Installation of a WIS2 Global Broker

Table of Contents

Based on the experience of Météo-France Global Broker and using only freely available tools, the following describes an implementation of a WIS2 compliant Global Broker. See https://wmo-im.github.io/wis2-guide/guide/wis2-guide-DRAFT.html

1. The software components

1.1. The must

- Docker
  No need to describe what docker does...
- Ansible
  A deployment management software tool. Running the same command on multiple hosts becomes so easy. And much more.
- Traefik
  A very versatile tool used extensively in this configuration:
  - terminating SSL connections
  - managing the active/passive setup of the antiloop service
  - managing multiple service sharing a single IP address
  - ...
- Redis
  Redis is a very fast key/value database. Deployed in a cluster mode, it provides the redundancy for achieving a very high SLA. The recommended practice is to build the redis cluster using a minimum of 6 instances. This is what is done here.
- EMQX

A state-of-the-art MQTT broker. Compliant with MQTT v3.1.1 and v5, it can be deployed in a cluster using docker

- Antiloop
  This is the specific WIS2 function. It discards messages received multiple times. Details can by found in the guide on WIS, see above. The tool is available as an opensource docker container.
- Prometheus
  In order to make the required statistics available to the Global Monitoring, prometheus is required.

1.2. The (strongly) suggested

- Portainer
  A tool to provide an easy to manage docker based deployment on multiple hosts. This is not absolutely necessary. It makes the management of the overall solution easier.
- Grafana
  To visualize locally the statistics stored in prometheus, grafana can be deployed.

1.3. Last but not least

- A valid certificate
  The Global Broker will be reachable using `wss` and `mqtts` protocols. A valid certificate must be used to ensure to users that the identity of the Global Broker is correct. It is possible to use Let's Encrypt to obtain and renew the certificate. Configuring Let's Encrypt is beyond the scope of this guide.

2. The architecture of the deployment

A total of seven (7) hosts or Virtual Machines (VM) is required. For the rest of the guide, we will consider only the VM options. 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.

Three VMs will host the cluster of EMQX broker. As a convention, they will be called wbroker01, wbroker02 and wbroker03.

Three VMs will host the Antiloop containers. For each WIS2 Node, two instances of the same container will be deployed randomly on the three hosts. As a convention, they will called waloop01, waloop02 and waloop03.

The seventh VM is the management host. All setup will be done on this VMs and using Ansible the required configuration will be distributed to all wbroker0x and waloop0x VMs. As a convention, it will be called wmanage.

The suggested configuration for each wbroker0x and waloop0x VM is as follow:

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

```
Disk: 80 GBytes
```

The suggested configuration for the wmanage VM is:

```
CPU: 2 cores
RAM: 16 GBytes
Disk: 80 GBytes
```

On those VMs, a linux distribution must be deployed. The prerequisites for the distribution are very low. On wmanage, ansible is required. It needs python3.x. On all 7 VMs, docker has to be installed. Both tools (python3 and docker) are available on all modern linux distributions.

After having installed the Linux Operating System, and before installing the additional tools, a minimum of three users must be created:

- One user to run docker containers. This is an unprivileged user, without sudo rights. For the rest of this guide this user will be called `gb` and its home directory will be `/home/gb`. The username can be anything. No need to have it as `gb`. For security reasons, it shouldn't be possible to connect directly using this `gb` user.
- One user to run ansible. This is a standard user with sudo privileges without requiring a passwd. For the rest of this guide this user will be called `ansadm` and its home directory will be `/home/ansadm`. The username can be anything. No need to have it as `ansadm`. It shouldn't be possible to connect directly using this `ansadm` user.
- One user for each person going to manage the various hosts. Those users will follow the practices of the organisation running the Global Broker. Each user having to manage the configuration should have sudo right to connect as `gb` and as `ansadm`.

It is strongly suggested to configure *ssh* access using preshared keys and not passwords.

2.1. Security consideration

3. Software components

3.1. Ansible

3.1.1. General configuration

During the build phase and the running phase, ansible facilitate the deployments of all the components. Typically, when a WIS2 Node is added to the Global Broker, a set of files (docker-compose, environments files, prometheus, traefik parameters,...) are deployed to fully configure this new node. Using ansible means that all can be scripted and deployed quickly and reliably.

Installing ansible can be done in two ways:

- Using pip3

- Using pre-defined packages (.deb, .rpm)

Both method are equally valid. Please refer to installation documentation to see how to proceed.

In the Global Broker setup, ansible must be installed **only** on the wmanage host. According to ansible terminology, wmanage is the *Control Node* and wbroker0x, waloop0x and wmanage itself are the *Managed Nodes*.

The user ansadm, see chapter above, must exists on all VMs: wmanage, wbroker0x and waloop0x. On all VMs, ansadm must have sudo rights without the need of a password. This is typically configured on a linux system with `sudo` command available by adding in `/etc/sudoers.d` directory a file with:

```
# User rules for ansadm
ansadm ALL=(ALL) NOPASSWD:ALL
```

Adapt accordingly, if your user to run ansible is not ansadm.

ssh access using preshared keys must also be configured from wmanage to wbroker0x, waloop0x.

To check that everything is configured properly, it must be possible:

- from wmanage host, and connected as user ansadm, when running `ssh wbroker01`, it must connect to wbroker01 without the need of entering a password. Repeat for the other wbroker0x and waloop0x.
- when connected as ansadm on any of wbroker0x, waloop0x and wmanage, `sudo su -` must give `root` access without the need of entering a password.

3.1.2. Inventory file

In ansible, *Managed nodes* can be grouped to facilitate the execution of the same command on a particular group of nodes. When connected as root on wmanage, create the file `/etc/ansible/hosts` with the following content :

```
[broker]
wbroker01
wbroker02
wbroker03

[antiloop]
waloop01
waloop02
waloop03

[manage]
localhost ansible_connection=local
```

It creates three groups named `broker`, `antiloop`and `manage`. Then, using ansible, it will be possible

to use those three group names to run the same command on all wbroker0x, waloop0x or on wmanage itself.

For this to work, the names wbroker01, 02, 03, waloop01, 02, 03 must resolve to the IP addresses of the various hosts. This can be done using DNS or `/etc/hosts` file on wmanage.

To check that it is working as expected, and as user ansadm run ansible broker -a uptime`. This should run the `uptime` command on all wbroker0x hosts. If successful, the result should be something like:

```
wbroker02 | CHANGED | rc=0 >>
 04:21:21 up 2 days, 14:45,  1 user,  load average: 0.00, 0.00, 0.00
wbroker01 | CHANGED | rc=0 >>
 04:21:22 up 1 day, 18:14,  1 user,  load average: 0.07, 0.02, 0.00
wbroker03 | CHANGED | rc=0 >>
 04:21:22 up 2 days, 14:43,  1 user,  load average: 0.00, 0.00, 0.00
```

You can also check on `antiloop` and `manage`. When everything is properly configured, it means that ansible is ready.

3.2. Redis

3.2.1. Configuration

On each wbroker0x and each waloop0x host:

- Create a redis directory in the home directory of the docker user

```
mkdir redis
```

- In the redis directory create a `redis.conf` file with the following content:

```
bind 0.0.0.0
protected-mode no
port 6379
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 15000
appendonly yes
```

3.2.2. Docker stack configuration

Using portainer, connect sequentially on each wbroker0x and waloop0x:

- Create a new stack called redis with the following configuration:

```
services:
  redis:
    container_name: redis
    image: redis:7.2.4-alpine3.19
    extra_hosts:
      - wbroker01: 192.168.0.1
      - wbroker02: 192.168.0.2
      - wbroker03: 192.168.0.3
      - waloop01: 192.168.0.4
      - waloop02: 192.168.0.5
      - waloop03: 192.168.0.6
    command: redis-server /data/redis.conf --save 20 1
    network_mode: host
    volumes:
      - /home/gb/redis:/data
    restart: unless-stopped
    user: 1000:1000
```

- Modify `redis:7.2.4-alpine3.19` with the latest available docker image for redis
- Modify `/home/gb` to be the home directory of the user configured to run the docker containers
- Modify `1000:1000` with the uid and gid of the user configured to run the docker containers
- Modify all IP addresses of the wbroker0x and waloop0x hosts

When done, start the container.

To check whether the container is working as expected, using portainer, look at the logs of redis. It should look similar to this:

```
* WARNING: Changing databases number from 16 to 1 since we are in cluster mode
* WARNING Memory overcommit must be enabled! Without it, a background save or
replication may fail under low memory condition. Being disabled, it can also cause
failures without low memory condition, see
https://github.com/jemalloc/jemalloc/issues/1328. To fix this issue add
'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command
'sysctl vm.overcommit_memory=1' for this to take effect.
* oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
* Redis version=7.2.4, bits=64, commit=00000000, modified=0, pid=1, just started
* Configuration loaded
* monotonic clock: POSIX clock_gettime
* Running mode=cluster, port=6379.
* No cluster configuration found, I'm 580288c54b03da55288d96c326116a09eb4297e5
* Server initialized
* Creating AOF base file appendonly.aof.1.base.rdb on server start
* Creating AOF incr file appendonly.aof.1.incr.aof on server start
* Ready to accept connections tcp
```

`Ready to accept connections tcp` indicates that redis has started.

3.2.3. Create the redis cluster

This must be done once after all six redis docker containers have been deployed and are running successfully.

On **one** of the wbroker0x or waloop0x, connect to the docker container, using portainer, open a `/bin/sh` shell on the redis container.

It shows: `/data $`

Then, enter:

```
redis-cli --cluster create wbroker01:6379 wbroker02:6379 wbroker03:6379 waloop01:6379
waloop02:6379 waloop03:6379
```

The following is displayed:

```
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 2730
Master[1] -> Slots 2731 - 5460
Master[2] -> Slots 5461 - 8191
Master[3] -> Slots 8192 - 10922
Master[4] -> Slots 10923 - 13652
Master[5] -> Slots 13653 - 16383
M: 580288c54b03da55288d96c326116a09eb4297e5 wbroker01:6379
   slots:[0-2730] (2731 slots) master
M: 0f28226023677a8cc2c517725224a127c58b8588 wbroker02:6379
   slots:[2731-5460] (2730 slots) master
M: b0bafd1023c46f9a4220b988a8de583481c2927c wbroker03:6379
   slots:[5461-8191] (2731 slots) master
M: 75b6c6d053bb59ee581fa2d63242761d6f1ddd38 waloop01:6379
   slots:[8192-10922] (2731 slots) master
M: 9510abcd3d18228e997cc49e1938b1c114155a9c waloop02:6379
   slots:[10923-13652] (2730 slots) master
M: 6c718043207b534b59642ab4be0d87e0af0ad669 waloop03:6379
   slots:[13653-16383] (2731 slots) master
Can I set the above configuration? (type 'yes' to accept):
```

Enter `yes`, the following is shown:

```
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
.
>>> Performing Cluster Check (using node wbroker01:6379)
M: 580288c54b03da55288d96c326116a09eb4297e5 wbroker01:6379
   slots:[0-2730] (2731 slots) master
```

```
M: b0bafd1023c46f9a4220b988a8de583481c2927c 192.168.168.112:6379
   slots:[5461-8191] (2731 slots) master
M: 9510abcd3d18228e997cc49e1938b1c114155a9c 192.168.168.114:6379
   slots:[10923-13652] (2730 slots) master
M: 75b6c6d053bb59ee581fa2d63242761d6f1ddd38 192.168.168.113:6379
   slots:[8192-10922] (2731 slots) master
M: 6c718043207b534b59642ab4be0d87e0af0ad669 192.168.168.115:6379
   slots:[13653-16383] (2731 slots) master
M: 0f28226023677a8cc2c517725224a127c58b8588 192.168.168.111:6379
   slots:[2731-5460] (2730 slots) master
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

The last line `[OK] All 16384 slots covered.` confirms that the cluster is successfully created.

This completes the installation of redis.