Nginx with dynamic upstreams

Posted on July 26, 2015

I recently made a setup at work where I had a Nginx server facing the user, which would forward requests to a service running behind an AWS Elastic Load Balancer (aka. ELB). That in itself doesn't sound like a difficult task, you just find the hostname for the ELB and point Nginx at it with a proxy_pass statement like this, right?

```
location / {
    proxy_pass http://service-1234567890.us-east-1.elb.am
}
```

Test it out and - barring the firewall/security group configuration is set up right - it should work just fine. Some hours later you may however find the service is no longer working, even though no changes has been made. Calling the ELB endpoint directly works just fine, but calling it through Nginx just times out.

ELB endpoint primer

In order to understand why the service suddenly stopped working, a quick introduction to how the ELB works is in order:

When you create an Elastic Load Balancer you will get a DNS record back, which AWS tell you to use for all access to the service. The DNS record is a round robin DNS record pointing at two or more IP addresses - depending on how many availability zones your service makes use of. The DNS record is set up with a 60 second time to live, meaning that there's almost no caching of the record.

The short TTL allows AWS to quickly change the machines running the load balancer without having to do any elaborate virtual IP stuff in order to not disrupt service. That's also the reason they specifically tell you to not look up the hostname and then send traffic to one of the IP addresses it returns, since that IP address will stop working as a load balancer for your service some undefined time in the future.

Back to Nginx

The reason this is a problem with Nginx is because when it sees a configuration like the one posted further up, it will do the DNS request for the hostname right away, and then use the result of that until the next time the configuration is reloaded. Before that time comes around the ELB may have changed IP addresses, leaving you with a Nginx that forwards requests to some addresses that no longer serve your service.

Nginx Plus

One way to solve this problem is to pay for Nginx Plus which adds the `resolve` flag to the `server` directive in an `upstream` group. That will make Nginx honour the TTL of the DNS record and occasionally re-resolve the record in order to get an updated list of servers to use.

Playing $1.500 per year per server for this feature seems like an awful lot. Of course you also get the other features Nginx Plus brings, but if you don't need them it becomes a prohibitively expensive upgrade.

The free alternative

A much cheaper option is to write the configuration like this:

```
resolver 172.16.0.23;
set $upstream_endpoint http://service-1234567890.us-east-
location / {
    proxy_pass $upstream_endpoint;
}
```

This will work and Nginx will honour the TTL of the DNS record and re-resolve it in case a request comes in and the cached entry has expired. But why is that?

The answer is in part found by the end of the documentation for the `proxy_pass` directive which states:

> *A server name, its port and the passed URI can also be specified using variables:*
>
> ```
> proxy_pass http://$host$uri;
> ```
>
> *or even like this:*
>
> ```
> proxy_pass $request;
> ```
>
> *In this case, the server name is searched among the described server groups, and, if not found, is determined using a resolver.*

We are basically making use of the altered behaviour when we provide `proxy_pass` with a variable, but that does however require us to specify a DNS resolver in the configuration. The one used in my example should work for all servers inside AWS which run in either the default VPC or in EC2-Classic. You can always check against `/etc/resolv.conf` to figure out which DNS server AWS has provided your server with and then use that.

Caveat regarding the forwarded URI

If the `location` you set up in Nginx is not just `/`, then you need to be aware of `proxy_pass`'s slightly changing behaviour when given a variable as the parameter.

First things first, a quick recap of how `proxy_pass` works during normal operation:

Normal behaviour

Imagine we have an Nginx configuration containing this:

```
location /foo/ {
    proxy_pass http://127.0.0.1:8080;
}
```

When you make a request to the site for `/foo/bar/baz` then Nginx will forward the request to `http://127.0.0.1:8080/foo/bar/baz`. If the configuration instead looked like this:

```
location /foo/ {
    # Note the trailing slash      ↓
    proxy_pass http://127.0.0.1:8080/;
}
```

Then Nginx will strip the part of the URI specified in the `location` directive and pass the rest on to the upstream server. A request to `/foo/bar/baz` will thus be forwarded to `http://127.0.0.1:8080/bar/baz`.

Changed behaviour

When we use a variable as the parameter for `proxy_pass` the behaviour shown above with the trailing slash changes. Say we have this configuration:

```
resolver 172.16.0.23;
set $upstream_endpoint http://service-1234567890.us-east-
location /foo/ {
    proxy_pass $upstream_endpoint;
}
```

When you make a request for `/foo/bar/baz` for that configuration, then the forwarded request will instead go to `/` and not `/bar/baz` as expected.

The workaround for this is to remove the trailing slash from the upstream endpoint, and then rewrite it manually like this:

```
resolver 172.16.0.23;
set $upstream_endpoint http://service-1234567890.us-east-
location /foo/ {
    rewrite ^/foo/(.*) /$1 break;
    proxy_pass $upstream_endpoint;
}
```

When you then make a request to `/foo/bar/baz` then the upstream will get a request for `/bar/baz` like we wanted.

Closing words

Just to make it clear, this doesn’t only affect setups using an ELB as an upstream server, but applies to any configuration where you use a changing DNS record as your upstream server in Nginx.

I hope this was useful to you. In case you have any comments to this guide or just want to get in touch with me, then find my on Twitter as @Tenzer.