Roll your own Ngrok with Nginx, Letsencrypt, and SSH reverse tunnelling

Posted on January 29, 2019

Ngrok is a fantastic tool for creating a secure tunnel from the public web to a machine behind NAT or a firewall. Sadly, it costs money and it's proprietary. If you're a developer, odds are that you're already renting a server in the public cloud, so why not roll your own ngrok?

It turns out that you can do it using free, off-the-shelf tools, with no sophisticated scripting required! In this article, I'll show you how.

Step 1. Configuring Nginx

Use a server block like this, so that incoming HTTP connections to `tunnel.yourdomain` are reverse proxied into the application listening on port `3333`.

```
server {
    server_name tunnel.yourdomain;

    access_log /var/log/nginx/$host;

    location / {
            proxy_pass http://localhost:3333/;
            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_set_header X-Forwarded-Proto https;
            proxy_redirect off;
    }

    error_page 502 /50x.html;
    location = /50x.html {
            root /usr/share/nginx/html;
    }
}
```

With this configuration in place, suppose I visited `tunnel.yourdomain`. Nginx will receive the connection, and see that it should reverse proxy it. It will effectively pass the connection on to whatever application is listening on port `3333`. Currently, there is nothing listening on this port, so we will get a `502 Bad Gateway` or `404 Not Found` error from Nginx.

Let's fix that.

Step 2. Using an SSH reverse tunnel

SSH reverse tunnelling port `N` to port `K` means making sshd listen on port `N` and effectively transfer incoming connections over the SSH connection to the SSH client. The SSH client will then transfer the connection to the application listening on port `K` on the client machine.

Here's the command to run on your client machine: `ssh -R N:localhost:K yourdomain`

An interactive session on your server should begin; while it is open, the reverse tunnel from port `N` to port `K` is active, and sshd will allow connections originating only from `localhost`, i.e. your server.

Choosing `N` = `3333` will make it so Nginx reverse proxies incoming connections on `tunnel.yourdomain` into sshd, over the SSH connection, and into the application running on your local machine on port `K`.

To test this out, on your local machine, in one shell run `python -m http.server 8888` and in another shell run `ssh -R 3333:localhost:8888 yourdomain`. Visit `tunnel.yourdomain`. You should see a directory listing for whatever directory you were in when you ran the Python command!

However, there's a glaring problem with this setup.

Step 3. Securing the connection in the browser

The connection the browser is making to Nginx is at the moment not secure: it was a plain HTTP connection. You can fix this by obtaining a free TLS certificate with Letsencrypt, and using it to secure the connection the browser is making

by obtaining a free TLS certificate with Letsencrypt, and using it to secure the connection the browser is making.

There are already excellent tutorials available on setting up Letsencrypt, so I won't repeat that here. I recommend consulting the ArchWiki article here. Letsencrypt is a self-hosters dream-come-true since it is truly a set-it-and-forget-it type of thing. With the appropriate setup, (namely a simple systemd timer,) the certificate you get will renew itself when it its expiry is approaching.

Once you have a certificate, it suffices to adjust the Nginx server block above so it looks like this.

```
server {
    server_name tunnel.yourdomain;

    access_log /var/log/nginx/$host;

    # These three lines are new.
    listen 443 ssl;
    ssl_certificate /path/to/tls/cert/fullchain.pem;
    ssl_certificate_key /path/to/tls/cert/privkey.pem;

    location / {
            proxy_pass http://localhost:3333/;
            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_set_header X-Forwarded-Proto https;
            proxy_redirect off;
    }

    error_page 502 /50x.html;
    location = /50x.html {
            root /usr/share/nginx/html;
    }
}
```

Only *three lines* need to be added!

Conclusion

With very little setup, we saw how to configure Nginx to act as a reverse proxy, and how to use an SSH reverse tunnel. By combining these off-the-shelf tools, we essentially replicated the core functionality of the fantastic tool Ngrok. Using this double-reverse-proxy technique, web applications running on a machine behind NAT or a firewall can be accessed easily and securely from a public domain or IP address.

If you have any comments or concerns, open an issue on Github.
