Basic Configuration for Reverse Proxy and SSL Termination

Here's a step-by-step guide to set up HAProxy as a reverse proxy with SSL termination:

Step 1: Install HAProxy On a Debian/Ubuntu system, you can install HAProxy using:

```
sudo apt-get update
sudo apt-get install haproxy
```

Step 2: Obtain SSL Certificate You need an SSL certificate and private key. This can be obtained from a Certificate Authority (CA) or generated using Let's Encrypt.

Combine the certificate and key into a single file:

```
cat /etc/ssl/private/your_domain.crt /etc/ssl/private/your_domain.key >
/etc/ssl/private/haproxy.pem
```

Step 3: Configure HAProxy Edit the HAProxy configuration file, typically located at /etc/haproxy/haproxy.cfg.

```
global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon
    tune.ssl.default-dh-param 2048

defaults
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend http_front
    bind *:80
    bind *:443 ssl crt /etc/ssl/private/haproxy.pem
    redirect scheme https code 301 if !{ ssl_fc }
    default_backend http_back

backend http_back
    balance roundrobin
    option httpchk GET /health
```

```
    http-check expect status 200
    server web1 192.168.1.1:80 check
    server web2 192.168.1.2:80 check
```

Explanation of Configuration:

1. Global Section: - Configures logging, runtime chroot, and user/group under which HAProxy runs. - Sets the default DH parameter size for SSL.

2. Defaults Section: - Specifies default logging, timeout, and connection options.

3. Frontend Section: - `bind *:80`: Binds HAProxy to port 80 for HTTP traffic. - `bind *:443 ssl crt /etc/ssl/private/haproxy.pem`: Binds HAProxy to port 443 for HTTPS traffic and specifies the SSL certificate. - `redirect scheme https code 301 if !{ ssl_fc }`: Redirects HTTP traffic to HTTPS. - `default_backend http_back`: Specifies the default backend to use for incoming traffic.

4. Backend Section: - `balance roundrobin`: Distributes requests using the round-robin algorithm. - `option httpchk GET /health`: Performs HTTP health checks on backend servers. - `http-check expect status 200`: Expects a 200 OK response for health checks. - `server web1 192.168.1.1:80 check`: Specifies a backend server and its health check configuration. - `server web2 192.168.1.2:80 check`: Specifies another backend server.

Advanced Configuration Options:

Using Let's Encrypt for SSL Certificates To automate SSL certificate issuance and renewal with Let's Encrypt, you can use a tool like `certbot` and configure HAProxy to use the generated certificates.

1. Install Certbot:

```
sudo apt-get install certbot
```

2. Generate Certificate:

```
sudo certbot certonly --standalone -d your_domain.com
```

3. Configure HAProxy to Use Certbot Certificates:

Update your HAProxy configuration to use the Certbot-generated certificates:

```
frontend http_front
    bind *:80
    bind *:443 ssl crt /etc/letsencrypt/live/your_domain.com/fullchain.pem
crt /etc/letsencrypt/live/your_domain.com/privkey.pem
    redirect scheme https code 301 if !{ ssl_fc }
    default_backend http_back
```

Enabling HTTP/2 HAProxy supports HTTP/2 for encrypted connections. To enable HTTP/2, add the alpn h2,http/1.1 directive to the SSL binding in the frontend configuration.

```
frontend http_front
    bind *:443 ssl crt /etc/ssl/private/haproxy.pem alpn h2,http/1.1
```

Final Configuration Example

Here is the final configuration file, incorporating all the above elements:

```
global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon
    tune.ssl.default-dh-param 2048

defaults
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend http_front
    bind *:80
    bind *:443 ssl crt /etc/ssl/private/haproxy.pem alpn h2,http/1.1
    redirect scheme https code 301 if !{ ssl_fc }
    default_backend http_back

backend http_back
    balance roundrobin
    option httpchk GET /health
    http-check expect status 200
    server web1 192.168.1.1:80 check
    server web2 192.168.1.2:80 check

listen stats
    bind *:8404
    stats enable
    stats uri /haproxy?stats
    stats refresh 10s
    stats auth admin:password
```

Conclusion By configuring HAProxy as a reverse proxy with SSL termination, you can efficiently manage your web traffic, offload SSL processing from backend servers, and enhance the security and performance of your application. With advanced options like HTTP/2 support and automated certificate management, HAProxy becomes a powerful tool in your infrastructure toolkit.

Enabling HAProxy Statistics

To enable and configure HAProxy statistics, you need to add a listen or frontend section in your HAProxy configuration file. Here's a step-by-step guide to set up and access the HAProxy statistics:

Configure the HAProxy Statistics Section Add the following section to your haproxy.cfg file:

```
listen stats
    bind *:8404
    mode http
    stats enable
    stats uri /haproxy?stats
    stats refresh 10s
    stats show-node
    stats auth admin:password
    stats admin if LOCALHOST
```

Explanation of Configuration - `listen stats`: Creates a new listener named stats. - `bind *:8404`: Binds the statistics page to port 8404 on all interfaces. - `mode http`: Sets the mode to HTTP for accessing the statistics page. - `stats enable`: Enables the statistics reporting. - `stats uri /haproxy?stats`: Sets the URI path to access the statistics page. - `stats refresh 10s`: Refreshes the statistics page every 10 seconds. - `stats show-node`: Displays the HAProxy node name on the stats page. - `stats auth admin:password`: Protects the statistics page with basic authentication (username: admin, password: password). - `stats admin if LOCALHOST`: Allows administrative actions (e.g., stopping servers) only from localhost.

Accessing HAProxy Statistics Start or Restart HAProxy: After adding the statistics configuration, start or restart the HAProxy service to apply the changes.

```
sudo systemctl restart haproxy
```

Access the Statistics Page: Open a web browser and navigate to the HAProxy statistics page. For example:

```
http://your-haproxy-server:8404/haproxy?stats
```

Authenticate: Enter the username and password configured in the stats auth directive (in this case, `admin` and `password`).

HAProxy Using ACLs

Once defined, ACLs can be used with actions like `http-request`, `use_backend`, `redirect`, and `deny`.

Example: Blocking Traffic Based on IP

```
frontend http_front
    bind *:80
    acl blocked_ip src 192.168.2.0/24
    http-request deny if blocked_ip
    default_backend http_back
```

Example: Redirecting Based on Path

```
frontend http_front
    bind *:80
    acl is_old_path path_beg /old
    http-request redirect location /new if is_old_path
    default_backend http_back
```

Example: Routing Based on HTTP Header

```
frontend http_front
    bind *:80
    acl is_mobile hdr_sub(User-Agent) Mobile
    use_backend mobile_back if is_mobile
    default_backend http_back
```

Complex Conditions You can combine multiple ACLs using logical operators (and, or, !). For instance:

Example: Combining ACLs

```
frontend http_front
    bind *:80
    acl is_admin src 192.168.1.100
    acl is_post method POST
    http-request deny if is_admin !is_post
    default_backend http_back
```

Defining Backends and Using ACLs ACLs are often used to route traffic to different backends based on the conditions.

Example: Routing to Different Backends

```
frontend http_front
    bind *:80
```

```
    acl is_static path_beg /static
    acl is_api path_beg /api
    use_backend static_back if is_static
    use_backend api_back if is_api
    default_backend http_back

backend static_back
    server static1 192.168.1.1:80 check
    server static2 192.168.1.2:80 check

backend api_back
    server api1 192.168.2.1:80 check
    server api2 192.168.2.2:80 check

backend http_back
    server web1 192.168.3.1:80 check
    server web2 192.168.3.2:80 check
```

Example: Advanced Configuration with ACLs and Conditions

```
global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

defaults
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend http_front
    bind *:80
    # Define ACLs
    acl is_admin src 192.168.1.100
    acl is_static path_beg /static
    acl is_api path_beg /api
    acl is_get method GET
    acl is_post method POST
    acl has_auth_token hdr_sub(Authorization) Bearer

    # Apply ACLs
    http-request deny if !is_admin has_auth_token
    http-request redirect location /new_static if is_static
    use_backend static_back if is_static
```

```
    use_backend api_back if is_api
    default_backend http_back

backend static_back
    balance roundrobin
    server static1 192.168.1.1:80 check
    server static2 192.168.1.2:80 check

backend api_back
    balance leastconn
    option httpchk GET /health
    http-check expect status 200
    server api1 192.168.2.1:80 check
    server api2 192.168.2.2:80 check

backend http_back
    balance roundrobin
    server web1 192.168.3.1:80 check
    server web2 192.168.3.2:80 check

listen stats
    bind *:8404
    mode http
    stats enable
    stats uri /haproxy?stats
    stats refresh 10s
    stats show-node
    stats auth admin:password
    stats admin if LOCALHOST
```

Content Switching Configuration

Content switching typically involves defining Access Control Lists (ACLs) and using those ACLs to determine the appropriate backend for each request.

Example: Basic Content Switching Suppose you have a website with static content, an API, and a main application. You want to route requests based on the URL path: - **Static content** goes to `static_back`. - **API requests** go to `api_back`. - **Other requests** go to `app_back`. Here's how you can configure HAProxy to achieve this:

```
global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

defaults
```

```
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend http_front
    bind *:80

    # Define ACLs
    acl is_static path_beg /static
    acl is_api path_beg /api

    # Use backends based on ACLs
    use_backend static_back if is_static
    use_backend api_back if is_api
    default_backend app_back

backend static_back
    server static1 192.168.1.1:80 check
    server static2 192.168.1.2:80 check

backend api_back
    server api1 192.168.2.1:80 check
    server api2 192.168.2.2:80 check

backend app_back
    server app1 192.168.3.1:80 check
    server app2 192.168.3.2:80 check

listen stats
    bind *:8404
    mode http
    stats enable
    stats uri /haproxy?stats
    stats refresh 10s
    stats show-node
    stats auth admin:password
    stats admin if LOCALHOST
```

Advanced Content Switching You can define more complex content switching rules by using multiple ACLs and combining them with logical operators.

Example: Combining Multiple Criteria In this example, requests are routed based on both the URL path and the presence of a specific header.

```
global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
```

```
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

defaults
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend http_front
    bind *:80

    # Define ACLs
    acl is_static path_beg /static
    acl is_api path_beg /api
    acl has_auth_token hdr_sub(Authorization) Bearer

    # Use backends based on combined ACLs
    use_backend static_back if is_static
    use_backend api_back if is_api !has_auth_token
    use_backend secure_api_back if is_api has_auth_token
    default_backend app_back

backend static_back
    server static1 192.168.1.1:80 check
    server static2 192.168.1.2:80 check

backend api_back
    server api1 192.168.2.1:80 check
    server api2 192.168.2.2:80 check

backend secure_api_back
    server secure_api1 192.168.2.3:80 check
    server secure_api2 192.168.2.4:80 check

backend app_back
    server app1 192.168.3.1:80 check
    server app2 192.168.3.2:80 check

listen stats
    bind *:8404
    mode http
    stats enable
    stats uri /haproxy?stats
    stats refresh 10s
    stats show-node
    stats auth admin:password
    stats admin if LOCALHOST
```

Content Switching with SSL Termination If you need to perform content switching on an HTTPS frontend, you must configure SSL termination.

```
global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon
    tune.ssl.default-dh-param 2048

defaults
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend https_front
    bind *:443 ssl crt /etc/ssl/private/haproxy.pem
    redirect scheme https code 301 if !{ ssl_fc }

    # Define ACLs
    acl is_static path_beg /static
    acl is_api path_beg /api
    acl has_auth_token hdr_sub(Authorization) Bearer

    # Use backends based on combined ACLs
    use_backend static_back if is_static
    use_backend api_back if is_api !has_auth_token
    use_backend secure_api_back if is_api has_auth_token
    default_backend app_back

backend static_back
    server static1 192.168.1.1:80 check
    server static2 192.168.1.2:80 check

backend api_back
    server api1 192.168.2.1:80 check
    server api2 192.168.2.2:80 check

backend secure_api_back
    server secure_api1 192.168.2.3:80 check
    server secure_api2 192.168.2.4:80 check

backend app_back
    server app1 192.168.3.1:80 check
    server app2 192.168.3.2:80 check
```

```
listen stats
    bind *:8404
    mode http
    stats enable
    stats uri /haproxy?stats
    stats refresh 10s
    stats show-node
    stats auth admin:password
    stats admin if LOCALHOST
```

HAProxy HTTP Redirection & Rewriting

HAProxy provides powerful features for HTTP rewriting and redirection, enabling you to modify URLs and redirect traffic based on specific conditions. These features are essential for tasks such as URL normalization, security enhancements, and user-friendly redirects.

Example: Basic HTTP to HTTPS Redirection Redirect all HTTP traffic to HTTPS:

```
frontend http_front
    bind *:80
    redirect scheme https code 301 if !{ ssl_fc }
```

Example: Redirect Based on URL Path Redirect requests from /oldpath to /newpath:

```
frontend http_front
    bind *:80
    acl is_oldpath path_beg /oldpath
    http-request redirect location /newpath if is_oldpath
    default_backend http_back
```

Example: Basic URL Rewriting Rewrite URLs starting with /oldpath to /newpath:

```
frontend http_front
    bind *:80
    acl is_oldpath path_beg /oldpath
    http-request set-path /newpath%[path,regsub(^/oldpath,/)]
    default_backend http_back
```

Example: Rewriting with Query Parameters Rewrite URL and add a query parameter:

```
frontend http_front
    bind *:80
    acl is_oldpath path_beg /oldpath
    http-request set-query newparam=value if is_oldpath
```

```
    http-request set-path /newpath%[path,regsub(^/oldpath,/)]
    default_backend http_back
```

Advanced Examples

Example: Combining Redirection and Rewriting Redirect traffic from HTTP to HTTPS and rewrite the URL path:

```
frontend http_front
    bind *:80
    redirect scheme https code 301 if !{ ssl_fc }

frontend https_front
    bind *:443 ssl crt /etc/ssl/private/haproxy.pem
    acl is_oldpath path_beg /oldpath
    http-request set-path /newpath%[path,regsub(^/oldpath,/)] if is_oldpath
    default_backend https_back

backend https_back
    server web1 192.168.1.1:443 check ssl verify none
    server web2 192.168.1.2:443 check ssl verify none
```

Example: Conditional Redirection Based on Header Redirect requests with a specific header to a different domain:

```
frontend http_front
    bind *:80
    acl is_special_user hdr_sub(User-Agent) SpecialClient
    http-request redirect location https://special.example.com if
is_special_user
    default_backend http_back
```

URL Normalization Normalize URLs by removing or modifying specific parts of the URL:

```
frontend http_front
    bind *:80
    # Remove trailing slash
    acl has_trailing_slash path_end /
    http-request set-path %[path,regsub(/$,)] if has_trailing_slash
    default_backend http_back
```

Full Configuration Example Combining various redirection and rewriting rules in a single configuration:

```
global
    log /dev/log local0
```

```
    log /dev/log local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

defaults
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend http_front
    bind *:80
    redirect scheme https code 301 if !{ ssl_fc }

frontend https_front
    bind *:443 ssl crt /etc/ssl/private/haproxy.pem

    # ACLs for URL paths
    acl is_oldpath path_beg /oldpath
    acl is_special_user hdr_sub(User-Agent) SpecialClient

    # Redirection and Rewriting
    http-request redirect location https://special.example.com if
is_special_user
    http-request set-path /newpath%[path,regsub(^/oldpath,/)] if is_oldpath

    # Normalize URLs
    acl has_trailing_slash path_end /
    http-request set-path %[path,regsub(/$,)] if has_trailing_slash

    default_backend https_back

backend https_back
    server web1 192.168.1.1:443 check ssl verify none
    server web2 192.168.1.2:443 check ssl verify none

listen stats
    bind *:8404
    mode http
    stats enable
    stats uri /haproxy?stats
    stats refresh 10s
    stats show-node
    stats auth admin:password
    stats admin if LOCALHOST
```

HA