Load Balancing for a WIS2 Global Broker

Table of Contents

The documentation *Deploying a WIS2 Global Broker* details the software component required to run a WIS2 Global Broker. In addition to this, and to provide the required redundancy for the configuration, a load balancing mechanism must be available. The load balancing will ensure that the three brokers will be equally used and, in case of a failure of one of those, no incoming connection will try to use the failed resource.

There are many solutions on the market for this kind of services:

- If deployed in an openstack environment, it is possible to use *Octavia* (part of the openstack distribution) to load balance the traffic
- A commercial product like F5 Big IP can also be used
- An open source solution based on HAProxy is also possible

1. Required features of the Load Balancer

Whatever the solution used, and to comply with the *Deploying a WIS2 Global Broker*, it is anticipated that the load balancing tool used will:

- Be itself redundant! The Load Balancing shouldn't be a single point of failure.
- Be reachable using ports **443** (for MQTT over WSS access) and **8883** (for MQTTS access)
- Be protected by a redundant layer of firewalls preventing access, from the Internet, to *all* ports except those mentioned above. In addition, the firewall will NAT (Network Address Translation) the public IP address(es) of the Global Broker to private IP (using RFC1918) that will be used on the Load Balancer.
- Be terminating the **SSL** connection
- Be using a valid certificate
- Be relaying incoming connections on port **443** to port **8083** and on port **8883** to port **1883**
- Be using a *Health Check* mechanism to detect failure on the wbroker0x hosts. Typically, the Load Balancer will verify, on a regular basis, that each wbroker0x host is reachable on port **1883** and **8083**

It is beyond the scope of this document to describe the implementation of all these requirements for all kind of load balancers.

The rest of this document will present a possible solution using:

- two Linux based VMs running Debian 12 distribution
- virtual IP address management mechanism using pcs, pacemaker and corosync
- HAProxy as a load balancer software.

2. A HAProxy based setup

2.1. Two Virtual Machines or physical hosts

Two Virtual Machines (VMs) or physical servers are required. For the rest of the guide, we will consider only the VMs option. Using physical hosts is also possible and shouldn't differ much from using the VMs. The VMs can be deployed on premises or in the cloud. They must be collocated with the Global Broker environment.

As a convention, they will be called wlb01 and wlb02.

The suggested configuration for each VM is:

```
CPU: 8 cores
RAM: 64 GBytes
Disk: 80 GBytes
```

On those VMs, a linux distribution must be deployed. Almost any distribution can be used. The rest of the guide will be based on a Debian 12 installation.

2.2. Additional software packages

The load balancing will effectively have two main features:

- Sharing in an active/passive mode a virtual IP address between wlb01 and wlb02. For this, the required packages are pcs, corosync and pacemaker. Those tools can do much more than just sharing a virtual IP address. They are providing a set of feature to create a redundant linux environment, e.g. synchronizing a disk volume… Here only one feature will be used.
- Proxying incoming connections from the virtual IP to the real IP addresses of the wbroker0x, while terminating the *SSL* connection. HAProxy will be used for this feature.

In order to install the packages, as `root` or using `sudo`, on both wlb01 and wlb02, run:

```
apt install pcs corosync pacemaker haproxy
```

At the time of writing (Feb. 2024), the versions of the debian packages for the tools are:

- HAProxy: Version: 2.6.12-1+deb12u1
- pcs: Version: 0.11.5-1+deb12u1
- corosync: Version: 3.1.7-1
- pacemaker: Version: 2.1.5-1+deb12u1

Depending on evolution of those packages, the syntax of the setup and commands may have to be updated.

3. Shared IP address configuration

After having installed the required packages, the following configuration can be used to add a virtual IP, to be shared between wlb01 and wlb02.

As `root` or using `sudo`, on **both** wlb01 and wlb02, configure a password for the user hacluster that has been added during the package installation, start pcs and destroy the default cluster:

```
systemctl enable pcsd
passwd hacluster
pcs cluster destroy
```

The password can be anything and must be the same on wlb01 and wlb02.

Now, only on **one** host, say wlb01:

```
pcs host auth wlb01 wlb02
```

You will have to enter the password for hacluster. If successful, the output will be:

```
wlb01: Authorized
wlb02: Authorized
```

Now configure the cluster between the two wblb0x:

```
pcs cluster setup ha_cluster wlb01 wlb02
```

`ha_cluster` is a just an example. Any string will do.

The result will look like this, if successful

```
No addresses specified for host 'wlb01', using 'wlb01'
No addresses specified for host 'wlb02', using 'wlb02'
Destroying cluster on hosts: 'wlb01', 'wlb02'...
wlb01: Successfully destroyed cluster
wlb02: Successfully destroyed cluster
Requesting remove 'pcsd settings' from 'wlb01', 'wlb02'
wlb01: successful removal of the file 'pcsd settings'
wlb02: successful removal of the file 'pcsd settings'
Sending 'corosync authkey', 'pacemaker authkey' to 'wlb01', 'wlb02'
wlb01: successful distribution of the file 'corosync authkey'
wlb01: successful distribution of the file 'pacemaker authkey'
wlb02: successful distribution of the file 'corosync authkey'
wlb02: successful distribution of the file 'pacemaker authkey'
Sending 'corosync.conf' to 'wlb01', 'wlb02'
```

```
wlb01: successful distribution of the file 'corosync.conf'
wlb02: successful distribution of the file 'corosync.conf'
Cluster has been successfully set up.
```

Then, start the services for the cluster:

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

Will give:

```
wlb01: Starting Cluster...
wlb02: Starting Cluster...
```

As the cluster should start at reboot, enable the services:

```
pcs cluster enable --all
```

Will output:

```
wlb01: Cluster Enabled
wlb02: Cluster Enabled
```

To verify that all is correct on pacemaker:

```
pcs cluster status
```

As a result:

```
Cluster Status:
  Status of pacemakerd: 'Pacemaker is running' (last updated 2023-07-26 19:43:40
-05:00)
  Cluster Summary:
   * Stack: corosync
   * Current DC: wlb01 (version 2.1.5-a3f44794f94) - partition with quorum
   * Last updated: Wed Jul 26 19:43:41 2023
   * Last change:  Wed Jul 26 19:43:35 2023 by hacluster via crmd on wlb01
   * 2 nodes configured
   * 0 resource instances configured
  Node List:
   * Online: [ wlb01 wlb02 ]

PCSD Status:
  wlb02: Online
```

```
  wlb01: Online
```

And for corosync:

```
pcs status corosync
```

Will have the following output:

```
Membership information
----------------------
    Nodeid      Votes Name
         1          1 wlb01 (local)
         2          1 wlb02
```

To avoid a split brain situation (it is the term used when, in a cluster, two nodes can't communicate even if both are working normally, with the risk of each trying to be the active node), it is suggested to create a cluster with three nodes. Nevertheless, for a simple configuration, it is not required. To avoid the clustering mechanism to complain about this lack of quorum feature, add:

```
pcs property set stonith-enabled=false
pcs property set no-quorum-policy=ignore
```

With this configuration, wlb01 and wlb02 are in a cluster of two nodes. In the cluster, resources will be shared between the nodes. Here, only one IP address is needed as a resource.

```
pcs resource create ip_gb ocf:heartbeat:IPaddr2 ip=192.168.0.100 nic=ens3
cidr_netmask=32 op monitor interval=30s
```

- `ip_gb` will be the same of the shared resource in the cluster. Any string can be used.
- Replace `192.168.0.100` with the address of the shared IP address
- Replace `ens3` with the name of the Ethernet interface of the VM. This name can be displayed by running `ip addr`

If successfully created, `ping 192.168.100` will answer positively.

```
pcs status resources
```

will display the list of resources and the node owning the resource:

```
   * ip_gb  (ocf:heartbeat:IPaddr2):     Started wlb01
```

It is then possible to move the resource to the other node:

```
pcs resource move ip_gb wlb02
```

- ip_gb is the name given to the resource when created.

```
Location constraint to move resource 'ip_gb' has been created
Waiting for the cluster to apply configuration changes...
Location constraint created to move resource 'ip_gb' has been removed
Waiting for the cluster to apply configuration changes...
resource 'ip_gb' is running on node 'wlb02'
```

The resource has been successfully migrated to wlb02. This can be useful, for example, when maintenance will be done on wlb01 and wlb02 should host the virtual IP.

To move it back to wlb01

```
pcs resource move ip_gb wlb01
```

4. HAProxy configuration

HAProxy introduces the concept of `frontend` and `backend`. A `fronted` defines the port on which HAProxy will accept incoming connections. A `backend` describes the set of servers where HAProxy will forward incoming connections. In addition to this proxy feature, HAProxy can:

- terminate the `ssl` connection
- detect the the protocol used is effectively correct
- check if every backend server is working as expected

The configuration below, to be added at the end of the default `/etc/haproxy/haproxy.cfg` will:

- relay port **443** to port **8083** after having terminated the **SSL** connection. **8083** is the default port for MQTT over websocket on EMQX.
- relay port **8883** to port **1883** after having terminated the **SSL** connection. **8883** is the default port for MQTTS and **1883** the default port for MQTT.
- relay port **18083** to port **18083**. **18083** is the default port for the EMQX web interface. This interface **must not** be accessible from the internet.

In this example `domain.org` should be changed to the domain name of the organisation.

```
backend mqtt_backend
  mode tcp
  stick-table type string len 32 size 100k expire 30m
  stick on req.payload(0,0),mqtt_field_value(connect,client_identifier)
  balance roundrobin
  server wbroker01 wbroker01.domain.org:1883 check
```

```
  server wbroker02 wbroker02.domain.org:1883 check
  server wbroker03 wbroker03.domain.org:1883 check

frontend mqtts_frontend
  bind *:8883 ssl crt /etc/haproxy/certs/globalbroker.domain.org.pem
  mode tcp
  tcp-request inspect-delay 10s
  tcp-request content reject unless { req.payload(0,0),mqtt_is_valid }
  default_backend mqtt_backend

backend ws_backend
  mode tcp
  balance roundrobin
  server wbroker01 wbroker01.domain.org:8083 check
  server wbroker02 wbroker02.domain.org:8083 check
  server wbroker03 wbroker03.domain.org:8083 check

frontend wss_frontend
  bind *:443 ssl crt /etc/haproxy/certs/globalbroker.domain.org.pem
  mode tcp
  default_backend ws_backend

backend dash_backend
  mode tcp
  balance roundrobin
  server wbroker01 wbroker01.domain.org:18083 check
  server wbroker02 wbroker02.domain.org:18083 check
  server wbroker03 wbroker03.domain.org:18083 check

frontend dash_frontend
  bind *:18083
  mode tcp
  default_backend dash_backend
```

After having saved `haproxy.cfg`, start HAProxy and configure to be started at boot.

```
systemctl start haproxy
systemctl enable haproxy
```

This completes the configuration of the load balancer.