Deploying WRTC in High Available Mode

WRTC solution supports Highly Availability (HA) in order to minimize service impairment due to network connectivity issues, such as IP/HTTP Proxy failures or issues, either in the client or server such as a browser crash or a WRTC instance going down. WRTC support High Availability in a cloud environment through a cluster sharing the same cluster configuration. Each node in a cluster is identified through a unique VNF Identifier. All WRTC nodes in the cluster work in Active-Active mode. Hence, each WRTC node in the cluster share session state with each other, thus a transaction can be processed by any node during failure.

The advantages of using an Active-Active architecture are:

- The application load is distributed across the cluster instead of a single node unlike Active-Standby architecture where the secondary server is not utilized.
- Any node in the cluster can start handling the sessions when the node that was servicing those sessions goes down.
- There is no delay in switching traffic from an existing node to a different node when a node goes down unlike Active-Standby mode where some delay may be there in switching the virtual IP and messages may get lost in that window.
- Maintenance and upgrade is easier in Active-Active mode where rolling upgrades can be performed across nodes.

The cluster is managed through EMS. A WRTC cluster has to be created in a EMS server. And a WRTC node can be registered to the cluster in the EMS server. During registration the WRTC node learns required configuration from the cluster and shows itself as a node of that cluster.

Each node in a cluster share its information of Cassandra DB, Hazel Cast DB and Vert.x Event Bus across all the other nodes in the same cluster. When a node goes down, websocket connection from a client is reestablished with another node in the cluster, thus a new session will be rehydrate to the client.

WRTC HA Configuration

Perform the following steps for configuring WRTC in HA mode:

1. Create a Cluster in EMS server with required cluster configuration. Refer to Configuring WRTC Using EMS.
2. Register each WRTC nodes to the cluster with unique VNF ID. For example if you are registering three nodes Node 1, Node 2 and Node 3.

   Each of the nodes should have unique VNF ID, Same Cluster ID and IP address for each nodes.
3. Configure WRTC load balancer to share traffic to WRTC upstream servers. If you are using NGINX as load balancer refer to WRTC Load Balancer Configuration to configure the same.
4. Below is the sample configuration of NGINX for three WRTC nodes acting in HA mode.

```
#user nobody;
worker_processes 1;
#error_log logs/error.log;
#error_log logs/error.log notice;
#error_log logs/error.log info;
#pid logs/nginx.pid;
events {
worker_connections 1024;
}
http {
include mime.types;
default_type application/octet-stream;
fastcgi_buffers 8 16k;
fastcgi_buffer_size 32k;
upstream ws_rack{
# ip_hash;
server 10.54.48.11:9080;
server 10.54.48.12:9080;
server 10.54.48.152:9080;
}
upstream http_rack_443{
server 10.54.48.11:8088;
server 10.54.48.12:8088;
server 10.54.48.152:8088;
}
upstream http_rack_80{
server 10.54.48.11:80;
server 10.54.48.12:80;
server 10.54.48.152:80;
```

```
}
upstream http_rack_8081{
server 10.54.48.11:8081;
server 10.54.48.12:8081;
server 10.54.48.152:8081;
}
server {
listen 9080 ssl;
server_name 10.54.48.57;
large_client_header_buffers 8 32k;
ssl on;
ssl_certificate /opt/nginx/ssl/server.crt;
ssl_certificate_key /opt/nginx/ssl/server.key;
ssl_session_timeout 5m;
ssl_protocols SSLv2 SSLv3 TLSv1;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers on;
#ssl_verify_client off;
location / {
#root html;
# index index.html index.htm;
proxy_pass http://ws_rack;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
# WebSocket support

proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
proxy_set_header Host $host;
proxy_read_timeout 86400;
proxy_redirect off;
proxy_buffers 8 32k;
proxy_buffer_size 64k;

# proxy_ssl_session_reuse off;
# proxy_set_header X_FORWARDED_PROTO https;
}

}
server {
listen 443 ssl;
server_name 10.54.48.57;
large_client_header_buffers 8 32k;
ssl on;
ssl_certificate /opt/nginx/ssl/server.crt;
ssl_certificate_key /opt/nginx/ssl/server.key;
ssl_session_timeout 5m;
ssl_protocols SSLv2 SSLv3 TLSv1;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers on;
#ssl_verify_client off;
tcp_nopush on;
location / {
#root html;
index index.html index.htm;
proxy_pass http://http_rack_443/;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_buffers 8 16k;
proxy_buffer_size 32k;
fastcgi_buffers 8 16k;
fastcgi_buffer_size 32k;
proxy_read_timeout 300;
proxy_connect_timeout 300;
tcp_nopush on;
}
}
server {
listen 80;
```

```
server_name 10.54.48.57;
large_client_header_buffers 8 32k;
#ssl_verify_client off;
tcp_nopush on;
location / {
#root html;
index index.html index.htm;
proxy_pass http://http_rack_80/;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_buffers 8 16k;
proxy_buffer_size 32k;
fastcgi_buffers 8 16k;
fastcgi_buffer_size 32k;
proxy_read_timeout 300;
proxy_connect_timeout 300;
tcp_nopush on;
}
}
server {
listen 8081 ssl;
server_name 10.54.48.57;
large_client_header_buffers 8 32k;
ssl on;
ssl_certificate /opt/nginx/ssl/server.crt;
ssl_certificate_key /opt/nginx/ssl/server.key;
ssl_session_timeout 5m;
ssl_protocols SSLv2 SSLv3 TLSv1;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers on;
#ssl_verify_client off;
tcp_nopush on;
location / {
#root html;
index index.html index.htm;
proxy_pass http://http_rack_8081/;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_buffers 8 16k;
proxy_buffer_size 32k;
fastcgi_buffers 8 16k;
fastcgi_buffer_size 32k;
proxy_read_timeout 300;
proxy_connect_timeout 300;
tcp_nopush on;
}
```

```
}
}
```

5. With the above configuration start Node 1, by running the following command. Refer Initializing WRTC Node for more information.

```
cd /opt/sonus/wrtc
./wrtcnodeinit start
```

6. After the Node 1 is registered with EMS, start WRTC application by running the following command.

```
 su - wrtc
./sonuswrtc start
```

7. When Node 1 becomes online in EMS, repeat Step 5 and 6 for Node 2 and Node 3. Now all the nodes are started and application is successfully running. Each node in the cluster will have other node details. If any node goes down, the sessions served by failed node will be rehydrate to other node in the cluster in round robin manner.