Configuring Websockets behind an AWS ELB

2015 July 20

AWS (//blog.jverkamp.com/category/programming/by-topic/aws) Docker (//blog.jverkamp.com/category/programming/by-topic/docker) Flask (//blog.jverkamp.com/category/programming/by-topic/flask) Networks (//blog.jverkamp.com/category/programming/by-topic/networks) Programming (//blog.jverkamp.com/category/programming) Python (//blog.jverkamp.com/category/programming/by-language/python) Websites (//blog.jverkamp.com/category/programming/by-topic/websites) nginx (//blog.jverkamp.com/category/programming/by-topic/nginx)

Recently at work, we were trying to get an application that uses websockets (https://en.wikipedia.org/wiki/websockets) working on an AWS (https://aws.amazon.com/) instance behind an ELB (load balancer) (https://aws.amazon.com/elasticloadbalancing/) and nginx (http://nginx.org/) on the instance.

If you're either not using a secure connection or handling the cryptography on the instance (either in nginx or Flask), it works right out of the box. But if you want the ELB to handle TLS termination it doesn't work nearly as well... Luckily, after a bit of fiddling, I got it working.

First, we have a basic application. For my purposes, I wrote a quick Websocket chat app: ws-chat (https://github.com/jpverkamp/ws-chat). The particular implementation details aren't as important. We'll start with the nginx config file:

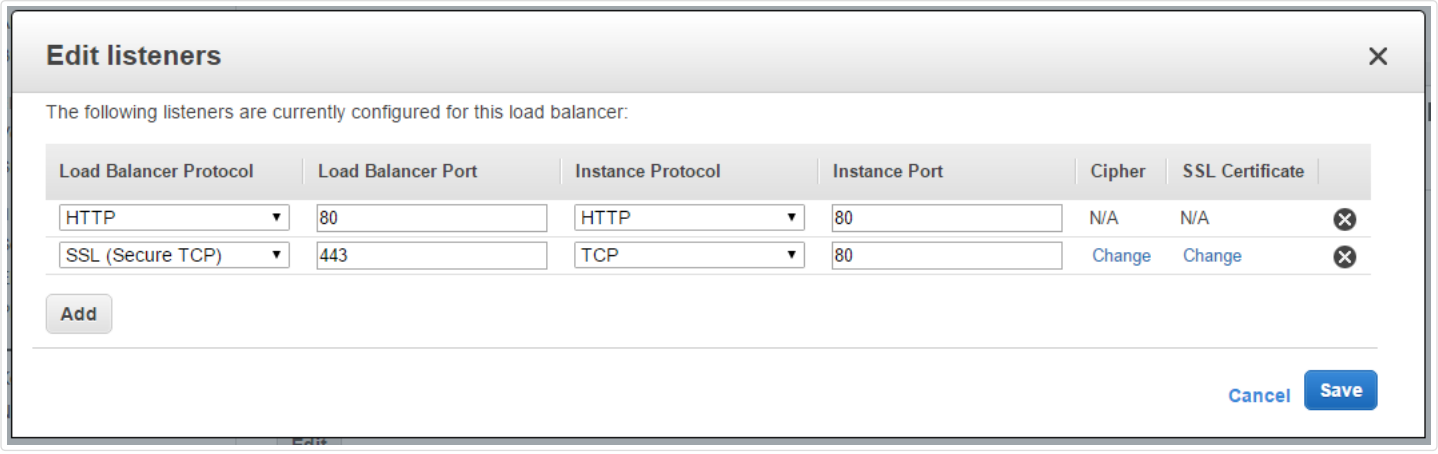
```
upstream webserver {
    server 127.0.0.1:8000;
}

upstream wsserver {
    server 127.0.0.1:9000;
}

server {
    listen 80 proxy_protocol;

    location / {
        if ($http_x_forwarded_proto = "http") {
            return 301 https://$host$request_uri;
        }

        proxy_pass http://webserver;
    }

    location /ws/ {
        proxy_pass http://wsserver;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

Straight forward enough. We have two backend services: a web server (https://github.com/jpverkamp/ws-chat/blob/master/app/web-server.py) running on port 8000 (a simple Flask server that just servers a single HTML page (https://github.com/jpverkamp/ws-chat/blob/master/app/templates/index.html)) and the websocket backend (https://github.com/jpverkamp/ws-chat/blob/master/app/ws-server.py) running on port 9000. Alternatively, these could be the same codebase. The important parts are that you allow the Websocket `upgrade` header to pass through to establish the connection and that you tell nginx to listen using the `proxy_protocol`, an extra header that passes through extra information:

```
PROXY_STRING + single space + INET_PROTOCOL + single space + CLIENT_IP + single space + PROXY_IP + single space + CLIENT_PORT + singl
e space + PROXY_PORT + "\r\n"
```

This seems like it wouldn't be necessary, except that without `proxy_protocol` AWS ELBs seem to strip something important to the connection.

Next, we need to configure the load balancer. One complication here is that telling the load balancer to forward HTTPS traffic to HTTP will not work for the websockets. Instead, you have to configure it to forward TCP (SSL) to TCP. This will still work for HTTP/HTTPS traffic (as HTTP is just a specific protocol over TCP and HTTPS is just HTTP with a TLS layer), but it will also allow the non-HTTP websocket traffic to pass through successfully. Something like this:



(//blog.jverkamp.com/2015/07/20/configuring-websockets-behind-an-aws-elb/configure-elb.png)

(Don't forget to set the certificate :))

Finally, you have to configure the ELB also to speak proxy protocol. This part is slightly more annoying, since (at least now), there's no way to configure this through the AWS console. You have to use the AWS CLI (https://aws.amazon.com/cli/).

First, create the new policy (assuming you have an environment variable `ELB_NAME` defined):

```
aws elb create-load-balancer-policy \
    --load-balancer-name $ELB_NAME \
    --policy-name $ELB_NAME-proxy-protocol \
    --policy-type-name ProxyProtocolPolicyType \
    --policy-attributes AttributeName=ProxyProtocol,AttributeValue=True
```

Then, attach it to the load balancer. You will have to run this once for each port that the instance is listening on:

```
aws elb set-load-balancer-policies-for-backend-server \
    --load-balancer-name $ELB_NAME \
    --instance-port 80 \
    --policy-names $ELB_NAME-proxy-protocol
```

Make sure that you're using `https://` for the web traffic and `wss://` for the websocket and you're golden. Encrypted websockets behind an AWS ELB. Now if only they would expose the proxy protocol options in the console...

14 Comments jverkamp.com Login

Recommend 1 Share Sort by Best

Join the discussion...

Naveen • 4 months ago
I am having the same setup.
but in nginix i am not getting the client ip correctly it is always the elb ip
Reply • Share ›

gagan • 4 months ago
We are facing similar Kind of issue. We have configured two instances behind the load balancer and Socket.Io is not working if we access the site through elb endpoint but on checking individual instance ip socket connection is working fine among both the instances. Above solution is little bit confusing. Can someone type the complete answer here ?

If possible can you share your contact # JP ?
Reply • Share ›

Arlindo Santos • 9 months ago
Where is your SSL termination occurring in this case? I don't think SSL termination can occur at the AWS ELB because its using the TCP transport layer? Is your nginx doing it?
Reply • Share ›

JP Verkamp Mod ➚ Arlindo Santos • 9 months ago
The ELB is doing it. When you set up a listener from SSL (Secure TCP) to TCP it will do the termination.
Reply • Share ›