SET UP YOUR OWN PRERENDER SERVICE

WITH AWS OPSWORKS

INTRODUCTION – JACOB SENECAL

- Senior Developer at PartCycle Technologies
- I work primarily on backend / Rails development
- Lots of infrastructure and process automation
- We also use Heroku, Ember, Postgres…

PARTCYCLE
QUALITY RECYCLED AUTO PARTS

github.com/jakesen

WHAT IS AWS OPSWORKS

- Amazon Web Services' tool for managing applications and servers
- It allows you to configure a Stack comprised of Layers, Apps and Instances
- The Stack can be configured using Chef recipes and cookbooks
- The instances are EC2 (Elastic Compute Cloud) virtualized servers

WHAT IS PRERENDER

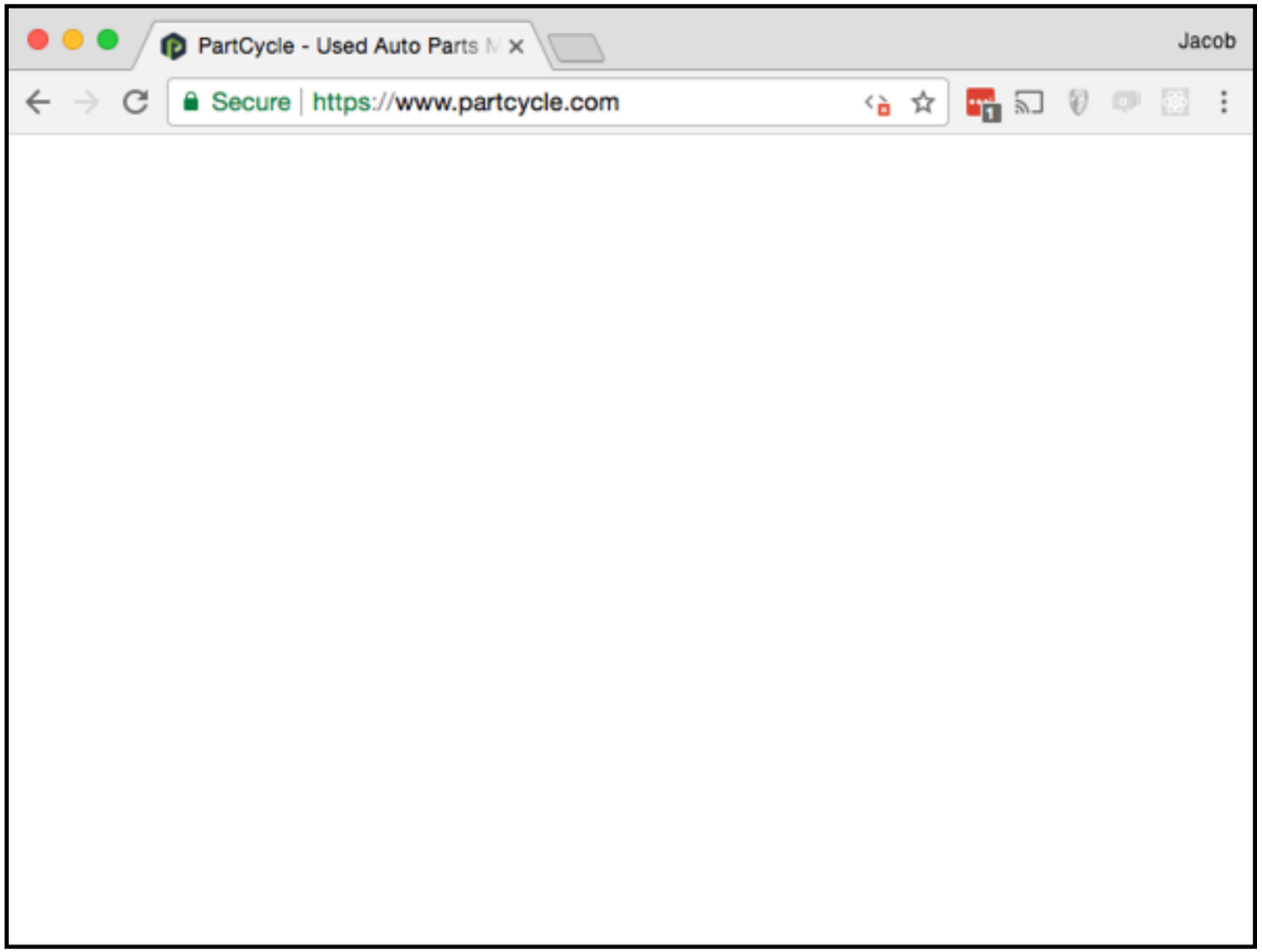
- Open source Node.js application for converting JS rendered pages to pure HTML on demand
- It uses Phantom.js to render pages in Chromium
- It uses the prerender middleware to intercept requests from search bots (available for Rails, Express, etc)
- You can also use the paid service at Prerender.io
- More info and documentation at http://github.com/prerender/prerender

WHAT WE'RE GOING DO

- Create a github repo containing our customized prerender app
- Create a deploy key
- Set up an OpsWorks stack in Amazon Web Services
- Deploy and test our app!
- Demo app: https://github.com/jakesen/prerender-opsworks-demo

...AND WHY...

- Client rendered pages aren't recognized by Search bots
- While Google has been working on JS support, it doesn't always work
- Assume the worst - search bots won't run JS



OPSWORKS NODE.JS APP REQUIREMENTS

- Must have server.js
- Must have package.json
- Must listen on port 80 or 443 or env.PORT

GETTING STARTED

- Prerender suggests using a clone or fork of the prerender repo
- You can also create a new node project that imports prerender
- > npm init
- > npm install prerender –save

{} package.json ×

```
1  {
2    "name": "prerender-opsworks-demo",
3    "version": "1.0.0",
4    "description": "Demonstration of setting up Prerender for AWS OpsWorks",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "repository": {
10     "type": "git",
11     "url": "git+https://github.com/jakesen/prerender-opsworks-demo.git"
12   },
13   "author": "Jacob Senecal",
14   "license": "MIT",
15   "bugs": {
16     "url": "https://github.com/jakesen/prerender-opsworks-demo/issues"
17   },
18   "homepage": "https://github.com/jakesen/prerender-opsworks-demo#readme",
19   "dependencies": {
20     "prerender": "^4.4.1"
21   }
22 }
23
```

COPY SERVER

- Create a new file named server.js (this is the filename required by OpsWorks App Layer)
- Copy the contents of server.js from pre-render
- Ignore optional plugins (for now)

server.js

```
#!/usr/bin/env node
var prerender = require('prerender');

var server = prerender({
    workers: process.env.PRERENDER_NUM_WORKERS,
    iterations: process.env.PRERENDER_NUM_ITERATIONS
});



server.use(prerender.sendPrerenderHeader());
// server.use(prerender.basicAuth());
// server.use(prerender.whitelist());
server.use(prerender.blacklist());
// server.use(prerender.logger());
server.use(prerender.removeScriptTags());
server.use(prerender.httpHeaders());
// server.use(prerender.inMemoryHtmlCache());
// server.use(prerender.s3HtmlCache());

server.start();
```

PROBLEM: HEALTH CHECKS WON'T WORK!

- The default configuration of prerender returns 5xx errors if no render url is provided
- The default OpsWorks chef recipes configure monit to send requests to the root path do determine the health of our Node.js layer
- The load balancer also needs to have a path for health checks

CUSTOMIZE SERVER

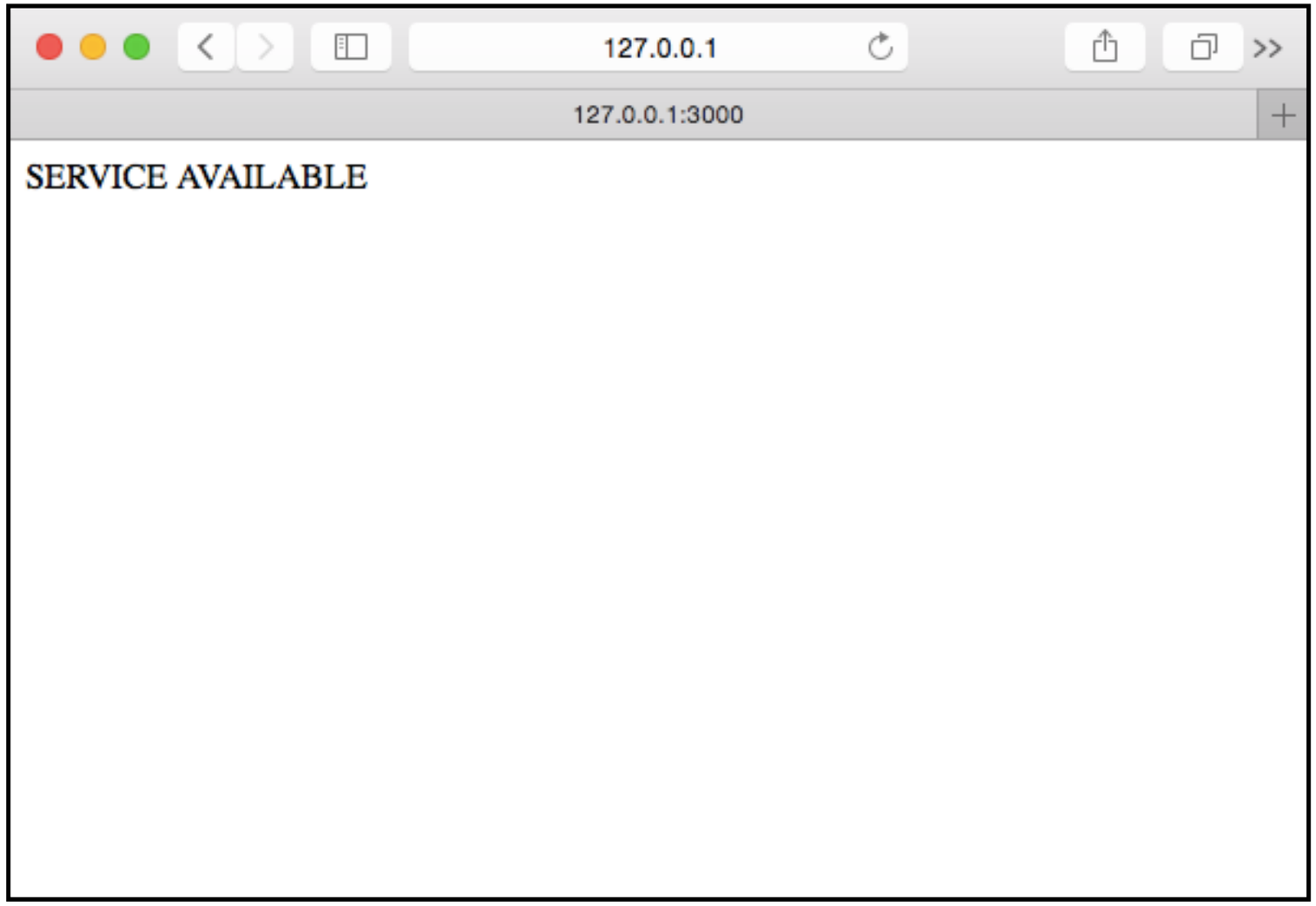
- Use the prerender **beforePhantomRequest** plugin handler to intercept requests without a url
- Return a success message and 200 status code

server.js

```js
#!/usr/bin/env node
var prerender = require('prerender');

var server = prerender({
    workers: process.env.PRERENDER_NUM_WORKERS,
    iterations: process.env.PRERENDER_NUM_ITERATIONS
});

server.use({
    beforePhantomRequest: function(req, res, next) {
        if(!req.url || req.url == '/') {
            req.prerender.documentHTML = "SERVICE AVAILABLE";
            return res.send(200);
        }
        next();
    }
});

server.use(prerender.sendPrerenderHeader());
// server.use(prerender.basicAuth());
// server.use(prerender.whitelist());
server.use(prerender.blacklist());
// server.use(prerender.logger());
server.use(prerender.removeScriptTags());
server.use(prerender.httpHeaders());
// server.use(prerender.inMemoryHtmlCache());
// server.use(prerender.s3HtmlCache());

server.start();
```

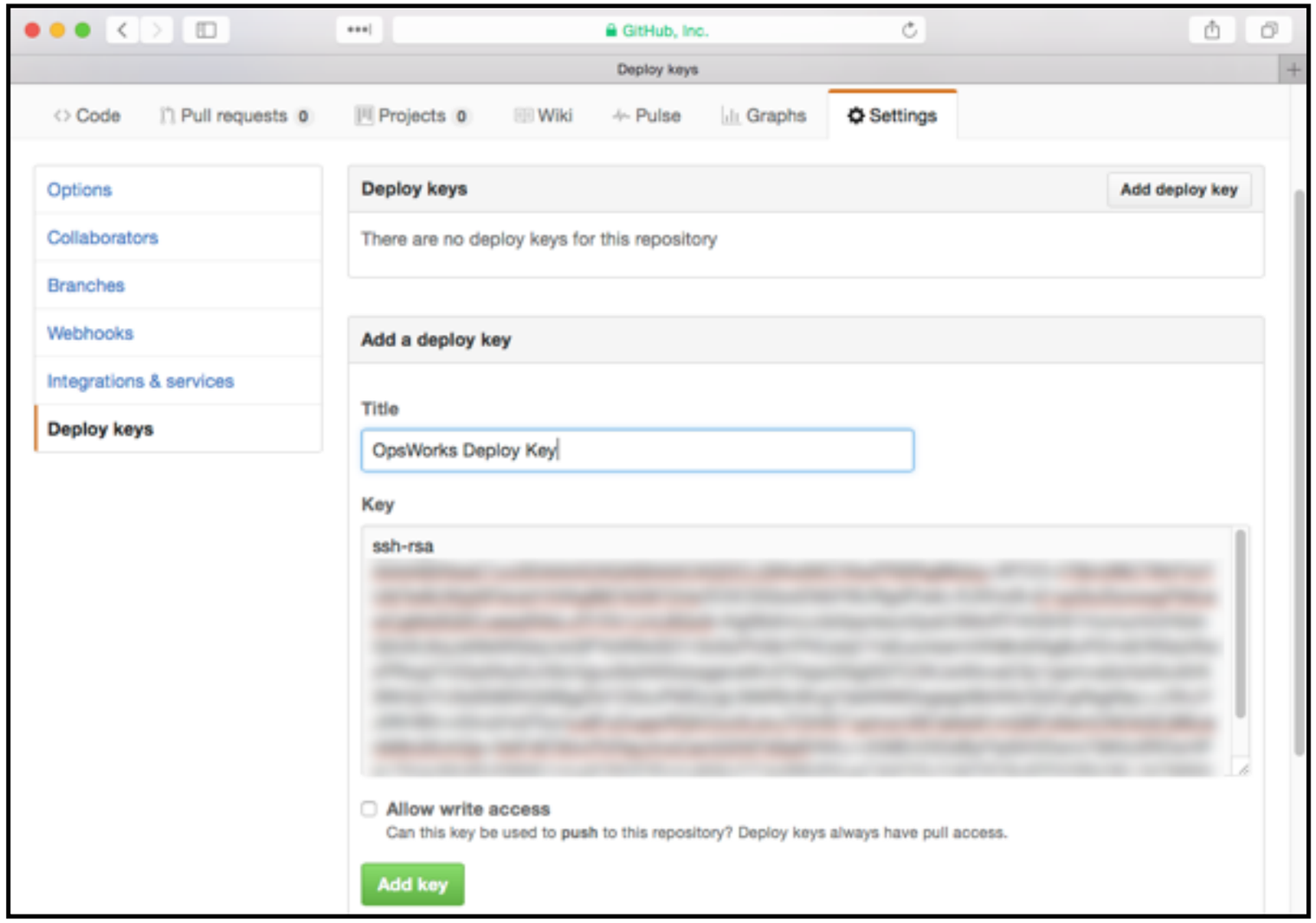
TEST LOCALLY

- > node server.js
- Open the local server in your browser
- Try adding a valid web URL as the path



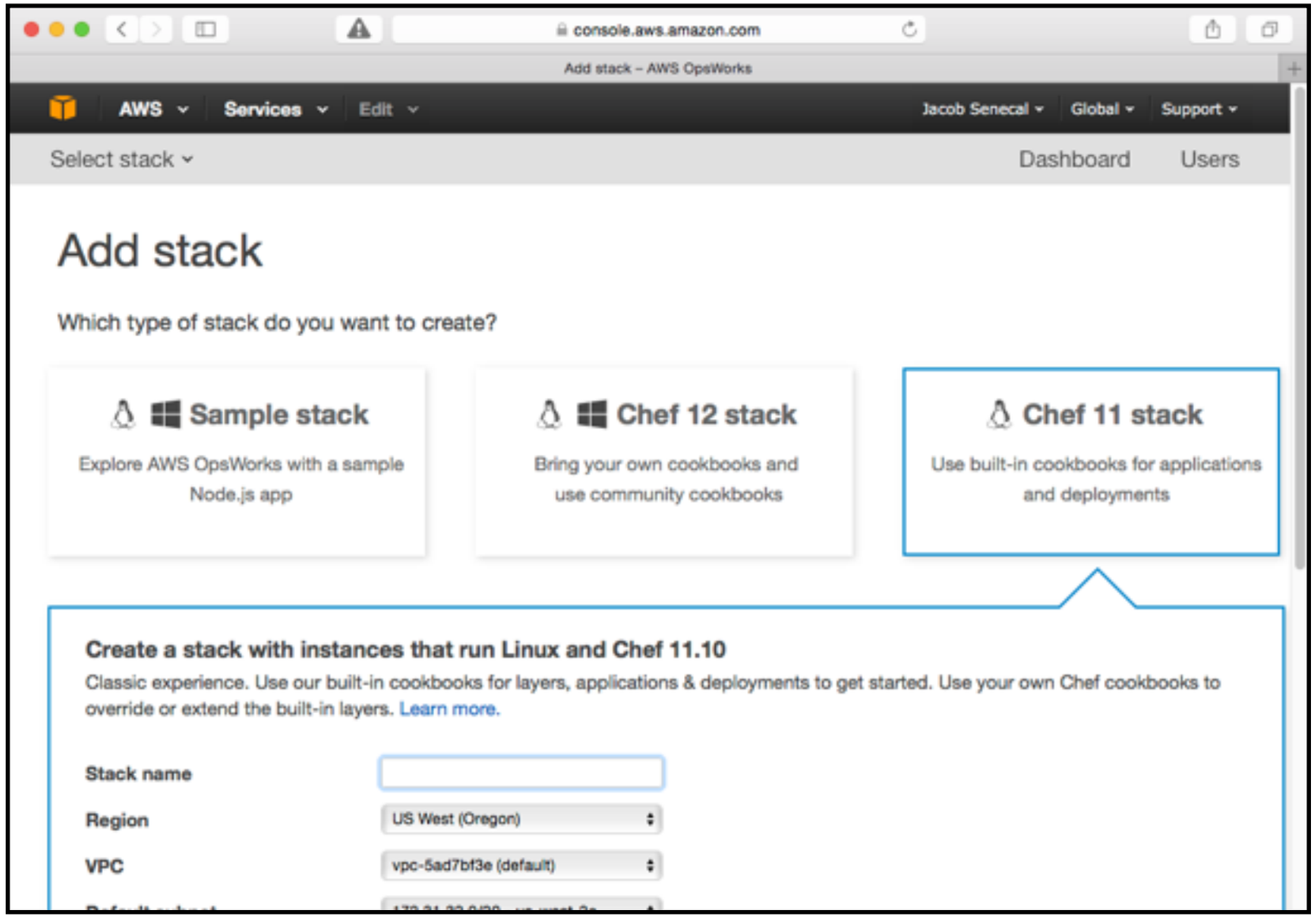
SET UP DEPLOY KEY

- Create a public/ private key pair
- Go to github repo Settings
- Add the public key to the repo's Deploy keys



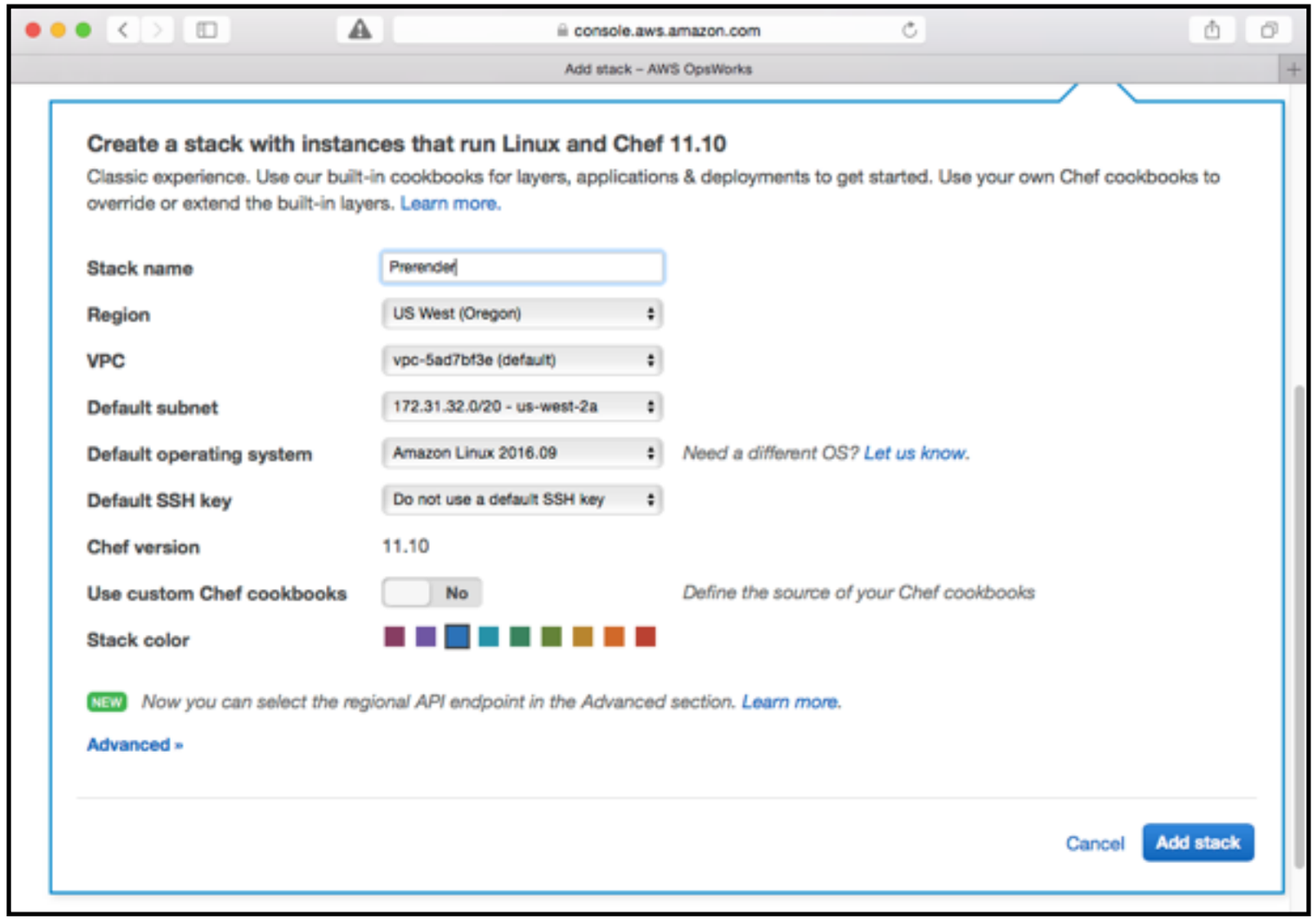
ADD OPSWORKS STACK

- Go to AWS OpsWorks dashboard
- Add stack, choose Chef 11



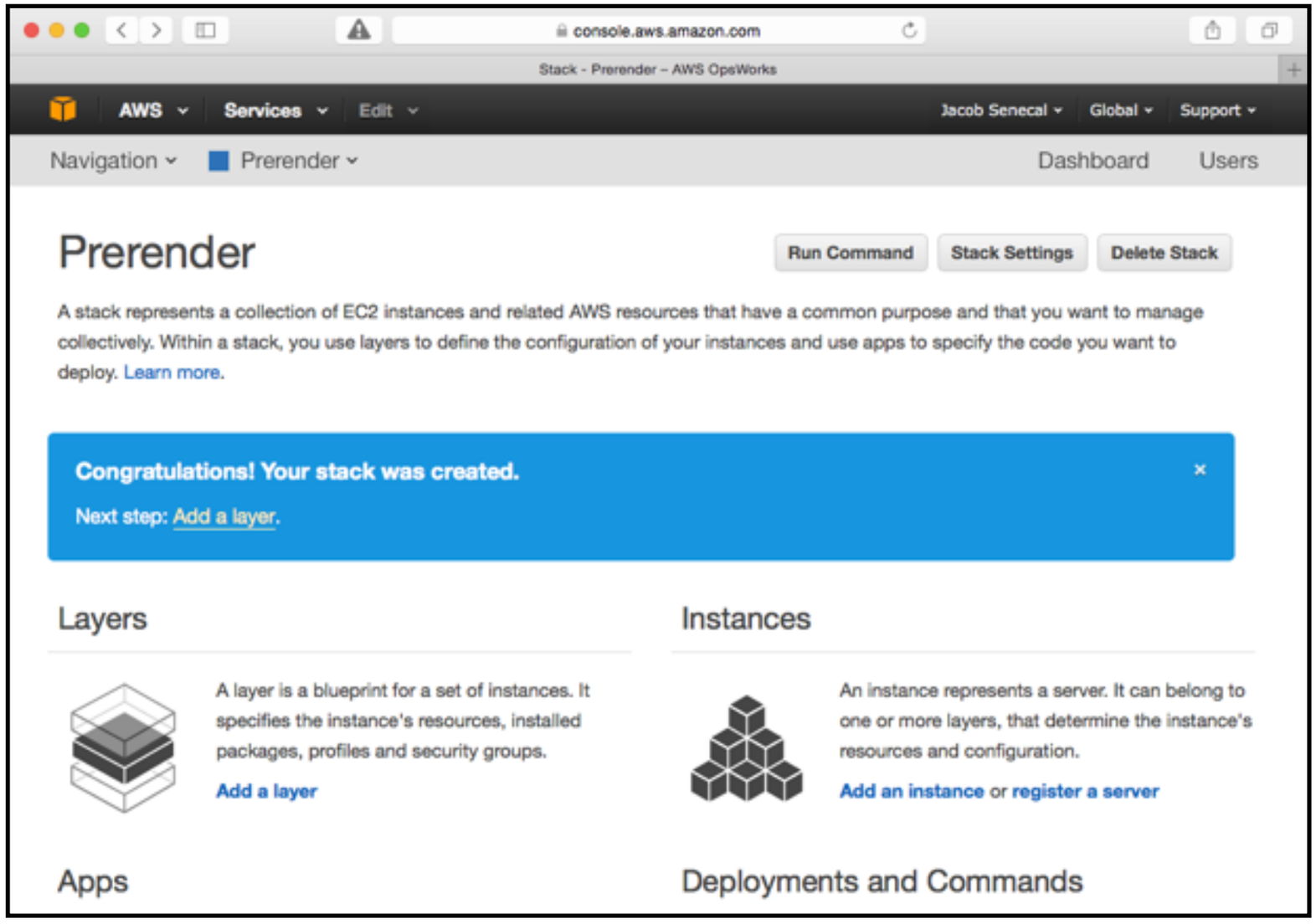
OPSWORKS STACK SETTINGS

- Give it a name
- Choose Region, VPC, etc (we'll use the defaults for this demo)



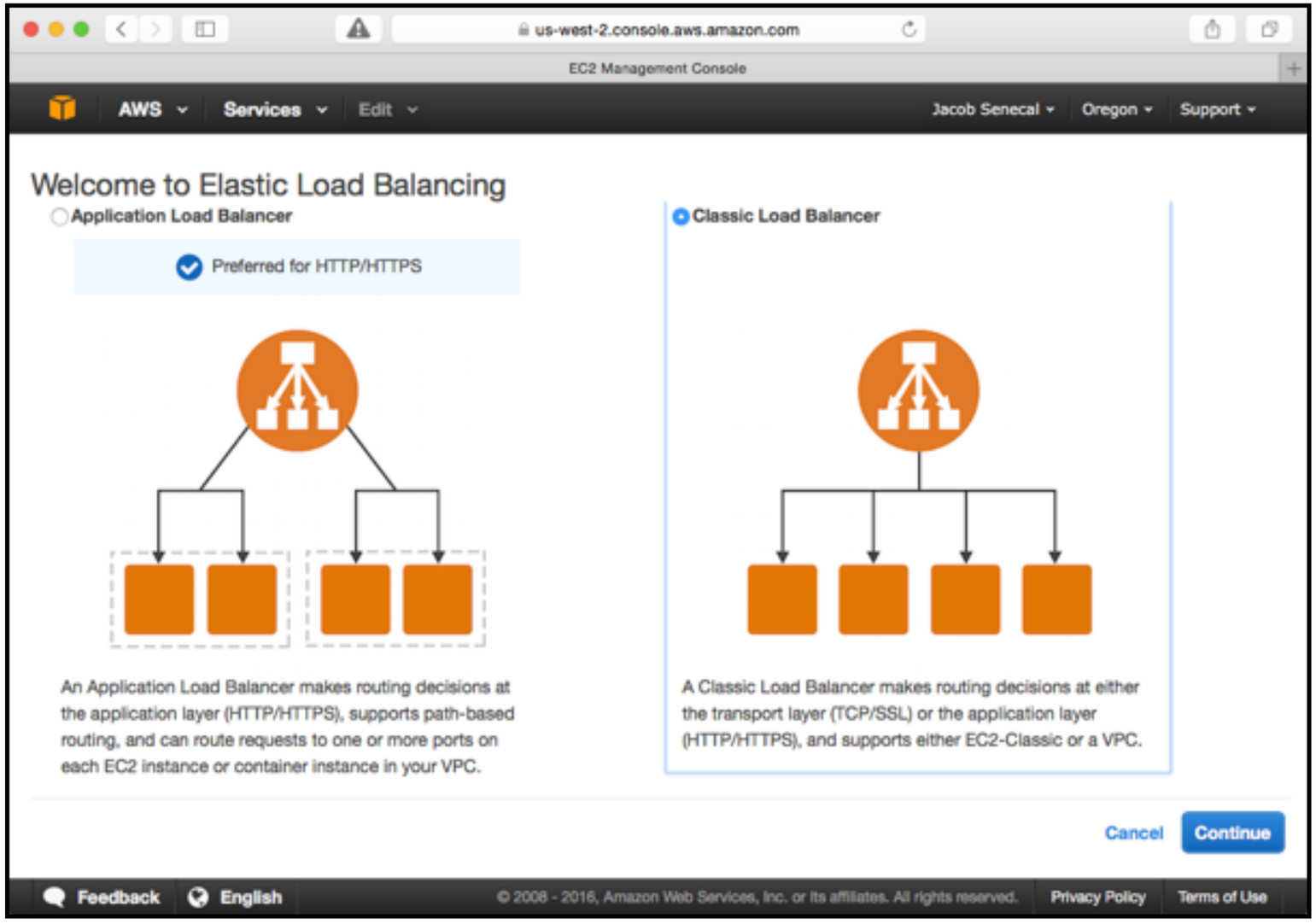
ALL DONE! (JK)

- Now we have a stack, but there's still lots to do
- Next, we need to add a layer
- But first…



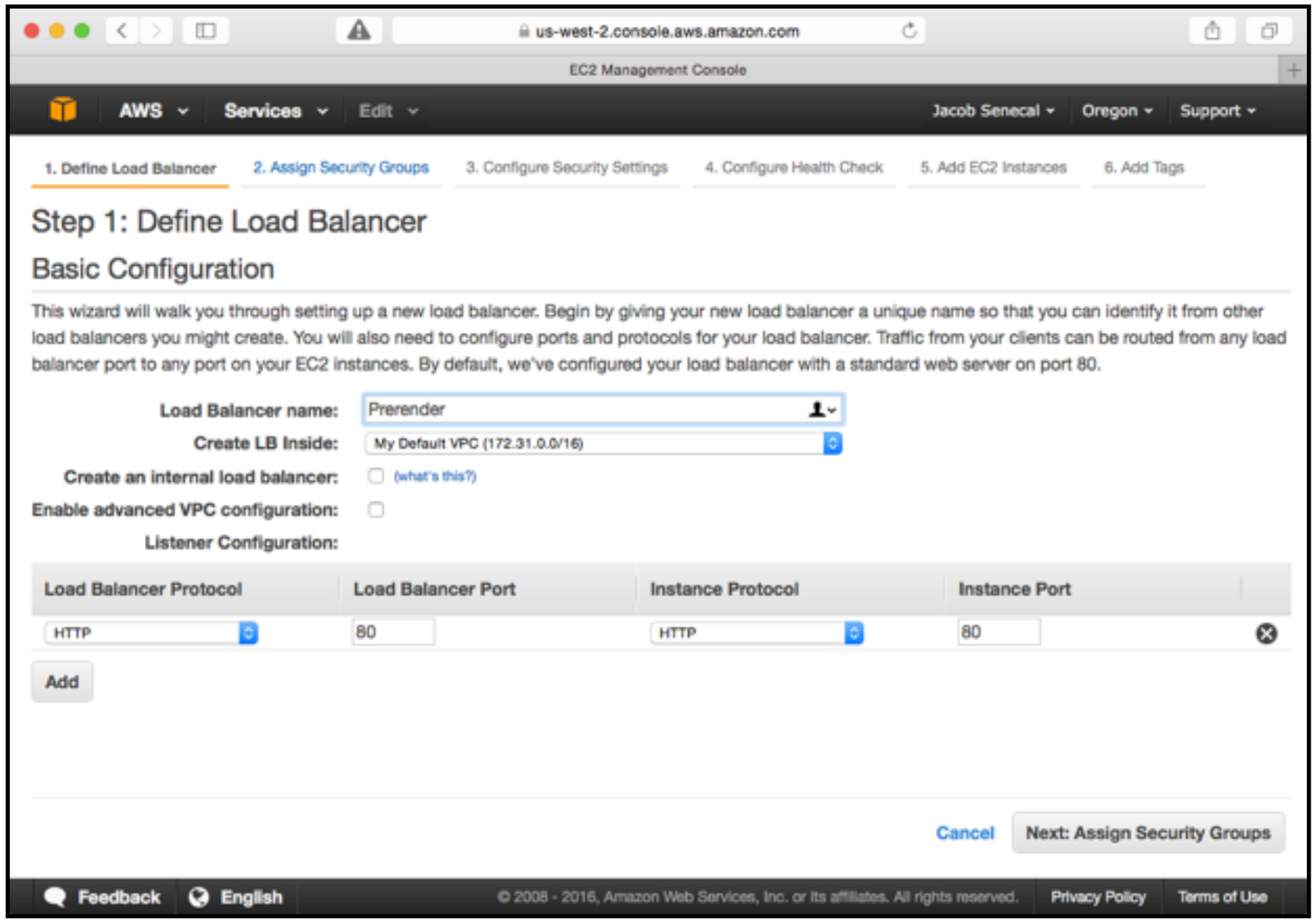
CREATE LOAD BALANCER

- Go to AWS EC2 dashboard
- Click Load Balancers from Nav menu
- Click Create Load Balancer
- Choose Classic Load Balancer



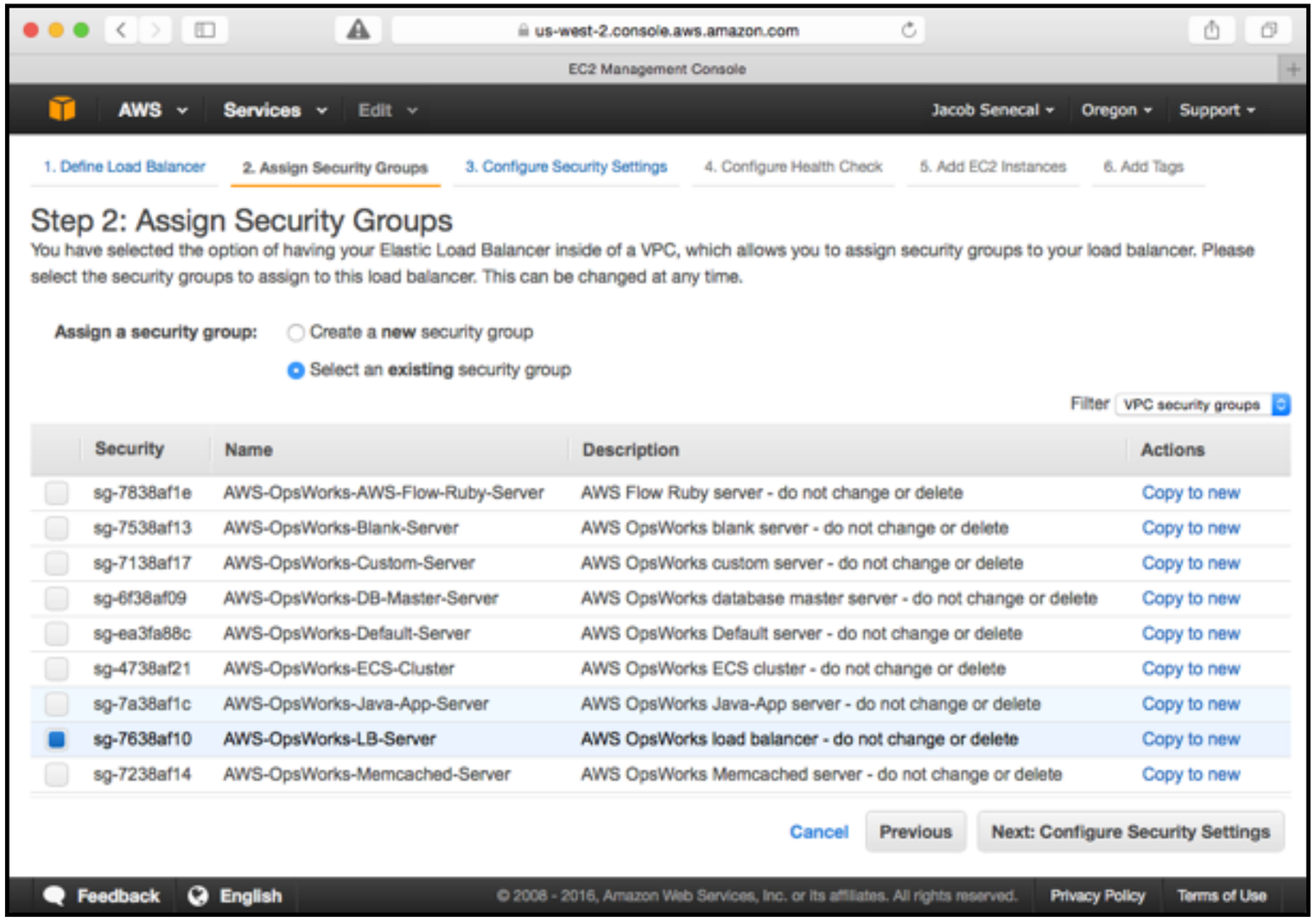
CONFIGURE LOAD BALANCER

- Give it a name
- Assign to same VPC as OpsWorks stack
- Configure protocols (Needed if you want SSL support, etc)



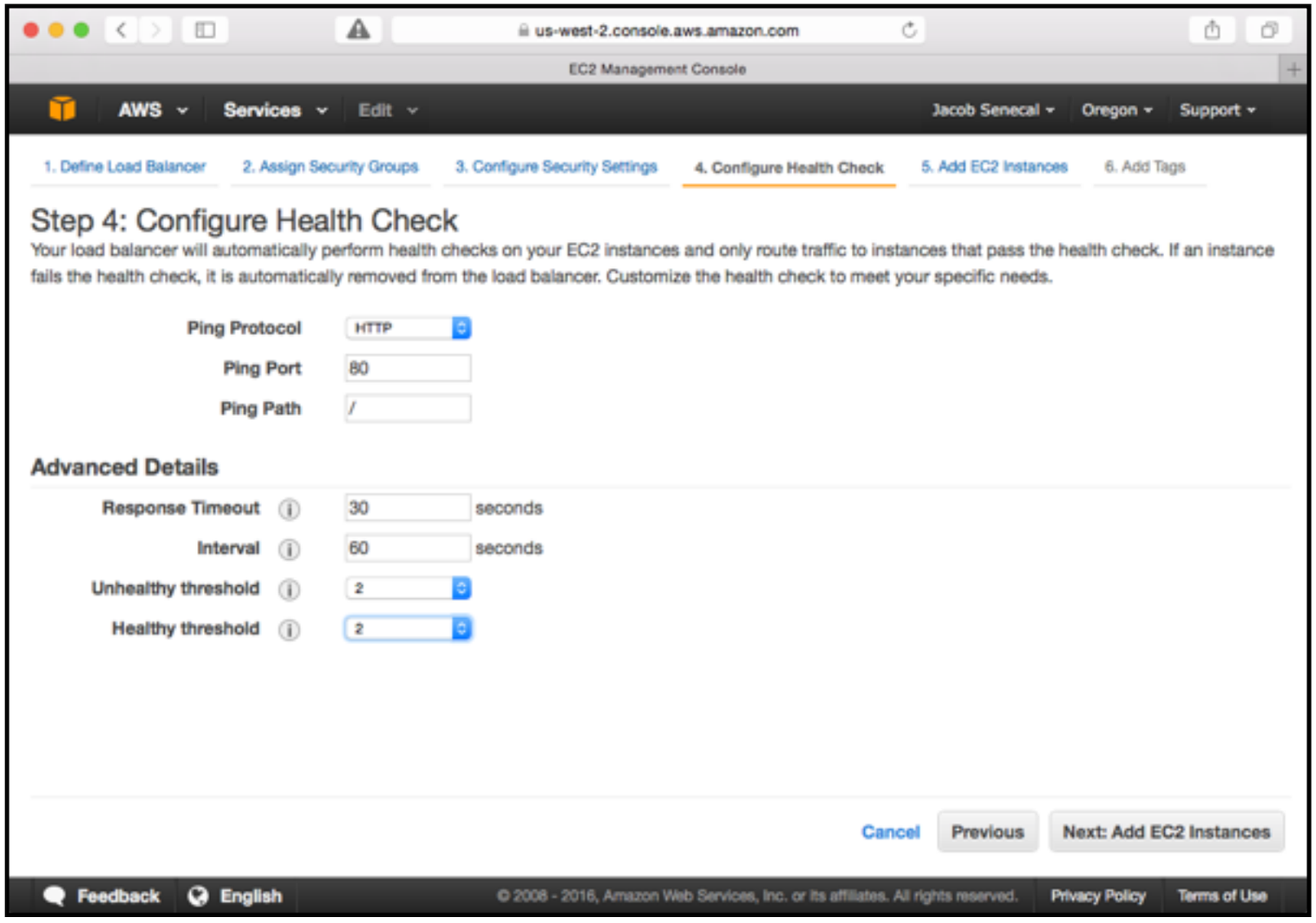
CONFIGURE LOAD BALANCER

- Assign security group (conveniently created by OpsWorks!)
- We'll skip Step 3



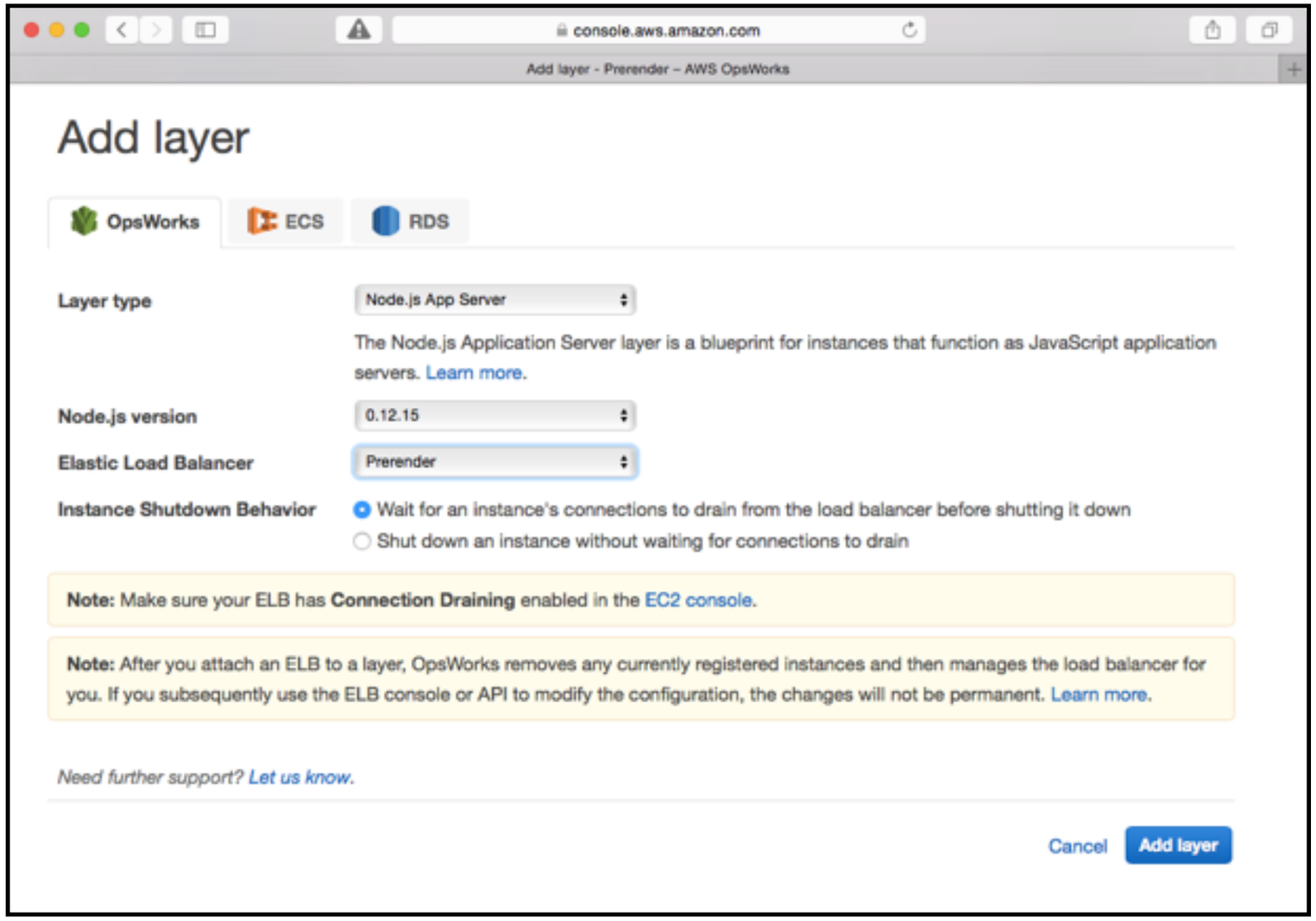
CONFIGURE HEALTH CHECK

- Ping path should be root (/)
- Ping timeout and interval should be generous
- Skip steps 5 & 6 and click Create
- Now back to our stack…



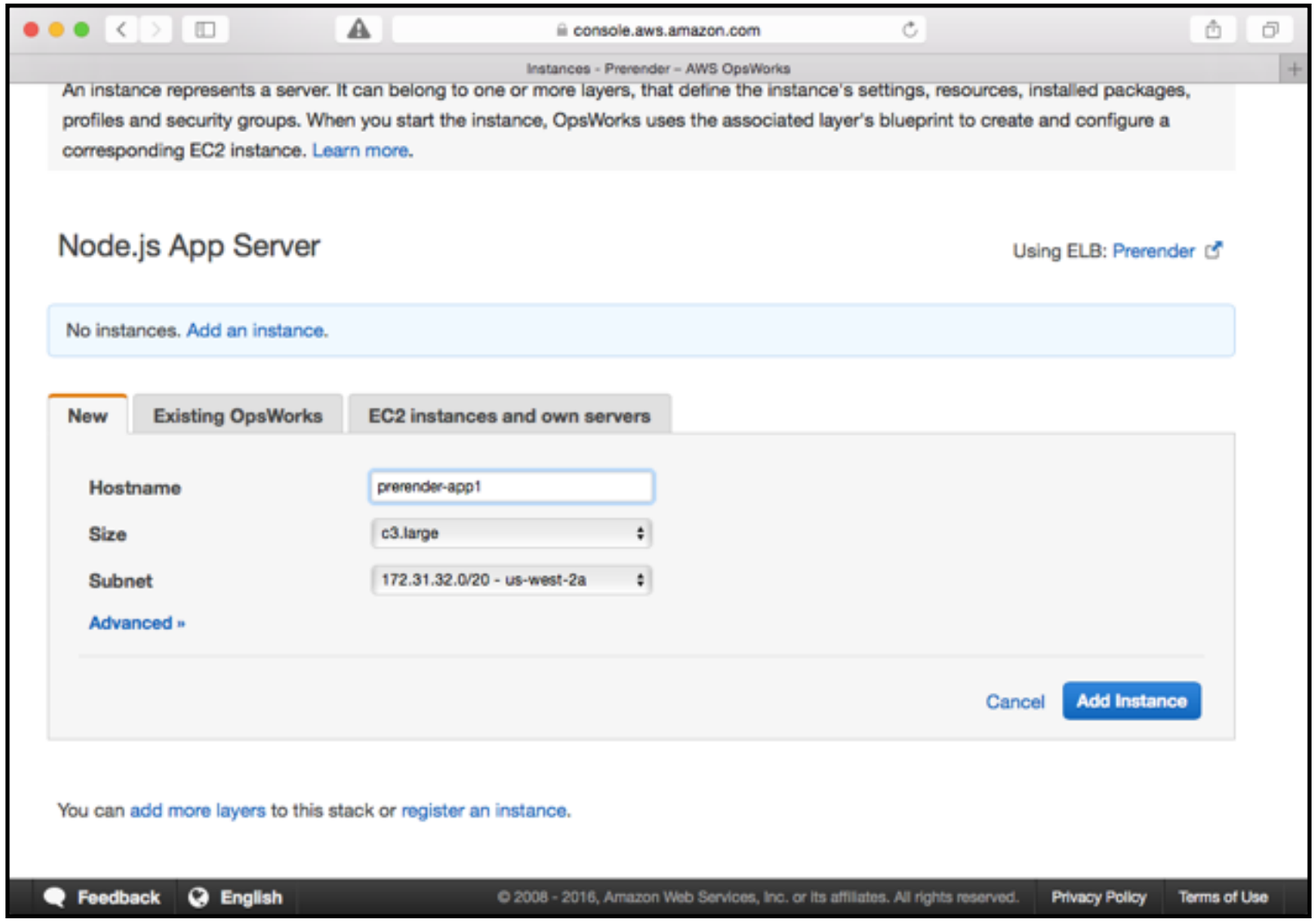
ADD NODE.JS LAYER

- Click Add Layer
- Layer type should be Node.js App Server
- Select the load balancer that was just created



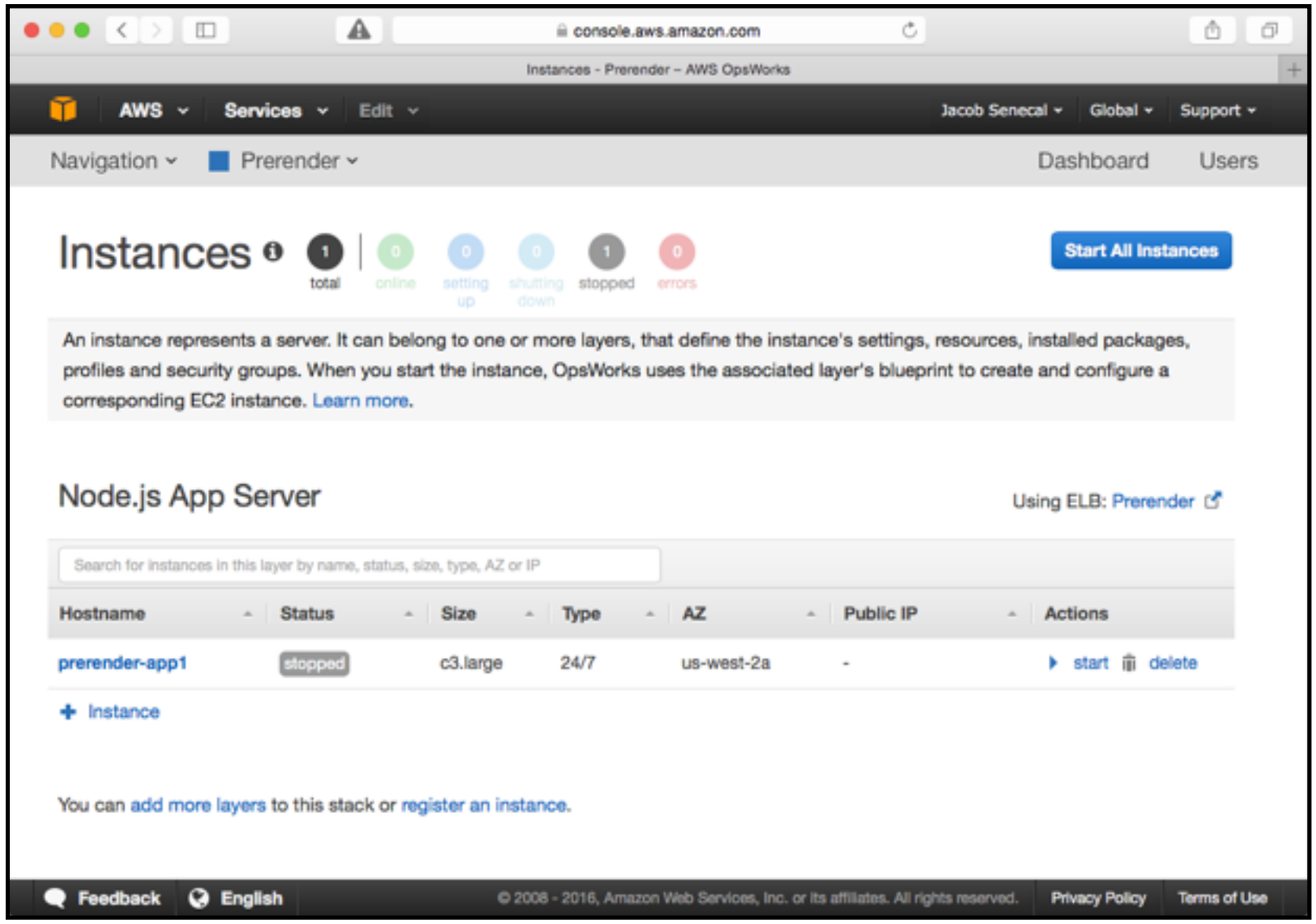
ADD NEW INSTANCE

- Click Add Instance
- Type is New (EC2)
- Set hostname and size
- Default of c3.large is a good starting point



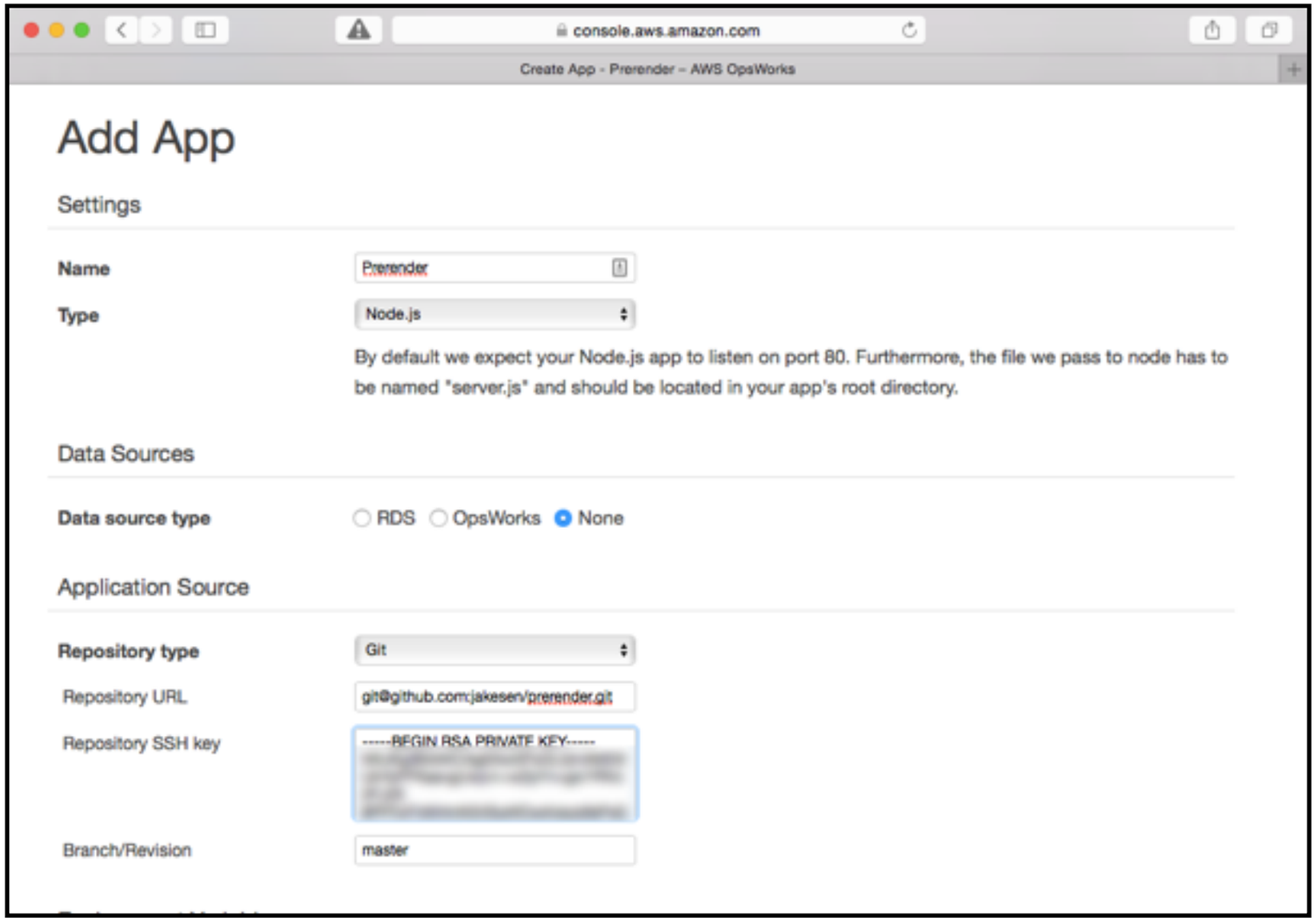
START NEW INSTANCE

- lick start to provision and boot new EC2 instance
- You'll see the status change to Requested, Booting, Running Setup, and finally Online



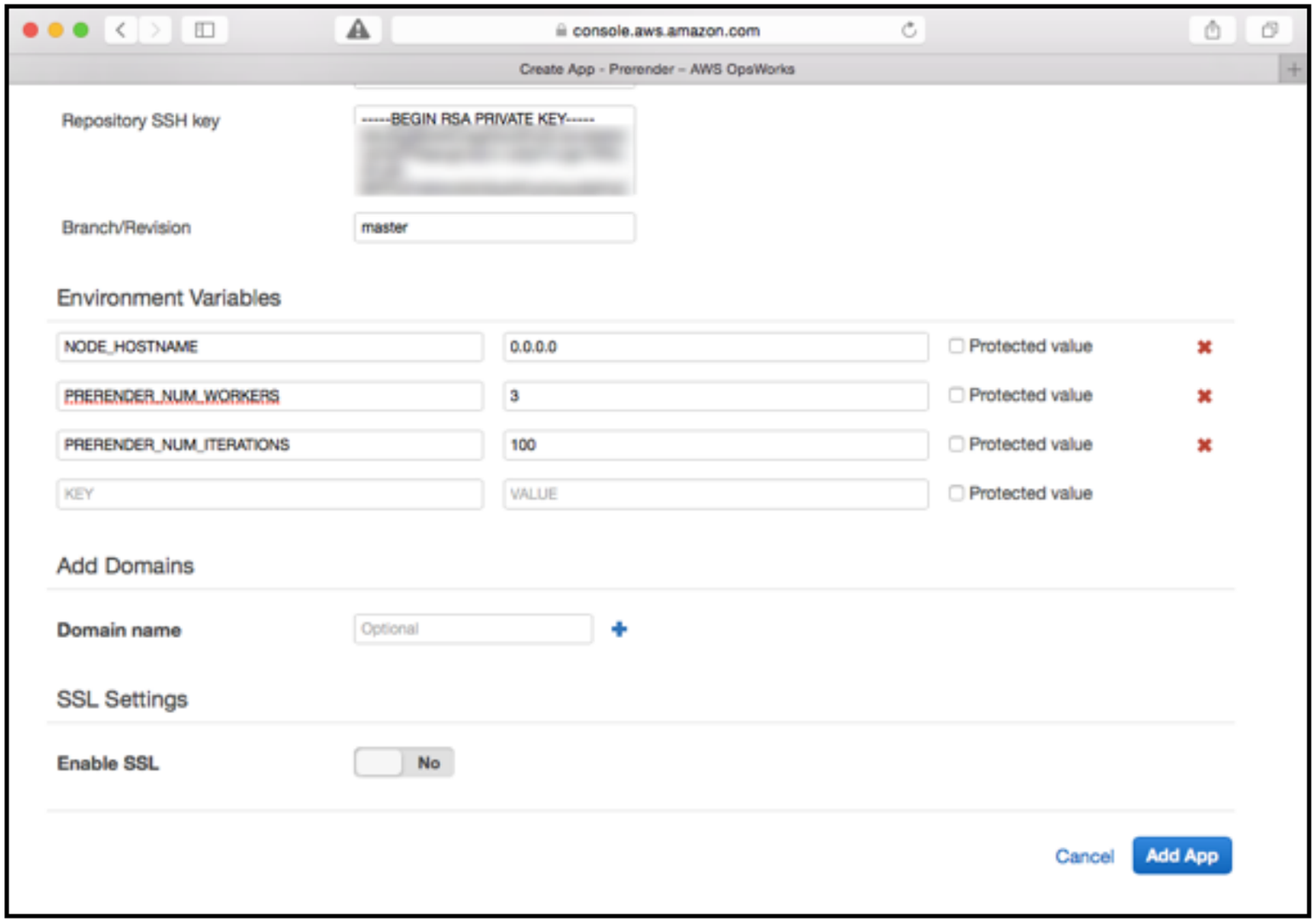
ADD PRERENDER APP

- Click Add App
- Enter name
- Choose Node.js type
- Add repo URL and private key
- Other settings can be left default



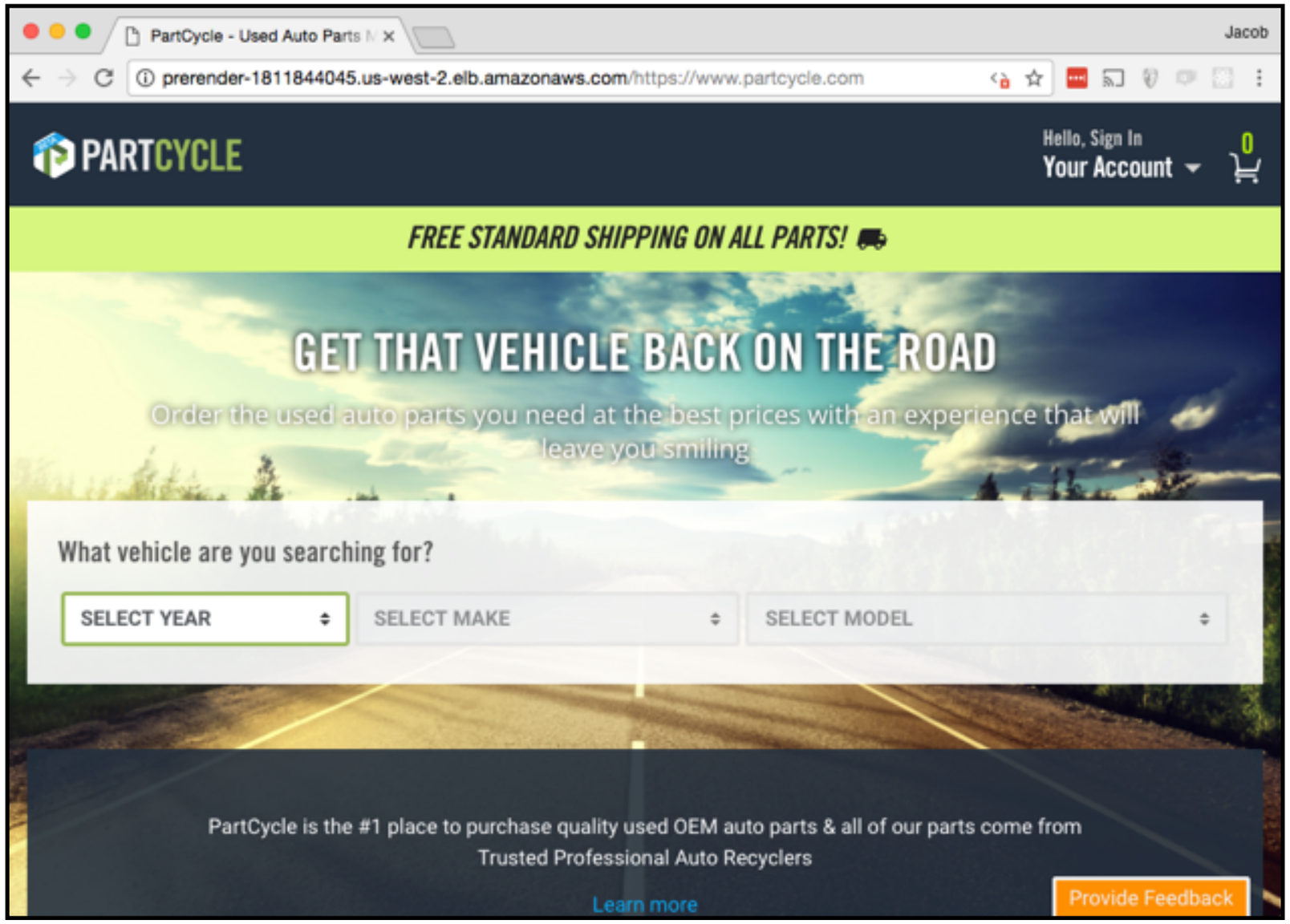
PRERENDER APP CONFIG

- Set environment variables
- NODE_HOSTNAME is required for the App to work with AWS' Node.js layer
- Prerender variables are optional
- Deploy



TESTING OUR PRERENDER STACK

- Copy the AWS load balancer endpoint address into your browser
- You should see SERVICE AVAILABLE
- Try adding a website URL
- You should see the rendered page!



RECAP

- If you need to serve your client-rendered JS site to search engines, Prerender may be just what you need
- Don't forget your middleware
- Caching / Pre-caching
- There's always the paid service: Prerender.io
- This demo can be found at https://github.com/jakesen/prerender-opsworks-demo
- PartCycle is hiring!

DevSpace would like to thank our sponsors

RESULT STACK

novation systems

Progress

CODEROCKET

COMPLETE DEVELOPER PODCAST

RED HAT OPENSHIFT Container Platform

PLURALSIGHT

Dynetics

ROCKET CITY DIGITAL

U.S. Space & Rocket Center

TOYOTA MOTOR MANUFACTURING ALABAMA

Nodevember

PHOCAZ