Deployment Guide

Load Balancing

5/10/2026

Load Balancing

Contents

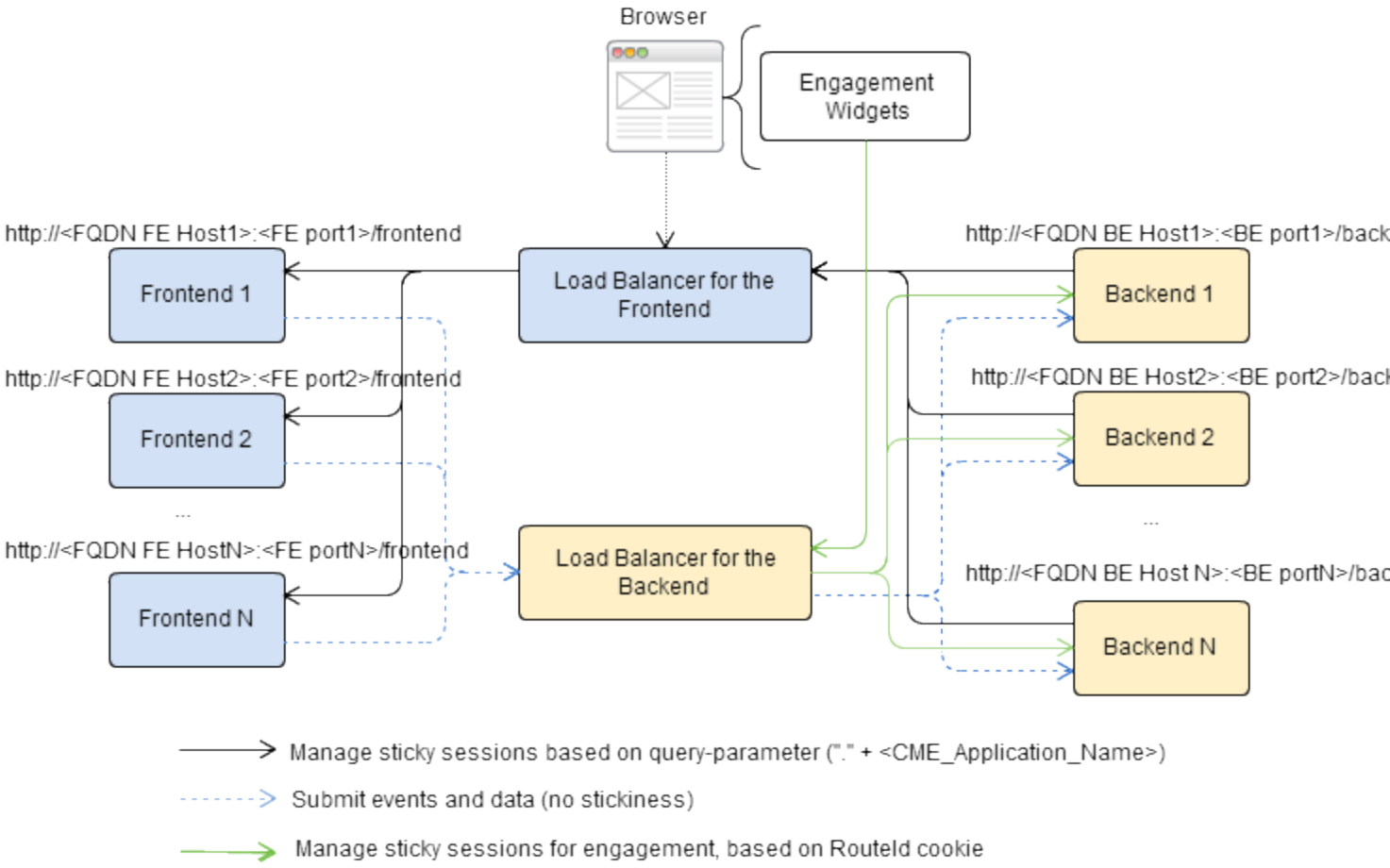
Genesys Web Engagement supports any third-party load balancer as long as the load balancing features include cookie support and URL encoding-based routing methods. You should deploy your load balancers at the latest stage of your Web Engagement deployment.

The following points are important for you to consider when setting up load balancing:

- Frontend and Backend load balancers are mandatory elements for a production deployment. Interconnecting the Frontend and Backend servers directly is only appropriate for a development environment.
- Due to Safari's strict cookie policy, Genesys recommends that your load balancer is hosted under the same domain as the website (or its subdomain). Otherwise, chat "stickiness" cookies might be rejected as "third-party", and the solution will not work (users won't be able to start chat).
- Apache does not support web sockets load balancing by default. If you want to enable this option, you must use the mod_proxy_wstunnel module. **Note:** This module requires Apache version 2.4+ and is only available for Linux.
- If your load balancer does not support WebSockets, make sure that you disable WebSockets on the client side. See Chat Application - disableWebSockets and Tracker Application - disableWebSockets for details.

Architecture

The following diagram shows how you can implement a load balancing configuration for your Web Engagement servers.



Sample of Deployment for Load Balancing

In the above example:

- The load balancer for the Frontend Server implements sticky sessions for the deployed Web Engagement application based on a query-parameter in this format: `"." + Frontend_Server_Application_Name_In_Genesys_CME`. The Frontend Server is responsible for specifying the related query-parameter.
- The load balancer for the Backend Server implements sticky sessions to route IP addresses when customers are engaged based on the **RouteId** cookie. The load balancer is responsible for specifying this cookie. This example uses **RouteId** as the name of the cookie, but you can use any name.

Sticky Sessions

Genesys Web Engagement uses sticky sessions as follows:

- The load balancer for the Frontend Server implements sticky sessions based on URL encoding static parameters for the deployed Web Engagement application. Web Engagement creates the parameter as follows:
  `"." + Frontend_Server_Application_Name_In_Genesys_CME`.

> **Important**
>
> If you use Apache, you do not need to add "." to your static parameter; this will be done by the Web Engagement Server.

    - The load balancer for the Frontend Server uses this URL parameter instead of a cookie because it also communicates with the Backend Server, which does not have access to cookies.
- The load balancer for the Backend Server implements sticky sessions based on cookies to route IP addresses when the customers are engaged. The load balancer for the Backend Server must support the following features:
    - Cookie-based stickiness to enable engagement.
    - Storage of sticky parameters into cookies.

> **Important**
>
> All the cookies are created by the load balancing system, not by the Genesys Web Engagement servers.

In both cases, GWE requires sticky sessions not only for performance reasons, but also to switch over ongoing transactions if a node (Frontend or Backend) fails.

Pure Cookie-based Stickiness

Since version 8.1.200.39, Genesys Web Engagement supports cookie-based load balancing for both Frontend and Backend servers.

It is possible to implement load balancing for the Frontend Server using cookie-based sticky sessions. In order to do so, the load balancer is required to set the following cookie:

Cookie Name	Cookie Value
FRONTEND_ROUTEID	"." + Frontend_Server_Application_Name_In_Genesys_CME

The load balancer for the Backend Server implements sticky sessions based on cookies to route IP addresses when customers are engaged. The cookie should be set by the load balancer and there are no specific requirements for the cookie's name or value.

> **Tip**
>
> When using *pure* cookie-based stickiness, you can configure a single load balancer to balance frontend and backend traffic.

Sample Configurations

Genesys provides sample load balancing configurations for two common load balancers: Apache and Nginx. For details, select a tab below:

Apache

The following procedures provide an example load balancing configuration for Apache. Before you begin, make sure you have completed the following prerequisites:

- You already deployed your Web Engagement application into a production (or production-like) environment and have at least two Frontend Servers and three Backend Servers configured to work in the cluster (see Deploying and Configuring the Genesys Web Engagement Cluster for details).
- For this configuration example, you installed and configured two instances of Apache, version 2.2 (preferably on different hosts): one to serve the Frontend cluster and another one to serve the Backend cluster.

> **Tip**
>
> When using *pure* cookie-based stickiness, you can configure a single Apache Load Balancer to balance frontend and backend traffic.

Configuring the Apache Load Balancer for the Frontend Cluster

> **Important**
>
> Your frontend load balancer must be configured to match this format:
> `http(s)://host:port/frontend`

Start

1. Confirm that the following modules are present in your Apache load balancer:
   - mod_proxy.so
   - mod_proxy_balancer.so
   - mod_proxy_connect.so
   - mod_proxy_http.so
2. Edit the **./conf/httpd.conf** file and confirm that the modules are loaded:
   - LoadModule proxy_module modules/mod_proxy.so

- LoadModule proxy_module modules/mod_proxy_balancer.so
- LoadModule proxy_module modules/mod_proxy_connect.so
- LoadModule proxy_module modules/mod_proxy_http.so

3. Add the following configuration script to the end of the **httpd.conf** file:

```
# loadbalancer configuration for GWE Frontend (sticky sessions)
 ProxyPass / balancer://my_cluster/
 <Proxy balancer://my_cluster/>
     BalancerMember [http:// http://]<Node1 IP address>:<listeningPort1> route=NodeName1
     BalancerMember [http:// http://]<Node2 IP address>:<listeningPort2> route=NodeName2
     ...
     BalancerMember [http:// http://]<NodeN IP address>:<listeningPortN> route=NodeNameN
 ProxySet stickysession=alias
 </Proxy>
 #<NodeN IP address> is the IP address of your node, and <listeningPortN>, the associated
 listening port of your frontend application
 #query parameter alias for Frontend cluster sticky session
 #NodeNameN is name of Frontend Server application related to this node
```

4. Save your changes. The load balancer for the Frontend cluster is now configured.
5. Your Frontend Server nodes are healthy if the load balancer receives a successful response on requests to `http(s)://Frontend Server Host:Frontend Server Port (secured port for https)/frontend/about`

End

Example: Load Balancer Configuration for the Frontend Cluster

```
ProxyRequests Off
<Proxy balancer://mycluster/>
   BalancerMember http://198.51.100.1:8083/frontend route=GWE_Frontend_103
   BalancerMember http://198.51.100.2:8084/frontend route=GWE_Frontend_52
ProxySet stickysession=alias
</Proxy>
ProxyPass /frontend/ balancer://mycluster/
```

Configuring the Apache Load Balancer for the Backend Cluster

> **Important**
>
> Your Backend load balancer must be configured to match this format:
> `http(s)://host:port/backend`

Start

1. Confirm that the following modules are present in your Apache load balancer:
   - mod_proxy.so
   - mod_proxy_balancer.so
   - mod_proxy_connect.so

- mod_proxy_http.so
- mod_headers.so

2. Edit the **./conf/httpd.conf** file and confirm that the modules are loaded:
   - LoadModule proxy_module modules/mod_proxy.so
   - LoadModule proxy_module modules/mod_proxy_balancer.so
   - LoadModule proxy_module modules/mod_proxy_connect.so
   - LoadModule proxy_module modules/mod_proxy_http.so
   - LoadModule proxy_module modules/mod_headers.so
3. Add the following configuration script to the end of the **httpd.conf** file:

```
# loadbalancer configuration for GWE Backend (cookie-based)
ProxyRequests Off
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/"
env=BALANCER_ROUTE_CHANGED
<Proxy balancer://mycluster/>
    BalancerMember [http:// http://]<Node1 IP address>:<listeningPort1> route=NodeName1
    BalancerMember http://<Node2 IP address>:<listeningPort2> route=NodeName2
    ...
    BalancerMember http://<NodeN IP address>:<listeningPortN> route=NodeNameN
ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass /backend/ balancer://mycluster/
```

4. Save your changes. The load balancer for the Backend cluster is now configured.
5. Your Backend Server nodes are healthy if the load balancer receives a successful response on requests to `http(s)://`*`Backend`* `Server Host:Backend Server Port (secured port for https)/`*`backend/about`*

End

Example: Load Balancer Configuration for the Backend Cluster

```
ProxyRequests Off
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
<Proxy balancer://mycluster/>
    BalancerMember http://198.51.100.1:9083/backend route=1
    BalancerMember http://198.51.100.2:9084/backend route=2
        ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass /backend/ balancer://mycluster/
```

Enabling Balancer Manager on Your Load Balancers (Optional)

Complete this procedure to enable the Apache balancer manager on your load balancers. This will make the manager available at the **/balancer** URL.

Start

Complete the following steps for the Frontend and Backend Apache load balancers.

1. Edit the **./conf/httpd.conf** file and add the following to the **end** of the file:

```
<Location /balancer>
 SetHandler balancer-manager
 Order Deny,Allow
 #Deny from all # Specify which connections should be denied
 Allow from all # Specify which connections should be allowed
 </Location>
```

2. Save your changes. The balancer manager is now enabled.

End

Configuring a Single Apache Load Balancer for Cookie-based Balancing of the Frontend and Backend Clusters

Start

1. Confirm that the following modules are present in your Apache load balancer:
   - mod_proxy.so
   - mod_proxy_balancer.so
   - mod_proxy_connect.so
   - mod_proxy_http.so
   - mod_headers.so
2. Edit the **./conf/httpd.conf** file and confirm that the modules are loaded:
   - LoadModule proxy_module modules/mod_proxy.so
   - LoadModule proxy_module modules/mod_proxy_balancer.so
   - LoadModule proxy_module modules/mod_proxy_connect.so
   - LoadModule proxy_module modules/mod_proxy_http.so
   - LoadModule proxy_module modules/mod_headers.so
3. Add support of Virtual Hosts in the `httpd.conf` file:

```
<VirtualHost *:80>
     ProxyRequests Off
     <Proxy *>
         order allow,deny
         Allow from All
     </Proxy>
     ProxyPass /frontend http://localhost:<frontend server port>/frontend
     ProxyPass /backend http://localhost:<backend server port>/backend
     ProxyPassReverse /frontend http://localhost:<frontend server port>/frontend
     ProxyPassReverse /backend http://localhost:<backend server port>/backend
 </VirtualHost>
 Listen <frontend server load-balancing port>
 Listen <backend server load-balancing port>
```

4. Add a `VirtualHost` for the Frontend servers to the end of the `httpd.conf` file:

```
<VirtualHost *:<frontend server load-balancing port>>
     Header add Set-Cookie "FRONTEND_ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/"
 env=BALANCER_ROUTE_CHANGED
```

```
    <Proxy balancer://fcluster>
       BalancerMember http://<IP of host with Frontend Server 1>:<Port of Frontend
Server 1>/frontend route=<'.' + Frontend_Server_CME_App_Name>
       ProxySet stickysession=FRONTEND_ROUTEID
    </Proxy>
    ProxyPass /frontend balancer://fcluster
    <Location /balancer-manager>
        SetHandler balancer-manager
        Order Deny,Allow
        Allow from all
     </Location>
</VirtualHost>
```

5. Add a `VirtualHost` for the Backend servers to the end of the `httpd.conf` file:

```
<VirtualHost *:<backend server load-balancing port>>
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/"
env=BALANCER_ROUTE_CHANGED
<Proxy balancer://bcluster/>
    BalancerMember http://<IP of host with Backend Server 1>:<port Backend Server
1>>/backend route=1
    ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass /backend/ balancer://bcluster/
ProxyPassReverse /backend/ balancer://bcluster/
<Location /balancer-manager>
SetHandler balancer-manager
Order Deny,Allow
Allow from all
</Location>
</VirtualHost>
```

6. Save your changes. You have now configured a single load balancer for the Backend and Frontend clusters.

End

Example: Single Load Balancer for Backend and Frontend Cluster

```
<VirtualHost *:80>
ProxyRequests Off
<Proxy *>
order allow,deny
Allow from All
</Proxy>
ProxyPass /frontend http://localhost:8081/frontend
ProxyPass /backend http://localhost:9081/backend
ProxyPassReverse /frontend http://localhost:8081/frontend
ProxyPassReverse /backend http://localhost:9081/backend
</VirtualHost>
Listen 8081
Listen 9081
<VirtualHost *:8081>
Header add Set-Cookie "FRONTEND_ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/"
env=BALANCER_ROUTE_CHANGED
<Proxy balancer://fcluster>
   BalancerMember http://192.168.100.1:8081/frontend route=GPE_Frontend_1
   BalancerMember http://192.168.100.2:8082/frontend route=GPE_Frontend_2
   BalancerMember http://192.168.100.3:8082/frontend route=GPE_Frontend_3
ProxySet stickysession=FRONTEND_ROUTEID
</Proxy>
ProxyPass /frontend balancer://fcluster
```

```
<Location /balancer-manager>
SetHandler balancer-manager
Order Deny,Allow
Allow from all
</Location>
</VirtualHost>
<VirtualHost *:9081>
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
<Proxy balancer://bcluster/>
    BalancerMember http://192.168.100.1:9081/backend route=1
    BalancerMember http://192.168.100.2:9082/backend route=2
    BalancerMember http://192.168.100.2:9083/backend route=3
    ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass /backend/ balancer://bcluster/
ProxyPassReverse /backend/ balancer://bcluster/
<Location /balancer-manager>
SetHandler balancer-manager
Order Deny,Allow
Allow from all
</Location>
</VirtualHost>
```

Nginx

The information below includes sample Nginx configurations for both the load blanacer for the Frontend Server cluster and the load balancer for the Backend Server cluster.

Prerequisites

- You already deployed your Web Engagement application into a production (or production-like) environment and have at least two Frontend Servers and three Backend Servers configured to work in the cluster (see Deploying and Configuring the Genesys Web Engagement Cluster for details).
- For this configuration sample, you installed and configured two instances of Nginx (preferably on different hosts): one to serve the Frontend cluster and another one to serve the Backend cluster.

To configure your Nginx load balancers, edit the **./conf/nginx.conf** file and modify the configuration according to the samples provided below for the Frontend and Backend load balancers. For details about the configuration, consult the Nginx documentation.

Configuration Sample: Nginx Load Balancer for the Frontend Cluster

```
events {
    worker_connections  1024;
}
http {
```

```
    include       mime.types;
    default_type  application/octet-stream;
     # to handle longer names of GPE server applications
    map_hash_bucket_size 64;

log_format  main  '$remote_addr - $remote_user [$time_local] "$request" "$arg_alias"';

    access_log  logs/nginx_access.log main;
    error_log logs/nginx_error.log warn;

    upstream http_frontend_cluster {
        server 135.17.36.107:8088 fail_timeout=30s;
        server 135.225.51.237:8088 fail_timeout=30s;
        server 135.17.36.10:8088 fail_timeout=30s;
    }
    map $arg_alias $http_sticky_frontend {
        .GPE_F_n1_107 135.17.36.107:8088;
        .GPE_F_n3_237 135.225.51.237:8088;
        .GPE_F_n2_10 135.17.36.10:8088;
    }
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
    server {
        listen 8088;

        location @fallback {
            proxy_pass http://http_frontend_cluster;
        }

        location /frontend {
            # Allow websockets, see http://nginx.org/en/docs/http/websocket.html
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;

            # Increase buffer sizes to find room for DOM and CSS messages
            proxy_buffers 8 2m;
            proxy_buffer_size 10m;
            proxy_busy_buffers_size 10m;
            proxy_connect_timeout 5s;

            # Fall back if server responds incorrectly
            error_page 502 = @fallback;
            # or if doesn't respond at all.
            error_page 504 = @fallback;

            # Create a map of choices
            # see https://gist.github.com/jrom/1760790
            if ($scheme = 'http') {
                set $test HTTP;
            }
            if ($http_sticky_frontend) {
                set $test "${test}-STICKY";
            }
            if ($test = HTTP-STICKY) {
                proxy_pass http://$http_sticky_frontend$uri?$args;
                break;
            }
            if ($test = HTTP) {
                proxy_pass http://http_frontend_cluster;
                break;
```

```
            }
            return 500 "Misconfiguration";
        }
    }
}
```

Configuration Sample: Nginx Load Balancer for the Backend Cluster

```
events {
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
     # to handle longer names of GPE server applications
    map_hash_bucket_size 64;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" "$arg_alias"
"$cookie_routeid"';

    access_log  logs/nginx_access.log main;
    error_log logs/nginx_error.log warn;

    upstream http_backend_cluster {
        server 135.17.36.107:9088 fail_timeout=5s;
        server 135.225.51.237:9088 fail_timeout=5s;
        server 135.17.36.10:9088 fail_timeout=5s;

    }
    map $cookie_ROUTEID $http_sticky_backend {
        default 0;
        .1 135.17.36.107:9088;
        .2 135.225.51.237:9088;
        .3 135.17.36.10:9088;
    }
    map $upstream_addr $serverid {
        135.17.36.107:9088 .1;
        135.225.51.237:9088 .2;
        135.17.36.10:9088 .3;
    }
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
    server {
        listen 9088;

        location @fallback {
            proxy_pass http://http_backend_cluster;
        }
        location /backend {
            # Allow websockets, see http://nginx.org/en/docs/http/websocket.html
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;

            # Increase buffer sizes to find room for DOM and CSS messages
```

```
            proxy_buffers 8 2m;
            proxy_buffer_size 10m;
            proxy_busy_buffers_size 10m;
            proxy_connect_timeout 5s;

            # Fall back if server responds incorrectly
            error_page 502 = @fallback;
            # or if doesn't respond at all.
            error_page 504 = @fallback;

            # Create a map of choices
            # see https://gist.github.com/jrom/1760790

            if ($scheme = 'http') {
                set $test HTTP;
            }
            if ($http_sticky_backend) {
                set $test "${test}-STICKY";
            }
            if ($test = HTTP-STICKY) {
                proxy_pass http://$http_sticky_backend$uri?$args;
                break;
            }
            if ($test = HTTP) {
            add_header Set-Cookie "ROUTEID=$serverid;Max-Age=31536000;";
                proxy_pass http://http_backend_cluster;
                break;
            }
            return 500 "Misconfiguration";
        }
    }
}
```

Next Steps

- If you are completing the Clustering deployment scenario, you can return to Configuring Load Balancing.