Building a Cro App - Part B building-a-cro-app-part-b rakudo, programming, systems

We have our Cro app built, authenticating, and doing all kinds of fun stuff. Let's take a look at deploying this bad wamma jamma!

To do so we'll do the follow:

- Create a systemd service file
- Configure apache2 to serve our app
- Set up SSL with apache2 for our service (with Let's Encrypt)

Systemd

This section is fairly straight forward but it requires some consideration:

- Systemd is not loading your user's environment
- Systemd is running as a different user than your own so your app root must have the right file permissions, by default this user is `root`

For the purpose of this article we're just going to assume you've put your app root in `/opt/my-app` , you have a working rakudo, that you have all of the dependencies for your app installed, and that the app is runnable on the machine you're installing the systemd file on.

In your services file `/etc/systemd/system/my-app.service` :

```
[Unit]
Description=My Cro App - A Tutorial
DefaultDependencies=no
After=network.target

[Service]
Type=simple
WorkingDirectory=/opt/my-app
ExecStart=/usr/bin/raku -I. /opt/my-app/bin/app
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

This is the most basic systemd file you can make, there are a lot of other options you can put in here including things like a restart backoff if the app is not starting. It is important to note that you should use the full path to binaries in the service file, make no assumptions, and take frequent breaks.

Now, run `systemctl daemon-reload` so that systemd refreshes its cache, enable our service with `systemctl enable my-app`, and finally start the service with `systemctl start my-app`.

You should now be able to hit port `8666` (if you haven't modified anything from part 1 or cloned the repo). You can test this by running `curl localhost:8666` - you should get a response from our app.

If you're having issues and think that the app is not running, you can use `journalctl -u my-app.service` to see the logs and correct any errors the app is having getting started.

That's all there really is in a basic systemd service. If you'd like to delve into service files more, this is a hand resource. Onto to the tougher stuff.

Configuring Apache

Getting the certificates with `certbot` prior to setting up apache is much easier but it's a headache when you have to renew the certs so we're going to make this work with apache running so there's no downtime when you renew.

For this tutorial we're looking at Debian 11 so some of your install commands might be a bit different but installing apache2:

```
# apt install apache2
...
# a2enmod rewrite
...
# a2enmod proxy
...
# a2enmod proxy_http
...
# systemctl restart apache2
...
```

After getting apache installed we need to configure it to respond to our domain, so in your favorite version of vim load up `/etc/apache2/sites-available/000-default.conf` and make it look exactly like:

```
<VirtualHost *:80>
 ServerName pm6.dev
 ServerAlias www.pm6.dev
 DocumentRoot /var/www

 RewriteEngine On
 RewriteCond %{HTTPS} off
 RewriteCond %{REQUEST_URI} !^/\.well-known/
 RewriteRule (.*) https://www.pm6.dev/$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
 ServerName pm6.dev
 ServerAlias www.pm6.dev
 ProxyRequests     Off
 ProxyPreserveHost On

 ErrorLog ${APACHE_LOG_DIR}/error.log
 CustomLog ${APACHE_LOG_DIR}/access.log combined

 <Proxy *>
  Order deny,allow
  Allow from all
 </Proxy>
 ProxyPass         /  http://localhost:8666/ nocanon
 ProxyPassReverse  /  http://localhost:8666/
</VirtualHost>
```

Now we're ready to get certbot involved. The configuration above is using the hostname `pm6.dev` , you should replace that with whatever hostname you're looking to use. The other thing it does is redirects standard `http` requests to `https` *unless* the request is from certbot.

```
# apt install certbot
...
```

Now we're ready to set up the certificates, grab that shiny domain name and replace `{EMAIL}` and `{DOMAIN}` with your own!

```
# systemctl restart apache2
...
# certbot certonly --webroot -w '/var/www/' -d 'www.{DOMAIN}' -d '{DOMAIN}' -n --email
'{EMAIL}' --agree-tos
```

So far we're looking fresh, if certbot succeeded then we're set up for a hands free renewal in the future. Let's put the certs to use by editing our apache2 config again in notepad++:

```
# ...
<VirtualHost *:443>
   ServerName pm6.dev
   ServerAlias www.pm6.dev
   ProxyRequests     Off
   ProxyPreserveHost On

   SSLCertificateFile /etc/letsencrypt/live/www.pm6.dev/fullchain.pem
   SSLCertificateKeyFile /etc/letsencrypt/live/www.pm6.dev/privkey.pem

   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined

# ...
```

The two new lines are the ones starting with `SSLCertificate` . Enable `ssl` on apache2 and then restart apache2:

```
# a2enmod ssl
...
# systemctl restart apache2
```

Voila! That's it. That's all there is to it. If apache complains here then correct any errors and then restart this paragraph.