. Brandt Ethan L. Miller Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data, 2006.

[fen] Clusterlabs/fence-virt.

[KfaN] Madison Kelly and Chrissie from alteeve's Niche. High-availability clustering in the open source ecosystem.

[nfs] Filesystem comparison: Nfs vs gfs2 vs ocfs2.

[unk] unknown. Ceph architecture.

[vV14] Sander van Vugt. *Pro Linux High Availability Clustering.* Springer, 2014.