

OpenStack

High Availability Guide

current (September 14, 2014)



OpenStack High Availability Guide

Florian Haas

current (2014-09-14)

Copyright © 2012-2014 OpenStack Contributors All rights reserved.

This guide describes how to install, configure, and manage OpenStack for high availability.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Table of Contents

1. Introduction to OpenStack High Availability	1
Stateless vs. Stateful services	2
Active/Passive	2
Active/Active	2
I. HA using active/passive	4
2. The Pacemaker cluster stack	6
Install packages	6
Set up Corosync	6
Starting Corosync	9
Start Pacemaker	9
Set basic cluster properties	10
3. Cloud controller cluster stack	11
Highly available MySQL	11
Highly available RabbitMQ	14
4. API node cluster stack	18
Configure the VIP	18
Highly available OpenStack Identity	18
Highly available OpenStack Image API	20
Highly available Block Storage API	21
Highly available OpenStack Networking server	23
Highly available Telemetry central agent	25
Configure Pacemaker group	26
5. Network controller cluster stack	27
Highly available neutron L3 agent	27
Highly available neutron DHCP agent	28
Highly available neutron metadata agent	28
Manage network resources	29
II. HA using active/active	30
6. Database	32
MySQL with Galera	32
Galera monitoring scripts	34
Other ways to provide a highly available database	34
7. RabbitMQ	36
Install RabbitMQ	36
Configure RabbitMQ	36
Configure OpenStack services to use RabbitMQ	37
8. HAproxy nodes	39
9. OpenStack controller nodes	42
Run OpenStack API and schedulers	42
Memcached	43
10. OpenStack network nodes	44
Run neutron DHCP agent	44
Run neutron L3 agent	44
Run neutron metadata agent	44
A. Community support	45
Documentation	45
ask.openstack.org	46
OpenStack mailing lists	46

The OpenStack wiki	47
The Launchpad Bugs area	47
The OpenStack IRC channel	48
Documentation feedback	48
OpenStack distribution packages	48

1. Introduction to OpenStack High Availability

Table of Contents

Stateless vs. Stateful services	2
Active/Passive	2
Active/Active	2

High Availability systems seek to minimize two things:

- **System downtime** — occurs when a *user-facing* service is unavailable beyond a specified maximum amount of time, and
- **Data loss** — accidental deletion or destruction of data.

Most high availability systems guarantee protection against system downtime and data loss only in the event of a single failure. However, they are also expected to protect against cascading failures, where a single failure deteriorates into a series of consequential failures.

A crucial aspect of high availability is the elimination of single points of failure (SPOFs). A SPOF is an individual piece of equipment or software which will cause system downtime or data loss if it fails. In order to eliminate SPOFs, check that mechanisms exist for redundancy of:

- Network components, such as switches and routers
- Applications and automatic service migration
- Storage components
- Facility services such as power, air conditioning, and fire protection

Most high availability systems will fail in the event of multiple independent (non-consequential) failures. In this case, most systems will protect data over maintaining availability.

High-availability systems typically achieve uptime of 99.99% or more, which roughly equates to less than an hour of cumulative downtime per year. In order to achieve this, high availability systems should keep recovery times after a failure to about one to two minutes, sometimes significantly less.

OpenStack currently meets such availability requirements for its own infrastructure services, meaning that an uptime of 99.99% is feasible for the OpenStack infrastructure proper. However, OpenStack *does not* guarantee 99.99% availability for individual guest instances.

Preventing single points of failure can depend on whether or not a service is stateless.

Stateless vs. Stateful services

A stateless service is one that provides a response after your request, and then requires no further attention. To make a stateless service highly available, you need to provide redundant instances and load balance them. OpenStack services that are stateless include nova-api, nova-conductor, glance-api, keystone-api, neutron-api and nova-scheduler.

A stateful service is one where subsequent requests to the service depend on the results of the first request. Stateful services are more difficult to manage because a single action typically involves more than one request, so simply providing additional instances and load balancing will not solve the problem. For example, if the Horizon user interface reset itself every time you went to a new page, it wouldn't be very useful. OpenStack services that are stateful include the OpenStack database and message queue.

Making stateful services highly available can depend on whether you choose an active/passive or active/active configuration.

Active/Passive

In an active/passive configuration, systems are set up to bring additional resources online to replace those that have failed. For example, OpenStack would write to the main database while maintaining a disaster recovery database that can be brought online in the event that the main database fails.

Typically, an active/passive installation for a stateless service would maintain a redundant instance that can be brought online when required. Requests are load balanced using a virtual IP address and a load balancer such as HAProxy.

A typical active/passive installation for a stateful service maintains a replacement resource that can be brought online when required. A separate application (such as Pacemaker or Corosync) monitors these services, bringing the backup online as necessary.

Active/Active

In an active/active configuration, systems also use a backup but will manage both the main and redundant systems concurrently. This way, if there is a failure the user is unlikely to notice. The backup system is already online, and takes on increased load while the main system is fixed and brought back online.

Typically, an active/active installation for a stateless service would maintain a redundant instance, and requests are load balanced using a virtual IP address and a load balancer such as HAProxy.

A typical active/active installation for a stateful service would include redundant services with all instances having an identical state. For example, updates to one instance of a database would also update all other instances. This way a request to one instance is the same as a request to any other. A load balancer manages the traffic to these systems, ensuring that operational systems always handle the request.

These are some of the more common ways to implement these high availability architectures, but they are by no means the only ways to do it. The important thing is to make sure

that your services are redundant, and available; how you achieve that is up to you. This document will cover some of the more common options for highly available systems.

Part I. HA using active/passive

Table of Contents

2. The Pacemaker cluster stack	6
Install packages	6
Set up Corosync	6
Starting Corosync	9
Start Pacemaker	9
Set basic cluster properties	10
3. Cloud controller cluster stack	11
Highly available MySQL	11
Highly available RabbitMQ	14
4. API node cluster stack	18
Configure the VIP	18
Highly available OpenStack Identity	18
Highly available OpenStack Image API	20
Highly available Block Storage API	21
Highly available OpenStack Networking server	23
Highly available Telemetry central agent	25
Configure Pacemaker group	26
5. Network controller cluster stack	27
Highly available neutron L3 agent	27
Highly available neutron DHCP agent	28
Highly available neutron metadata agent	28
Manage network resources	29

2. The Pacemaker cluster stack

Table of Contents

Install packages	6
Set up Corosync	6
Starting Corosync	9
Start Pacemaker	9
Set basic cluster properties	10

OpenStack infrastructure high availability relies on the [Pacemaker](#) cluster stack, the state-of-the-art high availability and load balancing stack for the Linux platform. Pacemaker is storage and application-agnostic, and is in no way specific to OpenStack.

Pacemaker relies on the [Corosync](#) messaging layer for reliable cluster communications. Corosync implements the Totem single-ring ordering and membership protocol. It also provides UDP and InfiniBand based messaging, quorum, and cluster membership to Pacemaker.

Pacemaker interacts with applications through *resource agents* (RAs), of which it supports over 70 natively. Pacemaker can also easily use third-party RAs. An OpenStack high-availability configuration uses existing native Pacemaker RAs (such as those managing MySQL databases or virtual IP addresses), existing third-party RAs (such as for RabbitMQ), and native OpenStack RAs (such as those managing the OpenStack Identity and Image Services).

Install packages

On any host that is meant to be part of a Pacemaker cluster, you must first establish cluster communications through the Corosync messaging layer. This involves Install the following packages (and their dependencies, which your package manager will normally install automatically):

- `pacemaker` Note that the `crm` shell should be downloaded separately.
- `crmsh`
- `corosync`
- `cluster-glue`
- `fence-agents` (Fedora only; all other distributions use fencing agents from `cluster-glue`)
- `resource-agents`

Set up Corosync

Besides installing the `corosync` package, you must also create a configuration file, stored in `/etc/corosync/corosync.conf`. Most distributions ship an example configura-

tion file (`corosync.conf.example`) as part of the documentation bundled with the `corosync` package. An example Corosync configuration file is shown below:

Corosync configuration file (`corosync.conf`).

```
totem {
    version: 2

    # Time (in ms) to wait for a token ❶
    token: 10000

    # How many token retransmits before forming a new
    # configuration
    token_retransmits_before_loss_const: 10

    # Turn off the virtual synchrony filter
    vsftype: none

    # Enable encryption ❷
    secauth: on

    # How many threads to use for encryption/decryption
    threads: 0

    # This specifies the redundant ring protocol, which may be
    # none, active, or passive. ❸
    rrp_mode: active

    # The following is a two-ring multicast configuration. ❹
    interface {
        ringnumber: 0
        bindnetaddr: 192.168.42.0
        mcastaddr: 239.255.42.1
        mcastport: 5405
    }
    interface {
        ringnumber: 1
        bindnetaddr: 10.0.42.0
        mcastaddr: 239.255.42.2
        mcastport: 5405
    }
}

amf {
    mode: disabled
}

service {
    # Load the Pacemaker Cluster Resource Manager ❺
    ver: 1
    name: pacemaker
}

aisexec {
    user: root
    group: root
}

logging {
```

```
fileline: off
to_stderr: yes
to_logfile: no
to_syslog: yes
syslog_facility: daemon
debug: off
timestamp: on
logger_subsys {
    subsys: AMF
    debug: off
    tags: enter|leave|trace1|trace2|trace3|trace4|trace6
}
```

- ❶ The `token` value specifies the time, in milliseconds, during which the Corosync token is expected to be transmitted around the ring. When this timeout expires, the token is declared lost, and after `token_retransmits_before_loss_const` lost tokens the non-responding *processor* (cluster node) is declared dead. In other words, `token × token_retransmits_before_loss_const` is the maximum time a node is allowed to not respond to cluster messages before being considered dead. The default for `token` is 1000 (1 second), with 4 allowed retransmits. These defaults are intended to minimize failover times, but can cause frequent "false alarms" and unintended failovers in case of short network interruptions. The values used here are safer, albeit with slightly extended failover times.
- ❷ With `secauth` enabled, Corosync nodes mutually authenticate using a 128-byte shared secret stored in `/etc/corosync/authkey`, which may be generated with the `corosync-keygen` utility. When using `secauth`, cluster communications are also encrypted.
- ❸ In Corosync configurations using redundant networking (with more than one interface), you must select a Redundant Ring Protocol (RRP) mode other than `none`. `active` is the recommended RRP mode.
- ❹ There are several things to note about the recommended interface configuration:
 - The `ringnumber` must differ between all configured interfaces, starting with 0.
 - The `bindnetaddr` is the *network* address of the interfaces to bind to. The example uses two network addresses of /24 IPv4 subnets.
 - Multicast groups (`mcastaddr`) *must not* be reused across cluster boundaries. In other words, no two distinct clusters should ever use the same multicast group. Be sure to select multicast addresses compliant with [RFC 2365, "Administratively Scoped IP Multicast"](#).
 - For firewall configurations, note that Corosync communicates over UDP only, and uses `mcastport` (for receives) and `mcastport-1` (for sends).
- ❺ The service declaration for the `pacemaker` service may be placed in the `corosync.conf` file directly, or in its own separate file, `/etc/corosync/service.d/pacemaker`.

Once created, the `corosync.conf` file (and the `authkey` file if the `secauth` option is enabled) must be synchronized across all cluster nodes.

Starting Corosync

Corosync is started as a regular system service. Depending on your distribution, it may ship with a LSB (System V style) init script, an upstart job, or a systemd unit file. Either way, the service is usually named `corosync`:

- `/etc/init.d/corosync start` (LSB)
- `service corosync start` (LSB, alternate)
- `start corosync` (upstart)
- `systemctl start corosync` (systemd)

You can now check the Corosync connectivity with two tools.

The `corosync-cfgtool` utility, when invoked with the `-s` option, gives a summary of the health of the communication rings:

```
# corosync-cfgtool -s
Printing ring status.
Local node ID 435324542
RING ID 0
    id      = 192.168.42.82
    status  = ring 0 active with no faults
RING ID 1
    id      = 10.0.42.100
    status  = ring 1 active with no faults
```

The `corosync-objctl` utility can be used to dump the Corosync cluster member list:

```
# corosync-objctl runtime.totem.pg.mrp.srp.members
runtime.totem.pg.mrp.srp.435324542.ip=r(0) ip(192.168.42.82) r(1) ip(10.0.42.100)
runtime.totem.pg.mrp.srp.435324542.join_count=1
runtime.totem.pg.mrp.srp.435324542.status=joined
runtime.totem.pg.mrp.srp.983895584.ip=r(0) ip(192.168.42.87) r(1) ip(10.0.42.254)
runtime.totem.pg.mrp.srp.983895584.join_count=1
runtime.totem.pg.mrp.srp.983895584.status=joined
```

You should see a `status=joined` entry for each of your constituent cluster nodes.

Start Pacemaker

Once the Corosync services have been started, and you have established that the cluster is communicating properly, it is safe to start `pacemakerd`, the Pacemaker master control process:

- `/etc/init.d/pacemaker start` (LSB)
- `service pacemaker start` (LSB, alternate)
- `start pacemaker` (upstart)
- `systemctl start pacemaker` (systemd)

Once Pacemaker services have started, Pacemaker will create a default empty cluster configuration with no resources. You may observe Pacemaker's status with the `crm_mon` utility:

```
=====
Last updated: Sun Oct  7 21:07:52 2012
Last change: Sun Oct  7 20:46:00 2012 via cibadmin on node2
Stack: openais
Current DC: node2 - partition with quorum
Version: 1.1.6-9971ebba4494012a93c03b40a2c58ec0eb60f50c
2 Nodes configured, 2 expected votes
0 Resources configured.
=====

Online: [ node2 node1 ]
```

Set basic cluster properties

Once your Pacemaker cluster is set up, it is recommended to set a few basic cluster properties. To do so, start the `crm` shell and change into the configuration menu by entering `configure`. Alternatively, you may jump straight into the Pacemaker configuration menu by typing `crm configure` directly from a shell prompt.

Then, set the following properties:

```
property no-quorum-policy="ignore" \ # ❶
    pe-warn-series-max="1000" \      # ❷
    pe-input-series-max="1000" \
    pe-error-series-max="1000" \
    cluster-recheck-interval="5min" # ❸
```

- ❶ Setting `no-quorum-policy="ignore"` is required in 2-node Pacemaker clusters for the following reason: if quorum enforcement is enabled, and one of the two nodes fails, then the remaining node can not establish a *majority* of quorum votes necessary to run services, and thus it is unable to take over any resources. The appropriate workaround is to ignore loss of quorum in the cluster. This is safe and necessary *only* in 2-node clusters. Do not set this property in Pacemaker clusters with more than two nodes.
- ❷ Setting `pe-warn-series-max`, `pe-input-series-max` and `pe-error-series-max` to 1000 instructs Pacemaker to keep a longer history of the inputs processed, and errors and warnings generated, by its Policy Engine. This history is typically useful in case cluster troubleshooting becomes necessary.
- ❸ Pacemaker uses an event-driven approach to cluster state processing. However, certain Pacemaker actions occur at a configurable interval, `cluster-recheck-interval`, which defaults to 15 minutes. It is usually prudent to reduce this to a shorter interval, such as 5 or 3 minutes.

Once you have made these changes, you may `commit` the updated configuration.

3. Cloud controller cluster stack

Table of Contents

Highly available MySQL	11
Highly available RabbitMQ	14

The cloud controller runs on the management network and must talk to all other services.

Highly available MySQL

MySQL is the default database server used by many OpenStack services. Making the MySQL service highly available involves

- Configure a DRBD device for use by MySQL,
- Configure MySQL to use a data directory residing on that DRBD device,
- selecting and assigning a virtual IP address (VIP) that can freely float between cluster nodes,
- Configure MySQL to listen on that IP address,
- managing all resources, including the MySQL daemon itself, with the Pacemaker cluster manager.



Note

[MySQL/Galera](#) is an alternative method of Configure MySQL for high availability. It is likely to become the preferred method of achieving MySQL high availability once it has sufficiently matured. At the time of writing, however, the Pacemaker/DRBD based approach remains the recommended one for OpenStack environments.

Configure DRBD

The Pacemaker based MySQL server requires a DRBD resource from which it mounts the `/var/lib/mysql` directory. In this example, the DRBD resource is simply named `mysql`:

mysql DRBD resource configuration (`/etc/drbd.d/mysql.res`).

```
resource mysql {
    device      minor 0;
    disk        "/dev/data/mysql";
    meta-disk internal;
    on node1 {
        address ipv4 10.0.42.100:7700;
    }
    on node2 {
        address ipv4 10.0.42.254:7700;
    }
}
```

This resource uses an underlying local disk (in DRBD terminology, a *backing device*) named `/dev/data/mysql` on both cluster nodes, `node1` and `node2`. Normally, this would be an LVM Logical Volume specifically set aside for this purpose. The DRBD meta-disk is *internal*, meaning DRBD-specific metadata is being stored at the end of the disk device itself. The device is configured to communicate between IPv4 addresses 10.0.42.100 and 10.0.42.254, using TCP port 7700. Once enabled, it will map to a local DRBD block device with the device minor number 0, that is, `/dev/drbd0`.

Enabling a DRBD resource is explained in detail in [the DRBD User's Guide](#). In brief, the proper sequence of commands is this:

```
# drbdadm create-md mysql❶  
# drbdadm up mysql❷  
# drbdadm -- --force primary mysql❸
```

- ❶ Initializes DRBD metadata and writes the initial set of metadata to `/dev/data/mysql`. Must be completed on both nodes.
- ❷ Creates the `/dev/drbd0` device node, *attaches* the DRBD device to its backing store, and *connects* the DRBD node to its peer. Must be completed on both nodes.
- ❸ Kicks off the initial device synchronization, and puts the device into the *primary* (readable and writable) role. See [Resource roles](#) (from the DRBD User's Guide) for a more detailed description of the primary and secondary roles in DRBD. Must be completed *on one node only*, namely the one where you are about to continue with creating your filesystem.

Creating a file system

Once the DRBD resource is running and in the primary role (and potentially still in the process of running the initial device synchronization), you may proceed with creating the filesystem for MySQL data. XFS is the generally recommended filesystem:

```
# mkfs -t xfs /dev/drbd0
```

You may also use the alternate device path for the DRBD device, which may be easier to remember as it includes the self-explanatory resource name:

```
# mkfs -t xfs /dev/drbd/by-res/mysql
```

Once completed, you may safely return the device to the secondary role. Any ongoing device synchronization will continue in the background:

```
# drbdadm secondary mysql
```

Prepare MySQL for Pacemaker high availability

In order for Pacemaker monitoring to function properly, you must ensure that MySQL's database files reside on the DRBD device. If you already have an existing MySQL database, the simplest approach is to just move the contents of the existing `/var/lib/mysql` directory into the newly created filesystem on the DRBD device.



Warning

You must complete the next step while the MySQL database server is shut down.


```
# mount /dev/drbd/by-res/mysql /mnt
# mv /var/lib/mysql/* /mnt
# umount /mnt
```

For a new MySQL installation with no existing data, you may also run the `mysql_install_db` command:

```
# mount /dev/drbd/by-res/mysql /mnt
# mysql_install_db --datadir=/mnt
# umount /mnt
```

Regardless of the approach, the steps outlined here must be completed on only one cluster node.

Add MySQL resources to Pacemaker

You can now add the Pacemaker configuration for MySQL resources. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_ip_mysql ocf:heartbeat:IPaddr2 \
  params ip="192.168.42.101" cidr_netmask="24" \
  op monitor interval="30s"
primitive p_drbd_mysql ocf:linbit:drbd \
  params drbd_resource="mysql" \
  op start timeout="90s" \
  op stop timeout="180s" \
  op promote timeout="180s" \
  op demote timeout="180s" \
  op monitor interval="30s" role="Slave" \
  op monitor interval="29s" role="Master"
primitive p_fs_mysql ocf:heartbeat:Filesystem \
  params device="/dev/drbd/by-res/mysql" \
  directory="/var/lib/mysql" \
  fstype="xfs" \
  options="relatime" \
  op start timeout="60s" \
  op stop timeout="180s" \
  op monitor interval="60s" timeout="60s"
primitive p_mysql ocf:heartbeat:mysql \
  params additional_parameters="--bind-address=50.56.179.138" \
  config="/etc/mysql/my.cnf" \
  pid="/var/run/mysqld/mysqld.pid" \
  socket="/var/run/mysqld/mysqld.sock" \
  log="/var/log/mysql/mysqld.log" \
  op monitor interval="20s" timeout="10s" \
  op start timeout="120s" \
  op stop timeout="120s"
group g_mysql p_ip_mysql p_fs_mysql p_mysql
ms ms_drbd_mysql p_drbd_mysql \
  meta notify="true" clone-max="2"
colocation c_mysql_on_drbd inf: g_mysql ms_drbd_mysql:Master
order o_drbd_before_mysql inf: ms_drbd_mysql:promote g_mysql:start
```

This configuration creates

- `p_ip_mysql`, a virtual IP address for use by MySQL (192.168.42.101),
- `p_fs_mysql`, a Pacemaker managed filesystem mounted to `/var/lib/mysql` on whatever node currently runs the MySQL service,

- `ms_drbd_mysql`, the *master/slave* set managing the `mysql` DRBD resource,
- a `service` group and `order` and `colocation` constraints to ensure resources are started on the correct nodes, and in the correct sequence.

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required. For example, you may enter `edit p_ip_mysql` from the `crm configure` menu and edit the resource to match your preferred virtual IP address.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the MySQL service, and its dependent resources, on one of your nodes.

Configure OpenStack services for highly available MySQL

Your OpenStack services must now point their MySQL configuration to the highly available, virtual cluster IP address—rather than a MySQL server’s physical IP address as you normally would.

For OpenStack Image, for example, if your MySQL service IP address is 192.168.42.101 as in the configuration explained here, you would use the following line in your OpenStack Image registry configuration file (`glance-registry.conf`):

```
sql_connection = mysql://glancedbadmin:<password>@192.168.42.101/glance
```

No other changes are necessary to your OpenStack configuration. If the node currently hosting your database experiences a problem necessitating service failover, your OpenStack services may experience a brief MySQL interruption, as they would in the event of a network hiccup, and then continue to run normally.

Highly available RabbitMQ

RabbitMQ is the default AMQP server used by many OpenStack services. Making the RabbitMQ service highly available involves:

- configuring a DRBD device for use by RabbitMQ,
- configuring RabbitMQ to use a data directory residing on that DRBD device,
- selecting and assigning a virtual IP address (VIP) that can freely float between cluster nodes,
- configuring RabbitMQ to listen on that IP address,
- managing all resources, including the RabbitMQ daemon itself, with the Pacemaker cluster manager.



Note

There is an alternative method of configuring RabbitMQ for high availability. That approach, known as [active-active mirrored queues](#), happens to be the one preferred by the RabbitMQ developers—however it has shown less than ideal consistency and reliability in OpenStack clusters. Thus, at the time of writ-

ing, the Pacemaker/DRBD based approach remains the recommended one for OpenStack environments, although this may change in the near future as RabbitMQ active-active mirrored queues mature.

Configure DRBD

The Pacemaker based RabbitMQ server requires a DRBD resource from which it mounts the `/var/lib/rabbitmq` directory. In this example, the DRBD resource is simply named `rabbitmq`:

rabbitmq DRBD resource configuration (`/etc/drbd.d/rabbitmq.res`).

```
resource rabbitmq {
    device      minor 1;
    disk        "/dev/data/rabbitmq";
    meta-disk   internal;
    on node1 {
        address ipv4 10.0.42.100:7701;
    }
    on node2 {
        address ipv4 10.0.42.254:7701;
    }
}
```

This resource uses an underlying local disk (in DRBD terminology, a *backing device*) named `/dev/data/rabbitmq` on both cluster nodes, `node1` and `node2`. Normally, this would be an LVM Logical Volume specifically set aside for this purpose. The DRBD `meta-disk` is `internal`, meaning DRBD-specific metadata is being stored at the end of the `disk` device itself. The device is configured to communicate between IPv4 addresses 10.0.42.100 and 10.0.42.254, using TCP port 7701. Once enabled, it will map to a local DRBD block device with the device minor number 1, that is, `/dev/drbd1`.

Enabling a DRBD resource is explained in detail in [the DRBD User's Guide](#). In brief, the proper sequence of commands is this:

```
# drbdadm create-md rabbitmq❶
# drbdadm up rabbitmq❷
# drbdadm -- --force primary rabbitmq❸
```

- ❶ Initializes DRBD metadata and writes the initial set of metadata to `/dev/data/rabbitmq`. Must be completed on both nodes.
- ❷ Creates the `/dev/drbd1` device node, *attaches* the DRBD device to its backing store, and *connects* the DRBD node to its peer. Must be completed on both nodes.
- ❸ Kicks off the initial device synchronization, and puts the device into the `primary` (readable and writable) role. See [Resource roles](#) (from the DRBD User's Guide) for a more detailed description of the primary and secondary roles in DRBD. Must be completed *on one node only*, namely the one where you are about to continue with creating your filesystem.

Create a file system

Once the DRBD resource is running and in the primary role (and potentially still in the process of running the initial device synchronization), you may proceed with creating the filesystem for RabbitMQ data. XFS is generally the recommended filesystem:

```
# mkfs -t xfs /dev/drbd1
```

You may also use the alternate device path for the DRBD device, which may be easier to remember as it includes the self-explanatory resource name:

```
# mkfs -t xfs /dev/drbd/by-res/rabbitmq
```

Once completed, you may safely return the device to the secondary role. Any ongoing device synchronization will continue in the background:

```
# drbdadm secondary rabbitmq
```

Prepare RabbitMQ for Pacemaker high availability

In order for Pacemaker monitoring to function properly, you must ensure that RabbitMQ's `.erlang.cookie` files are identical on all nodes, regardless of whether DRBD is mounted there or not. The simplest way of doing so is to take an existing `.erlang.cookie` from one of your nodes, copying it to the RabbitMQ data directory on the other node, and also copying it to the DRBD-backed filesystem:

```
# scp -a /var/lib/rabbitmq/.erlang.cookie node2:/var/lib/rabbitmq/
# mount /dev/drbd/by-res/rabbitmq /mnt
# cp -a /var/lib/rabbitmq/.erlang.cookie /mnt
# umount /mnt
```

Add RabbitMQ resources to Pacemaker

You may now proceed with adding the Pacemaker configuration for RabbitMQ resources. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_ip_rabbitmq ocf:heartbeat:IPaddr2 \
  params ip="192.168.42.100" cidr_netmask="24" \
  op monitor interval="10s"
primitive p_drbd_rabbitmq ocf:linbit:drbd \
  params drbd_resource="rabbitmq" \
  op start timeout="90s" \
  op stop timeout="180s" \
  op promote timeout="180s" \
  op demote timeout="180s" \
  op monitor interval="30s" role="Slave" \
  op monitor interval="29s" role="Master"
primitive p_fs_rabbitmq ocf:heartbeat:Filesystem \
  params device="/dev/drbd/by-res/rabbitmq" \
  directory="/var/lib/rabbitmq" \
  fstype="xfs" options="relatime" \
  op start timeout="60s" \
  op stop timeout="180s" \
  op monitor interval="60s" timeout="60s"
primitive p_rabbitmq ocf:rabbitmq:rabbitmq-server \
  params nodename="rabbit@localhost" \
  mnesia_base="/var/lib/rabbitmq" \
  op monitor interval="20s" timeout="10s"
group g_rabbitmq p_ip_rabbitmq p_fs_rabbitmq p_rabbitmq
ms ms_drbd_rabbitmq p_drbd_rabbitmq \
  meta notify="true" master-max="1" clone-max="2"
colocation c_rabbitmq_on_drbd inf: g_rabbitmq ms_drbd_rabbitmq:Master
```

```
order o_drbd_before_rabbitmq inf: ms_drbd_rabbitmq:promote g_rabbitmq:start
```

This configuration creates

- `p_ip_rabbitmq`, a virtual IP address for use by RabbitMQ (192.168.42.100),
- `p_fs_rabbitmq`, a Pacemaker managed filesystem mounted to `/var/lib/rabbitmq` on whatever node currently runs the RabbitMQ service,
- `ms_drbd_rabbitmq`, the *master/slave set* managing the `rabbitmq` DRBD resource,
- a service group and order and colocation constraints to ensure resources are started on the correct nodes, and in the correct sequence.

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required. For example, you may enter `edit p_ip_rabbitmq` from the `crm configure` menu and edit the resource to match your preferred virtual IP address.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the RabbitMQ service, and its dependent resources, on one of your nodes.

Configure OpenStack services for highly available RabbitMQ

Your OpenStack services must now point their RabbitMQ configuration to the highly available, virtual cluster IP address—rather than a RabbitMQ server’s physical IP address as you normally would.

For OpenStack Image, for example, if your RabbitMQ service IP address is 192.168.42.100 as in the configuration explained here, you would use the following line in your OpenStack Image API configuration file (`glance-api.conf`):

```
rabbit_host = 192.168.42.100
```

No other changes are necessary to your OpenStack configuration. If the node currently hosting your RabbitMQ experiences a problem necessitating service failover, your OpenStack services may experience a brief RabbitMQ interruption, as they would in the event of a network hiccup, and then continue to run normally.

4. API node cluster stack

Table of Contents

Configure the VIP	18
Highly available OpenStack Identity	18
Highly available OpenStack Image API	20
Highly available Block Storage API	21
Highly available OpenStack Networking server	23
Highly available Telemetry central agent	25
Configure Pacemaker group	26

The API node exposes OpenStack API endpoints onto external network (Internet). It must talk to the cloud controller on the management network.

Configure the VIP

First, you must select and assign a virtual IP address (VIP) that can freely float between cluster nodes.

This configuration creates `p_ip_api`, a virtual IP address for use by the API node (192.168.42.103):

```
primitive p_api-ip ocf:heartbeat:IPaddr2 \  
  params ip="192.168.42.103" cidr_netmask="24" \  
  op monitor interval="30s"
```

Highly available OpenStack Identity

OpenStack Identity is the Identity Service in OpenStack and used by many services. Making the OpenStack Identity service highly available in active / passive mode involves

- Configure OpenStack Identity to listen on the VIP address,
- managing OpenStack Identity daemon with the Pacemaker cluster manager,
- Configure OpenStack services to use this IP address.



Note

Here is the [documentation](#) for installing OpenStack Identity service.

Add OpenStack Identity resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d  
# mkdir openstack  
# cd openstack
```

```
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/keystone
# chmod a+rx *
```

You can now add the Pacemaker configuration for OpenStack Identity resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_keystone ocf:openstack:keystone \
    params config="/etc/keystone/keystone.conf" os_password="secret"
    os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/" \
    op monitor interval="30s" timeout="30s"
```

This configuration creates `p_keystone`, a resource for managing the OpenStack Identity service.

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required. For example, you may enter `edit p_ip_keystone` from the `crm configure` menu and edit the resource to match your preferred virtual IP address.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the OpenStack Identity service, and its dependent resources, on one of your nodes.

Configure OpenStack Identity service

You need to edit your OpenStack Identity configuration file (`keystone.conf`) and change the bind parameters:

On Havana:

```
bind_host = 192.168.42.103
```

On Icehouse, the `admin_bind_host` option lets you use a private network for the admin access.

```
public_bind_host = 192.168.42.103
admin_bind_host = 192.168.42.103
```

To be sure all data will be highly available, you should be sure that you store everything in the MySQL database (which is also highly available):

```
[catalog]
driver = keystone.catalog.backends.sql.Catalog
...
[identity]
driver = keystone.identity.backends.sql.Identity
...
```

Configure OpenStack services to use the highly available OpenStack Identity

Your OpenStack services must now point their OpenStack Identity configuration to the highly available, virtual cluster IP address — rather than an OpenStack Identity server's physical IP address as you normally would.

For example with OpenStack Compute, if your OpenStack Identity service IP address is 192.168.42.103 as in the configuration explained here, you would use the following line in your API configuration file (`api-paste.ini`):

```
auth_host = 192.168.42.103
```

You also need to create the OpenStack Identity Endpoint with this IP.

NOTE: If you are using both private and public IP addresses, you should create two Virtual IP addresses and define your endpoint like this:

```
$ keystone endpoint-create --region $KEYSTONE_REGION \
--service-id $service-id --publicurl 'http://PUBLIC_VIP:5000/v2.0' \
--adminurl 'http://192.168.42.103:35357/v2.0' \
--internalurl 'http://192.168.42.103:5000/v2.0'
```

If you are using the horizon dashboard, you should edit the `local_settings.py` file:

```
OPENSTACK_HOST = 192.168.42.103
```

Highly available OpenStack Image API

OpenStack Image Service offers a service for discovering, registering, and retrieving virtual machine images. To make the OpenStack Image API service highly available in active / passive mode, you must:

- Configure OpenStack Image to listen on the VIP address.
- Manage OpenStack Image API daemon with the Pacemaker cluster manager.
- Configure OpenStack services to use this IP address.



Note

Here is the [documentation](#) for installing OpenStack Image API service.

Add OpenStack Image API resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/glance-api
# chmod a+rx *
```

You can now add the Pacemaker configuration for OpenStack Image API resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_glance-api ocf:openstack:glance-api \
    params config="/etc/glance/glance-api.conf" os_password="secrete"
    os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.
103:5000/v2.0/" \
    op monitor interval="30s" timeout="30s"
```

This configuration creates

- `p_glance-api`, a resource for manage OpenStack Image API service

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required. For example, you may enter `edit p_ip_glance-api` from the `crm configure` menu and edit the resource to match your preferred virtual IP address.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the OpenStack Image API service, and its dependent resources, on one of your nodes.

Configure OpenStack Image Service API

Edit `/etc/glance/glance-api.conf`:

```
# We have to use MySQL connection to store data:
sql_connection=mysql://glance:password@192.168.42.101/glance

# We bind OpenStack Image API to the VIP:
bind_host = 192.168.42.103

# Connect to OpenStack Image Registry service:
registry_host = 192.168.42.103

# We send notifications to High Available RabbitMQ:
notifier_strategy = rabbit
rabbit_host = 192.168.42.102
```

Configure OpenStack services to use high available OpenStack Image API

Your OpenStack services must now point their OpenStack Image API configuration to the highly available, virtual cluster IP address — rather than an OpenStack Image API server's physical IP address as you normally would.

For OpenStack Compute, for example, if your OpenStack Image API service IP address is 192.168.42.104 as in the configuration explained here, you would use the following line in your `nova.conf` file:

```
glance_api_servers = 192.168.42.103
```

You must also create the OpenStack Image API endpoint with this IP.



Note

If you are using both private and public IP addresses, you should create two Virtual IP addresses and define your endpoint like this:

```
$ keystone endpoint-create --region $KEYSTONE_REGION \
--service-id $service-id --publicurl 'http://PUBLIC_VIP:9292' \
--adminurl 'http://192.168.42.103:9292' \
--internalurl 'http://192.168.42.103:9292'
```

Highly available Block Storage API

Making the Block Storage (cinder) API service highly available in active / passive mode involves

- Configure Block Storage to listen on the VIP address,
- managing Block Storage API daemon with the Pacemaker cluster manager,
- Configure OpenStack services to use this IP address.



Note

Here is the [documentation](#) for installing Block Storage service.

Add Block Storage API resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/cinder-api
# chmod a+rx *
```

You can now add the Pacemaker configuration for Block Storage API resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_cinder-api ocf:openstack:cinder-api \
    params config="/etc/cinder/cinder.conf" os_password="secrete" os_username="admin" \
    os_tenant_name="admin" keystone_get_token_url="http://192.168.42.103:5000/v2.0/tokens" \
    op monitor interval="30s" timeout="30s"
```

This configuration creates

- `p_cinder-api`, a resource for manage Block Storage API service

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required. For example, you may enter `edit p_ip_cinder-api` from the `crm configure` menu and edit the resource to match your preferred virtual IP address.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the Block Storage API service, and its dependent resources, on one of your nodes.

Configure Block Storage API service

Edit `/etc/cinder/cinder.conf`:

```
# We have to use MySQL connection to store data:
sql_connection=mysql://cinder:password@192.168.42.101/cinder

# We bind Block Storage API to the VIP:
osapi_volume_listen = 192.168.42.103

# We send notifications to High Available RabbitMQ:
notifier_strategy = rabbit
rabbit_host = 192.168.42.102
```

Configure OpenStack services to use highly available Block Storage API

Your OpenStack services must now point their Block Storage API configuration to the highly available, virtual cluster IP address — rather than a Block Storage API server's physical IP address as you normally would.

You must create the Block Storage API endpoint with this IP.



Note

If you are using both private and public IP, you should create two Virtual IPs and define your endpoint like this:

```
$ keystone endpoint-create --region $KEYSTONE_REGION \
  --service-id $service-id --publicurl 'http://PUBLIC_VIP:8776/v1/
%(tenant_id)s' \
  --adminurl 'http://192.168.42.103:8776/v1/%(tenant_id)s' \
  --internalurl 'http://192.168.42.103:8776/v1/%(tenant_id)s'
```

Highly available OpenStack Networking server

OpenStack Networking is the network connectivity service in OpenStack. Making the OpenStack Networking Server service highly available in active / passive mode involves

- Configure OpenStack Networking to listen on the VIP address,
- managing OpenStack Networking API Server daemon with the Pacemaker cluster manager,
- Configure OpenStack services to use this IP address.



Note

Here is the [documentation](#) for installing OpenStack Networking service.

Add OpenStack Networking Server resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/
neutron-server
# chmod a+rx *
```

You can now add the Pacemaker configuration for OpenStack Networking Server resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_neutron-server ocf:openstack:neutron-server \
  params os_password="secrete" os_username="admin" os_tenant_name="admin" \
  keystone_get_token_url="http://192.168.42.103:5000/v2.0/tokens" \
  op monitor interval="30s" timeout="30s"
```

This configuration creates `p_neutron-server`, a resource for manage OpenStack Networking Server service

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required. For example, you may enter `edit p_neutron-server` from the `crm configure` menu and edit the resource to match your preferred virtual IP address.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the OpenStack Networking API service, and its dependent resources, on one of your nodes.

Configure OpenStack Networking server

Edit `/etc/neutron/neutron.conf`:

```
# We bind the service to the VIP:
bind_host = 192.168.42.103

# We bind OpenStack Networking Server to the VIP:
bind_host = 192.168.42.103

# We send notifications to Highly available RabbitMQ:
notifier_strategy = rabbit
rabbit_host = 192.168.42.102

[database]
# We have to use MySQL connection to store data:
connection = mysql://neutron:password@192.168.42.101/neutron
```

Configure OpenStack services to use highly available OpenStack Networking server

Your OpenStack services must now point their OpenStack Networking Server configuration to the highly available, virtual cluster IP address — rather than an OpenStack Networking server's physical IP address as you normally would.

For example, you should configure OpenStack Compute for using highly available OpenStack Networking server in editing `nova.conf` file:

```
neutron_url = http://192.168.42.103:9696
```

You need to create the OpenStack Networking server endpoint with this IP.



Note

If you are using both private and public IP addresses, you should create two Virtual IP addresses and define your endpoint like this:

```
$ keystone endpoint-create --region $KEYSTONE_REGION --service-id $service-id \
--publicurl 'http://PUBLIC_VIP:9696/' \
--adminurl 'http://192.168.42.103:9696/' \
--internalurl 'http://192.168.42.103:9696/'
```

Highly available Telemetry central agent

Telemetry (ceilometer) is the metering and monitoring service in OpenStack. The Central agent polls for resource utilization statistics for resources not tied to instances or compute nodes.



Note

Due to limitations of a polling model, a single instance of this agent can be polling a given list of meters. In this setup, we install this service on the API nodes also in the active / passive mode.

Making the Telemetry central agent service highly available in active / passive mode involves managing its daemon with the Pacemaker cluster manager.



Note

You will find at [this page](#) the process to install the Telemetry central agent.

Add the Telemetry central agent resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/ceilometer-agent-central
# chmod a+rx *
```

You may then proceed with adding the Pacemaker configuration for the Telemetry central agent resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_ceilometer-agent-central ocf:openstack:ceilometer-agent-central \
    params config="/etc/ceilometer/ceilometer.conf" \
    op monitor interval="30s" timeout="30s"
```

This configuration creates

- `p_ceilometer-agent-central`, a resource for manage Ceilometer Central Agent service

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the Ceilometer Central Agent service, and its dependent resources, on one of your nodes.

Configure Telemetry central agent service

Edit `/etc/ceilometer/ceilometer.conf`:

```
# We use API VIP for Identity Service connection:
os_auth_url=http://192.168.42.103:5000/v2.0

# We send notifications to High Available RabbitMQ:
notifier_strategy = rabbit
rabbit_host = 192.168.42.102

[database]
# We have to use MySQL connection to store data:
sql_connection=mysql://ceilometer:password@192.168.42.101/ceilometer
```

Configure Pacemaker group

Finally, we need to create a service group to ensure that virtual IP is linked to the API services resources:

```
group g_services_api p_api-ip p_keystone p_glance-api p_cinder-api \
  p_neutron-server p_glance-registry p_ceilometer-agent-central
```

5. Network controller cluster stack

Table of Contents

Highly available neutron L3 agent	27
Highly available neutron DHCP agent	28
Highly available neutron metadata agent	28
Manage network resources	29

The network controller sits on the management and data network, and needs to be connected to the Internet if a VM needs access to it.



Note

Both nodes should have the same hostname since the Networking scheduler will be aware of one node, for example a virtual router attached to a single L3 node.

Highly available neutron L3 agent

The neutron L3 agent provides L3/NAT forwarding to ensure external network access for VMs on tenant networks. High availability for the L3 agent is achieved by adopting Pacemaker.



Note

Here is the [documentation](#) for installing neutron L3 agent.

Add neutron L3 agent resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/
neutron-agent-l3
# chmod a+rx neutron-l3-agent
```

You may now proceed with adding the Pacemaker configuration for neutron L3 agent resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_neutron-l3-agent ocf:openstack:neutron-agent-l3 \
    params config="/etc/neutron/neutron.conf" \
    plugin_config="/etc/neutron/l3_agent.ini" \
    op monitor interval="30s" timeout="30s"
```

This configuration creates

- `p_neutron-l3-agent`, a resource for manage Neutron L3 Agent service

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the neutron L3 agent service, and its dependent resources, on one of your nodes.



Note

This method does not ensure a zero downtime since it has to recreate all the namespaces and virtual routers on the node.

Highly available neutron DHCP agent

Neutron DHCP agent distributes IP addresses to the VMs with `dnsmasq` (by default). High availability for the DHCP agent is achieved by adopting Pacemaker.



Note

Here is the [documentation](#) for installing neutron DHCP agent.

Add neutron DHCP agent resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/
neutron-agent-dhcp
# chmod a+rx neutron-agent-dhcp
```

You may now proceed with adding the Pacemaker configuration for neutron DHCP agent resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_neutron-dhcp-agent ocf:openstack:neutron-agent-dhcp \
  params config="/etc/neutron/neutron.conf" \
  plugin_config="/etc/neutron/dhcp_agent.ini" \
  op monitor interval="30s" timeout="30s"
```

This configuration creates:

- `p_neutron-agent-dhcp`, a resource for manage Neutron DHCP Agent service.

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the neutron DHCP agent service, and its dependent resources, on one of your nodes.

Highly available neutron metadata agent

Neutron metadata agent allows Compute API metadata to be reachable by VMs on tenant networks. High availability for the metadata agent is achieved by adopting Pacemaker.



Note

Here is the [documentation](#) for installing Neutron Metadata Agent.

Add neutron metadata agent resource to Pacemaker

First of all, you need to download the resource agent to your system:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/
neutron-metadata-agent
# chmod a+rx neutron-metadata-agent
```

You may now proceed with adding the Pacemaker configuration for neutron metadata agent resource. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
primitive p_neutron-metadata-agent ocf:openstack:neutron-metadata-agent \
  params config="/etc/neutron/neutron.conf" \
  plugin_config="/etc/neutron/metadata_agent.ini" \
  op monitor interval="30s" timeout="30s"
```

This configuration creates

- `p_neutron-metadata-agent`, a resource for manage Neutron Metadata Agent service

`crm configure` supports batch input, so you may copy and paste the above into your live pacemaker configuration, and then make changes as required.

Once completed, commit your configuration changes by entering `commit` from the `crm configure` menu. Pacemaker will then start the neutron metadata agent service, and its dependent resources, on one of your nodes.

Manage network resources

You can now add the Pacemaker configuration for managing all network resources together with a group. Connect to the Pacemaker cluster with `crm configure`, and add the following cluster resources:

```
group g_services_network p_neutron-l3-agent p_neutron-dhcp-agent \
  p_neutron-metadata_agent
```

Part II. HA using active/active

Table of Contents

6. Database	32
MySQL with Galera	32
Galera monitoring scripts	34
Other ways to provide a highly available database	34
7. RabbitMQ	36
Install RabbitMQ	36
Configure RabbitMQ	36
Configure OpenStack services to use RabbitMQ	37
8. HAproxy nodes	39
9. OpenStack controller nodes	42
Run OpenStack API and schedulers	42
Memcached	43
10. OpenStack network nodes	44
Run neutron DHCP agent	44
Run neutron L3 agent	44
Run neutron metadata agent	44

6. Database

Table of Contents

MySQL with Galera	32
Galera monitoring scripts	34
Other ways to provide a highly available database	34

The first step is installing the database that sits at the heart of the cluster. When we talk about High Availability, we talk about several databases (for redundancy) and a means to keep them synchronized. In this case, we must choose the MySQL database, along with Galera for synchronous multi-master replication.

The choice of database isn't a foregone conclusion; you're not required to use MySQL. It is, however, a fairly common choice in OpenStack installations, so we'll cover it here.

MySQL with Galera

Rather than starting with a vanilla version of MySQL, and then adding Galera, you will want to install a version of MySQL patched for wsrep (Write Set REplication) from <https://launchpad.net/codership-mysql/0.7>. The wsrep API is suitable for configuring MySQL High Availability in OpenStack because it supports synchronous replication.

Note that the installation requirements call for careful attention. Read the guide <https://launchpadlibrarian.net/66669857/README-wsrep> to ensure you follow all the required steps.

Installing Galera through a MySQL version patched for wsrep:

1. Download Galera from <https://launchpad.net/galera/+download>, and install the *.rpms or *.debs, which takes care of any dependencies that your system doesn't already have installed.
2. Adjust the configuration:

In the system-wide `my.cnf` file, make sure `mysqld` isn't bound to `127.0.0.1`, and that `/etc/mysql/conf.d/` is included. Typically you can find this file at `/etc/my.cnf`:

```
[mysqld]
...
!includedir /etc/mysql/conf.d/
...
#bind-address = 127.0.0.1
```

When adding a new node, you must configure it with a MySQL account that can access the other nodes. The new node must be able to request a state snapshot from one of the existing nodes:

3. Specify your MySQL account information in `/etc/mysql/conf.d/wsrep.cnf`:

```
wsrep_sst_auth=wsrep_sst:wspass
```

4. Connect as root and grant privileges to that user:

```
mysql -e "SET wsrep_on=OFF; GRANT ALL ON *.* TO wsrep_sst@%' IDENTIFIED BY 'wspass'";
```

5. Remove user accounts with empty user names because they cause problems:

```
$ mysql -e "SET wsrep_on=OFF; DELETE FROM mysql.user WHERE user='';"
```

6. Set up certain mandatory configuration options within MySQL itself. These include:

```
query_cache_size=0
binlog_format=ROW
default_storage_engine=innodb
innodb_autoinc_lock_mode=2
innodb_doublewrite=1
```

7. Check that the nodes can access each other through the firewall. Depending on your environment, this might mean adjusting iptables, as in:

```
# iptables --insert RH-Firewall-1-INPUT 1 --proto tcp \
--source <my IP>/24 --destination <my IP>/32 --dport 3306 \
-j ACCEPT
# iptables --insert RH-Firewall-1-INPUT 1 --proto tcp \
--source <my IP>/24 --destination <my IP>/32 --dport 4567 \
-j ACCEPT
```

This might also mean configuring any NAT firewall between nodes to allow direct connections. You might need to disable SELinux, or configure it to allow mysqld to listen to sockets at unprivileged ports.

Now you're ready to create the cluster.

Create the cluster

In creating a cluster, you first start a single instance, which creates the cluster. The rest of the MySQL instances then connect to that cluster:

An example of creating the cluster:

1. Start on 10.0.0.10 by executing the command:

```
# service mysql start wsrep_cluster_address=gcomm://
```

1. Connect to that cluster on the rest of the nodes by referencing the address of that node, as in:

```
# service mysql start wsrep_cluster_address=gcomm://10.0.0.10
```

You also have the option to set the `wsrep_cluster_address` in the `/etc/mysql/conf.d/wsrep.cnf` file, or within the client itself. (In fact, for some systems, such as MariaDB or Percona, this may be your only option.)

An example of checking the status of the cluster.

1. Open the MySQL client and check the status of the various parameters:

```
mysql> SET GLOBAL wsrep_cluster_address='<cluster address string>';
mysql> SHOW STATUS LIKE 'wsrep%';
```

You should see a status that looks something like this:

```
mysql> show status like 'wsrep%';
```

Variable_name	Value
wsrep_local_state_uuid	111fc28b-1b05-11e1-0800-e00ec5a7c930
wsrep_protocol_version	1
wsrep_last_committed	0
wsrep_replicated	0
wsrep_replicated_bytes	0
wsrep_received	2
wsrep_received_bytes	134
wsrep_local_commits	0
wsrep_local_cert_failures	0
wsrep_local_bf_aborts	0
wsrep_local_replays	0
wsrep_local_send_queue	0
wsrep_local_send_queue_avg	0.000000
wsrep_local_recv_queue	0
wsrep_local_recv_queue_avg	0.000000
wsrep_flow_control_paused	0.000000
wsrep_flow_control_sent	0
wsrep_flow_control_recv	0
wsrep_cert_deps_distance	0.000000
wsrep_apply_oooe	0.000000
wsrep_apply_ool	0.000000
wsrep_apply_window	0.000000
wsrep_commit_oooe	0.000000
wsrep_commit_ool	0.000000
wsrep_commit_window	0.000000
wsrep_local_state	4
wsrep_local_state_comment	Synced (6)
wsrep_cert_index_size	0
wsrep_cluster_conf_id	1
wsrep_cluster_size	1
wsrep_cluster_state_uuid	111fc28b-1b05-11e1-0800-e00ec5a7c930
wsrep_cluster_status	Primary
wsrep_connected	ON
wsrep_local_index	0
wsrep_provider_name	Galera
wsrep_provider_vendor	Codership Oy
wsrep_provider_version	21.1.0(r86)
wsrep_ready	ON

38 rows in set (0.01 sec)

Galera monitoring scripts

(Coming soon)

Other ways to provide a highly available database

MySQL with Galera is by no means the only way to achieve database HA. MariaDB (<https://mariadb.org/>) and Percona (<http://www.percona.com/>) also work with Galera. You also

have the option to use PostgreSQL, which has its own replication, or another database HA option.

7. RabbitMQ

Table of Contents

Install RabbitMQ	36
Configure RabbitMQ	36
Configure OpenStack services to use RabbitMQ	37

RabbitMQ is the default AMQP server used by many OpenStack services. Making the RabbitMQ service highly available involves the following steps:

- Install RabbitMQ
- Configure RabbitMQ for HA queues
- Configure OpenStack services to use Rabbit HA queues

Install RabbitMQ

This setup has been tested with RabbitMQ 2.7.1.

On Ubuntu / Debian

RabbitMQ is packaged on both distros:

```
# apt-get install rabbitmq-server rabbitmq-plugins
```

[Official manual for installing RabbitMQ on Ubuntu / Debian](#)

On Fedora / RHEL

RabbitMQ is packaged on both distros:

```
# yum install erlang
```

[Official manual for installing RabbitMQ on Fedora / RHEL](#)

Configure RabbitMQ

Here we are building a cluster of RabbitMQ nodes to construct a RabbitMQ broker. Mirrored queues in RabbitMQ improve the availability of service since it will be resilient to failures. We have to consider that while exchanges and bindings will survive the loss of individual nodes, queues and their messages will not because a queue and its contents is located on one node. If we lose this node, we also lose the queue.

We consider that we run (at least) two RabbitMQ servers. To build a broker, we need to ensure that all nodes have the same erlang cookie file. To do so, stop RabbitMQ everywhere and copy the cookie from rabbit1 server to other server(s):


```
# scp /var/lib/rabbitmq/.erlang.cookie \  
root@rabbit2:/var/lib/rabbitmq/.erlang.cookie
```

Then, start RabbitMQ on nodes. If RabbitMQ fails to start, you can't continue to the next step.

Now, we are building the HA cluster. From rabbit2, run these commands:

```
# rabbitmqctl stop_app  
# rabbitmqctl join_cluster rabbit@rabbit1  
# rabbitmqctl start_app
```

To verify the cluster status:

```
# rabbitmqctl cluster_status  
  
Cluster status of node rabbit@rabbit2 ...  
[{nodes,[{disc,[rabbit@rabbit1]},{ram,[rabbit@rabbit2]}]},{running_nodes,  
[rabbit@rabbit2,rabbit@rabbit1]}]
```

If the cluster is working, you can now proceed to creating users and passwords for queues.

Note for RabbitMQ version 3

Queue mirroring is no longer controlled by the *x-ha-policy* argument when declaring a queue. OpenStack can continue to declare this argument, but it won't cause queues to be mirrored. We need to make sure that all queues (except those with auto-generated names) are mirrored across all running nodes:

```
# rabbitmqctl set_policy HA '^(?!amq\..)*' '{"ha-mode": "all"}'
```

[More information about High availability in RabbitMQ](#)

Configure OpenStack services to use RabbitMQ

We have to configure the OpenStack components to use at least two RabbitMQ nodes.

Do this configuration on all services using RabbitMQ:

RabbitMQ HA cluster host:port pairs:

```
rabbit_hosts=rabbit1:5672,rabbit2:5672
```

How frequently to retry connecting with RabbitMQ:

```
rabbit_retry_interval=1
```

How long to back-off for between retries when connecting to RabbitMQ:

```
rabbit_retry_backoff=2
```

Maximum retries with trying to connect to RabbitMQ (infinite by default):

```
rabbit_max_retries=0
```

Use durable queues in RabbitMQ:

```
rabbit_durable_queues=false
```

Use H/A queues in RabbitMQ (x-ha-policy: all):

```
rabbit_ha_queues=true
```

If you change the configuration from an old setup which did not use HA queues, you should interrupt the service:

```
# rabbitmqctl stop_app  
# rabbitmqctl reset  
# rabbitmqctl start_app
```

Services currently working with HA queues: OpenStack Compute, OpenStack Block Storage, OpenStack Networking, Telemetry.

8. HAproxy nodes

HAProxy is a very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It is particularly suited for web sites crawling under very high loads while needing persistence or Layer 7 processing. Supporting tens of thousands of connections is clearly realistic with today's hardware.

For installing HAProxy on your nodes, you should consider its [official documentation](#). Also, you have to consider that this service should not be a single point of failure, so you need at least two nodes running HAProxy.

Here is an example for HAProxy configuration file:

```
global
    chroot    /var/lib/haproxy
    daemon
    group    haproxy
    maxconn  4000
    pidfile  /var/run/haproxy.pid
    user    haproxy

defaults
    log    global
    maxconn  8000
    option  redispatch
    retries  3
    timeout http-request 10s
    timeout queue 1m
    timeout connect 10s
    timeout client 1m
    timeout server 1m
    timeout check 10s

listen dashboard_cluster
    bind <Virtual IP>:443
    balance source
    option tcpka
    option httpchk
    option tcplog
    server controller1 10.0.0.1:443 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:443 check inter 2000 rise 2 fall 5

listen galera_cluster
    bind <Virtual IP>:3306
    balance source
    option httpchk
    server controller1 10.0.0.4:3306 check port 9200 inter 2000 rise 2 fall 5
    server controller2 10.0.0.5:3306 check port 9200 inter 2000 rise 2 fall 5
    server controller3 10.0.0.6:3306 check port 9200 inter 2000 rise 2 fall 5

listen glance_api_cluster
    bind <Virtual IP>:9292
    balance source
    option tcpka
    option httpchk
    option tcplog
    server controller1 10.0.0.1:9292 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:9292 check inter 2000 rise 2 fall 5
```

```
listen glance_registry_cluster
  bind <Virtual IP>:9191
  balance source
  option tcpka
  option tcplog
  server controller1 10.0.0.1:9191 check inter 2000 rise 2 fall 5
  server controller2 10.0.0.2:9191 check inter 2000 rise 2 fall 5

listen keystone_admin_cluster
  bind <Virtual IP>:35357
  balance source
  option tcpka
  option httpchk
  option tcplog
  server controller1 10.0.0.1:35357 check inter 2000 rise 2 fall 5
  server controller2 10.0.0.2:35357 check inter 2000 rise 2 fall 5

listen keystone_public_internal_cluster
  bind <Virtual IP>:5000
  balance source
  option tcpka
  option httpchk
  option tcplog
  server controller1 10.0.0.1:5000 check inter 2000 rise 2 fall 5
  server controller2 10.0.0.2:5000 check inter 2000 rise 2 fall 5

listen nova_ec2_api_cluster
  bind <Virtual IP>:8773
  balance source
  option tcpka
  option tcplog
  server controller1 10.0.0.1:8773 check inter 2000 rise 2 fall 5
  server controller2 10.0.0.2:8773 check inter 2000 rise 2 fall 5

listen nova_compute_api_cluster
  bind <Virtual IP>:8774
  balance source
  option tcpka
  option httpchk
  option tcplog
  server controller1 10.0.0.1:8774 check inter 2000 rise 2 fall 5
  server controller2 10.0.0.2:8774 check inter 2000 rise 2 fall 5

listen nova_metadata_api_cluster
  bind <Virtual IP>:8775
  balance source
  option tcpka
  option tcplog
  server controller1 10.0.0.1:8775 check inter 2000 rise 2 fall 5
  server controller2 10.0.0.2:8775 check inter 2000 rise 2 fall 5

listen cinder_api_cluster
  bind <Virtual IP>:8776
  balance source
  option tcpka
  option httpchk
  option tcplog
  server controller1 10.0.0.1:8776 check inter 2000 rise 2 fall 5
  server controller2 10.0.0.2:8776 check inter 2000 rise 2 fall 5
```

```
listen ceilometer_api_cluster
    bind <Virtual IP>:8777
    balance source
    option tcpka
    option httpchk
    option tcplog
    server controller1 10.0.0.1:8774 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8774 check inter 2000 rise 2 fall 5

listen spice_cluster
    bind <Virtual IP>:6082
    balance source
    option tcpka
    option tcplog
    server controller1 10.0.0.1:6080 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:6080 check inter 2000 rise 2 fall 5

listen neutron_api_cluster
    bind <Virtual IP>:9696
    balance source
    option tcpka
    option httpchk
    option tcplog
    server controller1 10.0.0.1:9696 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:9696 check inter 2000 rise 2 fall 5

listen swift_proxy_cluster
    bind <Virtual IP>:8080
    balance source
    option tcplog
    option tcpka
    server controller1 10.0.0.1:8080 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8080 check inter 2000 rise 2 fall 5
```

After each change of this file, you should restart HAproxy.

9. OpenStack controller nodes

Table of Contents

Run OpenStack API and schedulers	42
Memcached	43

OpenStack controller nodes contain:

- All OpenStack API services
- All OpenStack schedulers
- Memcached service

Run OpenStack API and schedulers

API services

All OpenStack projects have an API service for controlling all the resources in the Cloud. In Active / Active mode, the most common setup is to scale-out these services on at least two nodes and use load balancing and virtual IP (with HAproxy & Keepalived in this setup).

Configure API OpenStack services

To configure our Cloud using Highly available and scalable API services, we need to ensure that:

- You use Virtual IP when configuring OpenStack Identity endpoints.
- All OpenStack configuration files should refer to Virtual IP.

In case of failure

The monitor check is quite simple since it just establishes a TCP connection to the API port. Comparing to the Active / Passive mode using Corosync & Resources Agents, we don't check if the service is actually running). That's why all OpenStack API should be monitored by another tool, for example Nagios, with the goal to detect failures in the Cloud Framework infrastructure.

Schedulers

OpenStack schedulers are used to determine how to dispatch compute, network and volume requests. The most common setup is to use RabbitMQ as messaging system already documented in this guide. Those services are connected to the messaging backend and can scale-out:

- nova-scheduler

- nova-conductor
- cinder-scheduler
- neutron-server
- ceilometer-collector
- heat-engine

Please refer to the RabbitMQ section for configure these services with multiple messaging servers.

Memcached

Most of OpenStack services use an application to offer persistence and store ephemeral data (like tokens). Memcached is one of them and can scale-out easily without specific trick.

To install and configure it, read the [official documentation](#).

Memory caching is managed by oslo-incubator so the way to use multiple memcached servers is the same for all projects.

Example with two hosts:

```
memcached_servers = controller1:11211,controller2:11211
```

By default, controller1 handles the caching service but if the host goes down, controller2 does the job. For more information about memcached installation, see the OpenStack Cloud Administrator Guide.

10. OpenStack network nodes

Table of Contents

Run neutron DHCP agent	44
Run neutron L3 agent	44
Run neutron metadata agent	44

OpenStack network nodes contain:

- neutron DHCP agent
- neutron L2 agent
- Neutron L3 agent
- neutron metadata agent
- neutron lbaas agent



Note

The neutron L2 agent does not need to be highly available. It has to be installed on each Data Forwarding Node and controls the virtual networking drivers as Open vSwitch or Linux Bridge. One L2 agent runs per node and controls its virtual interfaces. That's why it cannot be distributed and highly available.

Run neutron DHCP agent

OpenStack Networking service has a scheduler that lets you run multiple agents across nodes. Also, the DHCP agent can be natively highly available. For details, see [OpenStack Configuration Reference](#).

Run neutron L3 agent

The neutron L3 agent is scalable thanks to the scheduler that allows distribution of virtual routers across multiple nodes. But there is no native feature to make these routers highly available. At this time, the Active / Passive solution exists to run the Neutron L3 agent in failover mode with Pacemaker. See the Active / Passive section of this guide.

Run neutron metadata agent

There is no native feature to make this service highly available. At this time, the Active / Passive solution exists to run the neutron metadata agent in failover mode with Pacemaker. See the Active / Passive section of this guide.

Appendix A. Community support

Table of Contents

Documentation	45
ask.openstack.org	46
OpenStack mailing lists	46
The OpenStack wiki	47
The Launchpad Bugs area	47
The OpenStack IRC channel	48
Documentation feedback	48
OpenStack distribution packages	48

The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support, and troubleshoot your installations.

Documentation

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at [OpenStack Documentation Mailing List](#), or [report a bug](#).

The following books explain how to install an OpenStack cloud and its associated components:

- [Installation Guide for Debian 7.0](#)
- [Installation Guide for openSUSE and SUSE Linux Enterprise Server](#)
- [Installation Guide for Red Hat Enterprise Linux, CentOS, and Fedora](#)
- [Installation Guide for Ubuntu 14.04 \(LTS\)](#)

The following books explain how to configure and run an OpenStack cloud:

- [Architecture Design Guide](#)
- [Cloud Administrator Guide](#)
- [Configuration Reference](#)
- [Operations Guide](#)
- [High Availability Guide](#)
- [Security Guide](#)

- [Virtual Machine Image Guide](#)

The following books explain how to use the OpenStack dashboard and command-line clients:

- [API Quick Start](#)
- [End User Guide](#)
- [Admin User Guide](#)
- [Command-Line Interface Reference](#)

The following documentation provides reference and guidance information for the OpenStack APIs:

- [OpenStack API Complete Reference \(HTML\)](#)
- [API Complete Reference \(PDF\)](#)
- [OpenStack Block Storage Service API v2 Reference](#)
- [OpenStack Compute API v2 and Extensions Reference](#)
- [OpenStack Identity Service API v2.0 Reference](#)
- [OpenStack Image Service API v2 Reference](#)
- [OpenStack Networking API v2.0 Reference](#)
- [OpenStack Object Storage API v1 Reference](#)

The [Training Guides](#) offer software training for cloud administration and management.

ask.openstack.org

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the <http://ask.openstack.org> site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to <http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack>. You might be interested in the other mailing lists for specific projects or development, which you can find [on the wiki](#). A description of all mailing lists is available at <http://wiki.openstack.org/MailingLists>.

The OpenStack wiki

The [OpenStack wiki](#) contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or nova, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at <https://launchpad.net/+login>. You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

- Give a clear, concise summary.
- Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.
- Be sure to include the software and package versions that you are using, especially if you are using a development branch, such as, "Juno release" vs `git commit bc79c3ecc55929bac585d04a03475b72e06a3208`.
- Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

- [Bugs: OpenStack Block Storage \(cinder\)](#)
- [Bugs: OpenStack Compute \(nova\)](#)
- [Bugs: OpenStack Dashboard \(horizon\)](#)
- [Bugs: OpenStack Identity \(keystone\)](#)
- [Bugs: OpenStack Image Service \(glance\)](#)
- [Bugs: OpenStack Networking \(neutron\)](#)
- [Bugs: OpenStack Object Storage \(swift\)](#)
- [Bugs: Bare Metal \(ironic\)](#)
- [Bugs: Data Processing Service \(sahara\)](#)
- [Bugs: Database Service \(trove\)](#)

- [Bugs: Orchestration \(heat\)](#)
- [Bugs: Telemetry \(ceilometer\)](#)
- [Bugs: Queue Service \(marconi\)](#)
- [Bugs: OpenStack API Documentation \(developer.openstack.org\)](#)
- [Bugs: OpenStack Documentation \(docs.openstack.org\)](#)

The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to <http://webchat.freenode.net/>. You can also use Colloquy (Mac OS X, <http://colloquy.info/>), mIRC (Windows, <http://www.mirc.com/>), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at <http://paste.openstack.org>. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is #openstack on `irc.freenode.net`. You can find a list of all OpenStack IRC channels at <https://wiki.openstack.org/wiki/IRC>.

Documentation feedback

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at [OpenStack Documentation Mailing List](#), or [report a bug](#).

OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- **Debian:** <http://wiki.debian.org/OpenStack>
- **CentOS, Fedora, and Red Hat Enterprise Linux:** <http://openstack.redhat.com/>
- **openSUSE and SUSE Linux Enterprise Server:** <http://en.opensuse.org/Portal:OpenStack>
- **Ubuntu:** <https://wiki.ubuntu.com/ServerTeam/CloudArchive>