High availability apps in the IBM Cloud

How to protect a cloud-enabled, production-grade application against a failure on any node

Skill Level: Intermediate

Dima Rekesh (dima@us.ibm.com)
Senior Technical Staff Member
IBM

Alan Robertson (alanr@us.ibm.com)
High Availability Consultant
IBM

10 Jan 2011

The new features of the IBM® Cloud enable application developers and architects to eliminate single points of failures in applications. This article provides a detailed guide on those features. It includes a discussion of the approach the IBM Cloud takes (added support for virtual IP addresses); how to prepare your cloud instances to take advantage of this feature; how to set up a highly available website; and how to test that site.

The IBM® Smart Business Development and Test on the IBM Cloud is a dynamically provisioned and scaled elastic environment that provides enterprise customers with everything they need to develop, test, and host applications. It includes a web portal to configure and manage the cloud resources, software images of IBM products that jump-start development and test efforts, and APIs that enable users to control cloud resources and software programmatically. And IBM teams have been adding new features since the debut of the IBM Cloud that are designed to provide additional flexibility and resiliency.

This greater agility and much improved elasticity — which helps adjust your application topology to the demands of the business in real time — comes with a tradeoff: A decrease in compatibility with cloud environments of such features as

high availability (HA).

High availability, the requirement to protect a production-grade application against a failure of any node, isn't a new concept by any standard; many software products address this challenge. But most of these products are, by and large, not compatible with the cloud; most public cloud providers do not provide the required functionality.

With that in mind, customers need to supplement their cloud deployments with HA constructs that exist outside the cloud. In this article, you will see what IBM has done with its cloud to address this issue and how you can tap into that feature:

- We'll discuss the approach the IBM Cloud takes (added support for virtual IP addresses).
- We'll show you how to prepare your cloud instances to take advantage of this feature.
- In an example, we'll show you how to set up a highly available website.
- And we'll show you how to test that site.

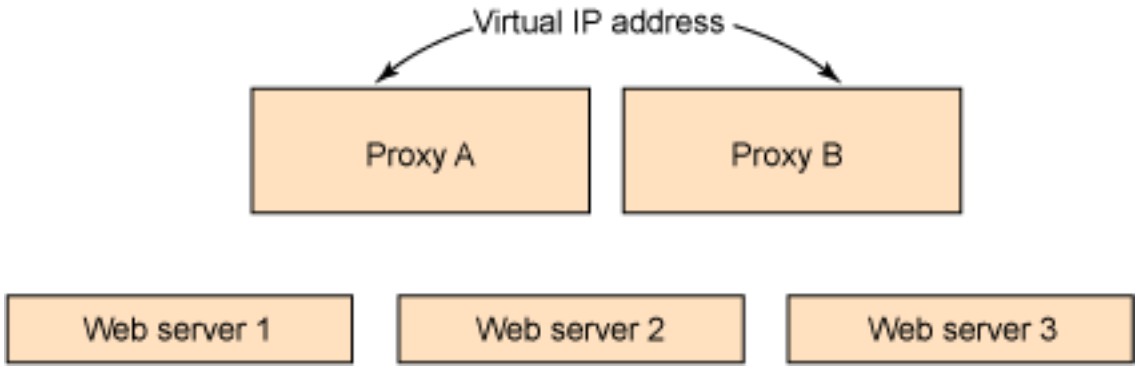
Secure, expedient virtual IP address support

To securely and expediently address the concerns of HA, IBM Cloud engineers have added support for virtual IP addresses (vIPs) on the IBM Cloud virtual instances. Although there are a variety of methods for providing highly available services (see Resources), the most popular and robust of these methods is to use virtual IPs.

In addition to a regular *static* IP address (each instance gets one when it is provisioned and it never changes), an instance can dynamically assume one or several additional virtual IP addresses. Because it is the application code running in the instances that controls the association between the vIPs and instances, the application topology can adjust to a node failure very quickly, in the sub-second range.

Consider this simple example. There is a pair of virtual machines *VM A* and *VM B* with static IP addresses of 192.168.1.101 and 192.168.1.102 respectively. In addition, both VMs are configured to allow the dynamic assignment of the vIP 192.168.1.10 (Figure 1).

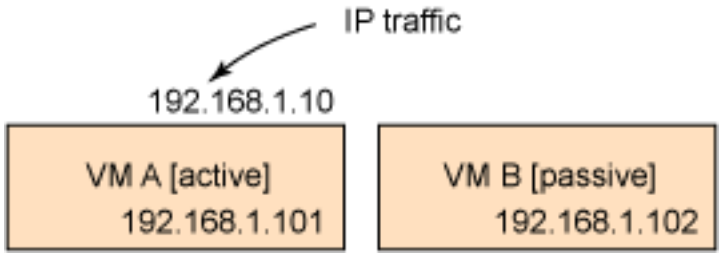
Figure 1. VM A and B have individual static IPs and can assume a vIP



In this configuration, static IP addresses are used to administer and maintain the instances. In contrast, the virtual IP address is exposed to the clients as the server IP address.

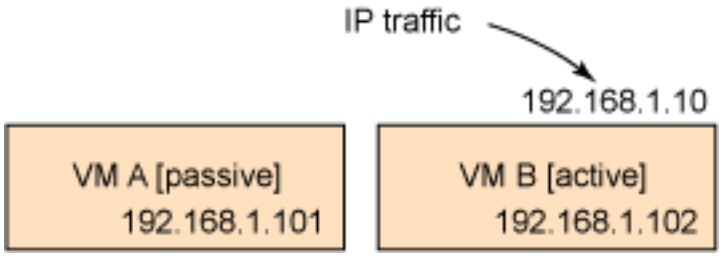
Initially, VM A holds the vIP and consequently, handles all of the service traffic. VM B is up and running, yet does not have the vIP and acts as a warm stand-by (Figure 2).

Figure 2. Active/passive configuration: VM A holds the vIP and services all traffic; VM B acts as a passive stand-by



Let us imagine now that the application detects a problem with VM A. Perhaps it is experiencing a slowdown or a failure. The application decides to transfer the vIP control to the other VM. Now, VM B serves all the traffic while VM A, once repaired, relinquishes the vIP and becomes the warm stand-by (Figure 3).

Figure 3. Active/passive configuration: VMs swap roles



Because the virtual IP address can be transferred between the VMs in the sub-second range, with careful programming you can significantly reduce or practically eliminate any potential outage, moving the IP address at the first sign of trouble. As simple as this sounds, this is a proven method and one which the IBM Cloud environment supports.

Now let's look at preparing your instances to take advantage of this.

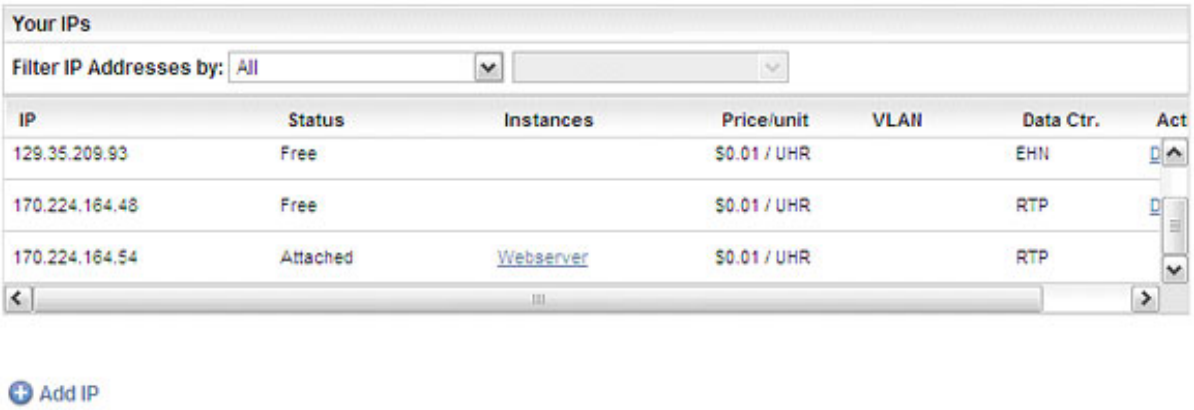
Preparing your instances for HA

Following are three steps to prepare an instance in the IBM Cloud to take advantage of the virtual IP address method of providing high availability.

Get a free, unattached reserved IP address

Make sure that you have a free, unattached reserved IP address. Log into the IBM Cloud portal and click the **Account Tab**. Scroll down to the **Your IPs** section to see a list of reserved IPs. Make sure there is at least one free address in the data center of your choice; if not, click **Add IP** and wait for new IP address to become available (Figure 4).

Figure 4. The reserved IP address panel of the IBM Cloud portal



Provision your instances

Now you simply provision your instances. Select the image of your choice and on the screen that follows, click the **Add IP** button, near the **Virtual IP** section, and select one of the free IP addresses (Figure 5).

Figure 5. This IBM Cloud instance provision dialog lets you assign a vIP to your instance

Note that the virtual IP address needs to be in the same data center as the instance you're assigning to it. Click **Close** and complete the provisioning request.

Once provisioned, each of the instances will initially assume its static IP address only. Note that the static IP address of an instance is bound to the eth0 network interface while the virtual IPs, if chosen during provisioning, get assigned to interfaces eth0, eth1, etc., in the order they are selected.

Associate a vIP to an instance

At any time during the life cycle of an instance, you can use `sudo /sbin/ifup <interface name>` to associate a virtual IP address with an instance. The

command `sudo /sbin/ifdown <interface name>` will dissociate it.

By the way, only one instance at a time can own a virtual IP address. The IBM Cloud does not support multicasting, so assigning an IP address to multiple VMs would only lead to collisions.

For example, let's say that an instance received a system-assigned static IP address of 170.224.163.231 and a virtual IP address of 170.224.163.161. /etc/sysconfig/network-scripts/ifcfg-eth0 will tell you more about the configuration of the primary interface:

```
DEVICE=eth0
TYPE=Ethernet
BOOTPROTO=static
HWADDR=DE:AD:BE:A7:13:52
IPADDR=170.224.163.231
NETMASK=255.255.240.0
ONBOOT=yes
GATEWAY=170.224.160.1
```

Compare that with the contents of the /etc/sysconfig/network-scripts/ifcfg-eth1 file:

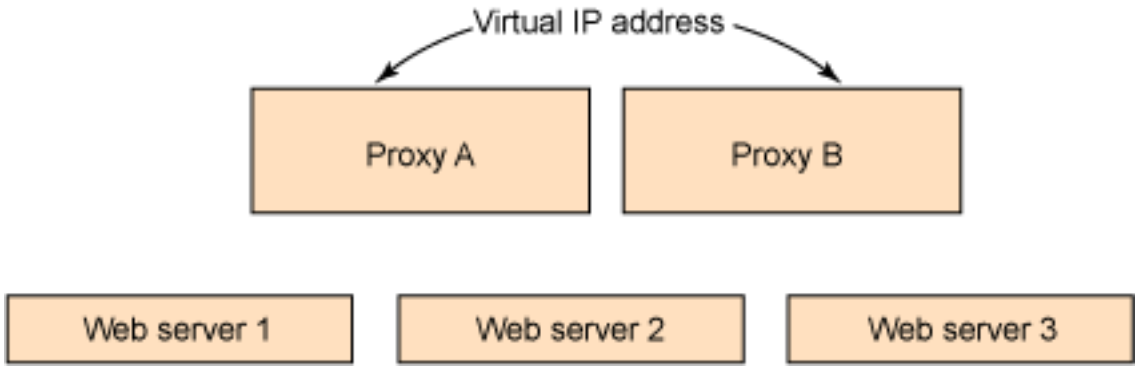
```
DEVICE=eth1
TYPE=Ethernet
BOOTPROTO=static
HWADDR=DE:AD:BE:C8:25:20
IPADDR=170.224.163.161
NETMASK=255.255.240.0
ONBOOT=no
```

Issue `sudo /sbin/ifup eth1` to associate 170.224.163.161 with the instance and `sudo /ifdown eth1` to dissociate it. If you want to get this IP address at boot time, you simply change the `ONBOOT` clause to `yes` in the /etc/sysconfig/network-scripts/ifcfg-eth1 file.

Setting up an HA website

In this example, we're going to put together a simple topology consisting of two proxy servers configured in the HA active/passive configuration and three web/application servers (as shown in Figure 6):

Figure 6. A simple topology where a pair of reverse proxies, set up in the active/passive configuration, sprays traffic across three web servers



The following software configuration is used for this example:

- The proxies are running nginx.
- High availability for the proxies are provided by Linux HA and the Pacemaker software.
- The web servers are running Apache (pre-installed as part of the Base OS image).
- All IBM Cloud instances use the base 64-bit bronze RHEL 5.4 image.

1. Using the IBM Cloud portal, you need to provision all five instances. Be sure to reserve and assign a vIP to the pair of the proxy nodes as described earlier in this article.
2. Configure the proxy instances. All the downloads are available from the Resources section.
3. You need a few prerequisites:

```
sudo yum -y install glib2-devel libxml2 libxml2-devel bzip2 bzip2-devel pcre-devel
```

4. Install nginx (similar to this):

```
wget http://nginx.org/download/nginx-0.8.53.tar.gz
tar -xvfz nginx-0.8.53.tar.gz
cd nginx-0.8.53
./configure -with-http_ssl_module
make
sudo make install
```

5. To add the basic proxy rules to nginx, edit the /usr/local/nginx/conf/nginx.conf file and under the server directive, add the following (use the actual IP addresses of the appservers):

```
upstream appservers {
   ip_hash;
   server appserver1_ip;
   server appserver2_ip;
   server appserver3_ip;
            }
```

6. It's important to have consistent time on all servers, so make sure ntpd starts automatically:

```
sudo /sbin/chkconfig --level 2345 ntpd on
sudo /sbin/service ntpd start
```

7. Install the Pacemaker and Linux-HA packages. Make sure the Clusterlabs (Pacemaker) and EPEL repositories are available. For RHEL 5.4, issue these two commands:

```
sudo rpm -Uvh \
http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-4.noarch.rpm
sudo wget -O \
/etc/yum.repos.d/pacemaker.repo http://clusterlabs.org/rpm/epel-5/clusterlabs.repo
```

Then install Pacemaker and required packages:

```
sudo yum install -y pacemaker heartbeat
```

8. You need a compatible script to manage the nginx server, so install the nginx resource agent. This script has been submitted to the Linux-HA project, and will eventually be supplied automatically. Until that happens, use this command:

```
sudo wget -O /usr/lib/ocf/heartbeat/resource.d/nginx \
http://lists.linux-ha.org/pipermail/linux-ha-dev/attachments/20101206/3c141ea6/
      attachment.txt
```

9. You must provide the proper configuration for the Heartbeat software from the Linux-HA project. For the IBM Cloud, it is necessary to configure all communication to take place using unicast transmission (the sending of messages to a single network destination identified by a unique address). That is why you installed the Heartbeat communication layer.
To configure the Heartbeat communication layer, there are three different files to configure: /etc/ha.d/ha.cf, /etc/ha.d/authkeys, and /etc/logd.cf.

   - All machines should have an exact copy of the /etc/ha.d/ha.cf file. To configure, add these lines:

```
use_logd yes
ucast eth0 cluster1-fixed-ip
ucast eth0 cluster2-fixed-ip # repeat for all cluster nodes
autojoin any
crm on
```

It is worth noting that you must use the `ucast` communication method when running the IBM Cloud since it does not support broadcast or multicast. This prevents you from using Corosync as your communication layer; it requires multicast or broadcast. The "fixed IP" addresses that have to be included are the real (not virtual) addresses of the servers. Put one line for each server.

- Each machine needs a copy of the /etc/ha.d/authkeys file too; it must be mode 0600. Use this method to generate the first copy, then copy this file to all the other nodes in the cluster:

```
cat <<-!AUTH >/etc/ha.d/authkeys
auth 1
        1 sha1 'dd if=/dev/urandom count=4 2>/dev/null | openssl dgst -sha1`
        !AUTH
```

- Last, configure the /etc/ha.d/logd.cf with this line: `logfacility daemon`.

10. Since it is simpler to configure Pacemaker after it is running, we'll defer that configuration until later.

11. Make sure the heartbeat starts automatically with this code:

```
sudo /sbin/chkconfig --levels 345 heartbeat on
```

12. The initial firewall configuration is a bit too restrictive, so add a few rules to /etc/sysconfig/iptables:

```
# allow pings
-A INPUT -p icmp --icmp-type any -j ACCEPT
# on the virtual IP address, allow only ports 80 and 443.
-A INPUT -d virtual_ip -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -d virtual_ip -m tcp -p tcp --dport 443 -j ACCEPT
-A INPUT -d virtual_ip  -j REJECT --reject-with icmp-host-prohibited

# allow ssh on the service IPs
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT

# allow heartbeat port
-A INPUT -p udp -m udp --dport 694 -j ACCEPT
```

13. Once the edits are completed, restart the firewall:

```
sudo /sbin/service iptables restart
```

.

14. The web servers require a little configuration attention. First, make sure that Apache starts automatically:

```
sudo /sbin/chkconfig -level 345 httpd on
sudo /sbin/service httpd start
```

Now, open ports 80 and 443 in the /etc/sysconfig/iptables:

```
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
```

Once you've done that, restart the firewall: `sudo /sbin/service iptables restart`. Make sure ntpd starts automatically on these nodes as well:

```
sudo /sbin/chkconfig --level 2345 ntpd on
sudo /sbin/service ntpd start
```

You should be all set. Now to test the system.

Testing the system

To test the system:

1. Start Heartbeat (which will start Pacemaker) on every node in the cluster.
2. Configure Pacemaker using the `crm` command.
3. Verify that things were started correctly.
4. Test the configuration. Perform some basic failover resource transfers.

Start Heartbeat

To start Heartbeat, issue a `service heartbeat start` command. Once you start it you should see a large number of processes that look similar to this:

```
ha_logd: read process
ha_logd: write process
heartbeat: master control process
heartbeat: FIFO reader
heartbeat: write: ucast some-ip-address
```

```
heartbeat: read: ucast some-ip-address
/usr/lib/heartbeat/ccm
/usr/lib/heartbeat/cib
/usr/lib/heartbeat/lrmd -r
/usr/lib/heartbeat/stonithd
/usr/lib/heartbeat/attrd
/usr/lib/heartbeat/crmd
/usr/lib/heartbeat/pengine
```

When you see those processes, you know that Heartbeat and Pacemaker are running.

Configure Pacemaker

To configure Pacemaker, edit the live configuration using the `crm configure edit` command, which opens a text editor. Insert the following lines in the configuration file just before the `property` statement at the bottom of the file:

```
primitive nginx-primitive ocf:heartbeat:nginx \
 op monitor interval="5s" timeout="120s" \
 OCF_CHECK_LEVEL="0" \
 op monitor interval="10s" timeout="120s" \
 OCF_CHECK_LEVEL="10" \
 params configfile="/etc/nginx/stci.d/nginx.conf"
primitive nginx-ipaddr ocf:heartbeat:IPaddr2 \
 op monitor interval="5s" timeout="120s"\
 params ip="NGINX-IP-ADDRESS"
primitive NFS-server lsb:nfs-kernel-server
primitive NFS-ipaddr ocf:heartbeat:IPaddr2 \
 op monitor interval="5s" timeout="120s"\
 params ip="NFS-IP-ADDRESS"
group nginx-group nginx-primitive nginx-ipaddr
group NFS-group NFS-server NFS-ipaddr
clone nginx-clone nginx-group \
        meta clone-max="1"
```

Verify startup

After you save the Pacemaker configuration, the system should have started your resources and propagated this new configuration to all the nodes in the system. This is the exciting part. If you check the system logs, you should see messages about services being started and so on. To see the HA system status, run the `sudo crm_mon` command. This shows you what resources are running, what nodes are up or down in the cluster and so on.

Test the configuration

The `crm_mon` command showed you which node is runing nginx. There are several approaches to testing this configuration; we suggest trying them all. ***A cluster which is not thoroughly tested will not be highly available***:

- Issue a `sudo service heartbeat stop` on the active node; see

where things moved, then restart it.

- Perform a `sudo reboot -f` on the active node, then watch (from another node) where the resources move to.
- Issue a `sudo crm node standby` command on the active node, see where the service went, then issue a `sudo crm node online`.

This completes the process of setting up and testing your nginx proxy service. Linux also provides a built-in kernel-level load balancer which can be used and made highly available in place of nginx.

In conclusion

The same set of techniques introduced in this article can be used to make nearly any service highly available in the cloud; for example, a highly available NFS server in the IBM Cloud that includes replicating data between cloud virtual servers.

This article detailed the approach the IBM Cloud takes for high availability (added support for virtual IP addresses); explained how to prepare your cloud instances to take advantage of the vIP feature; provided an example of how to set up a highly available website; and described how to test that site.

Resources

Learn

- There are methods other than virtual IP addresses that can provide HA services; see how availability, automation, managing systems, open source, and related topics come together at Managing Computers with Automation.
- nginx ("engine x") is an HTTP and reverse proxy server.
- In the developerWorks cloud developer resources, discover and share knowledge and experience of application and services developers building their projects for cloud deployment.
- Follow developerWorks on Twitter.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Access the IBM Smart Business Development and Test on the IBM Cloud.

Get products and technologies

- HA for the proxies in this article are provided by:
  - The Linux-HA project [Learn] maintains a set of building blocks for high availability cluster systems [Download Heartbeat | Download Cluster Glue | Download Resource Agents].
  - To be useful to users, the Heartbeat daemon needs to be combined with a cluster resource manager (CRM) — Pacemaker .
  - Extra Packages for Enterprise Linux (EPEL) is a volunteer-based community effort from the Fedora project to create a repository of high-quality add-on packages for Red Hat Enterprise Linux (RHEL) and its compatible spinoffs [Learn | Download].

Discuss

- Join a cloud computing group in the developerWorks community.
- Read all the great cloud blogs on from developerWorks community members.
- Join the developerWorks community, a professional network and unified set of community tools for connecting, sharing, and collaborating.

About the authors

Dima Rekesh

Dima is part of the strategic IBM Cloud Computing Enterprise Initiatives team. In the past, he worked on a number of large-scale Web application platforms, emerging technologies, as well as leadership data centers and green data center strategies.

Alan Robertson

Alan founded the open source Linux-HA project and is an internationally known expert on high availability systems. He is a frequently requested speaker on high availability and Linux.