

SET UP YOUR OWN PRERENDER SERVICE

WITH AWS OPSWORKS

INTRODUCTION – JACOB SENECA

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



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>

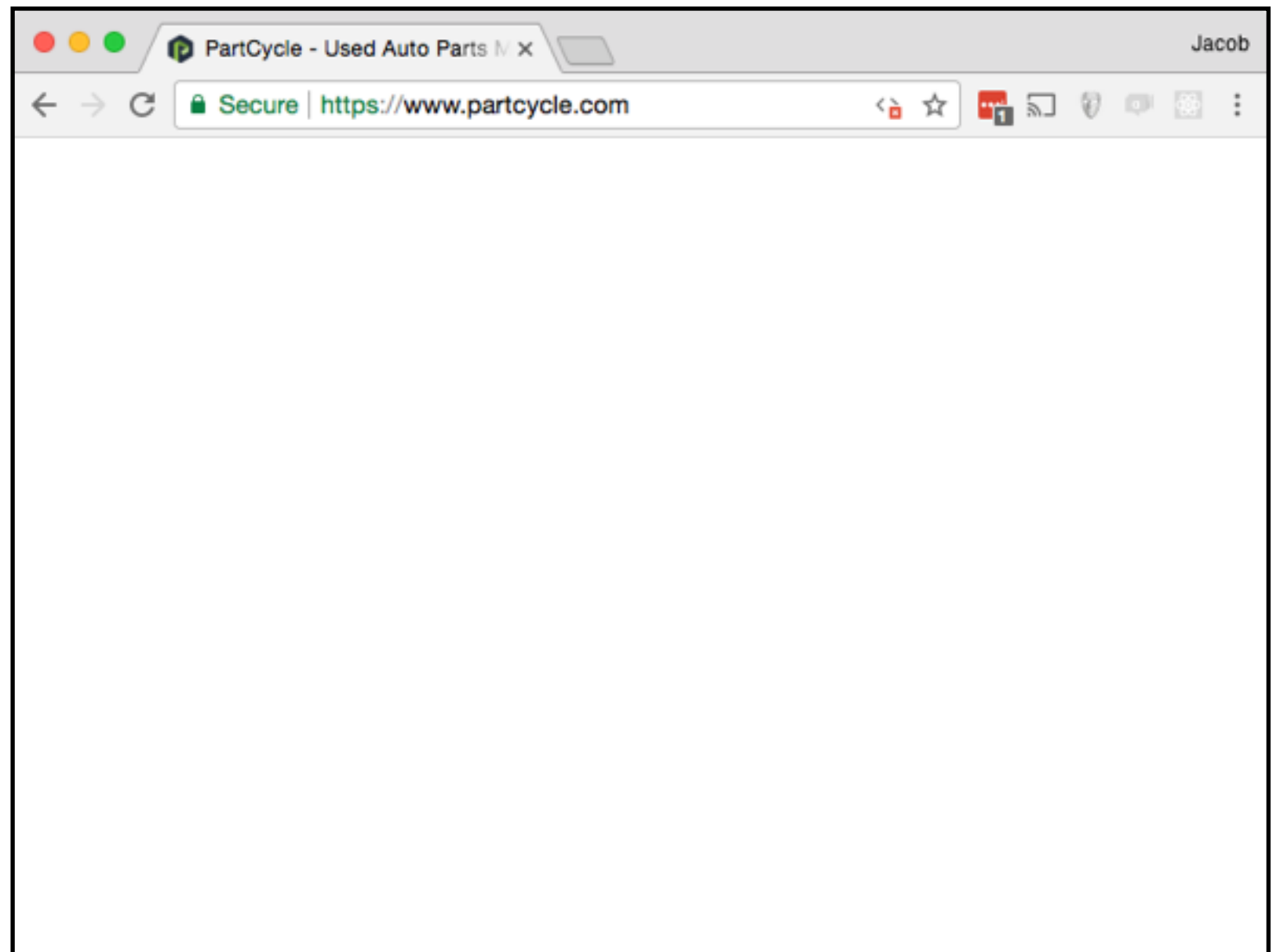
WHAT WE'RE GOING DO

- ▶ 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>

SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

...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 x
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)

```
JS server.js  x
1  #!/usr/bin/env node
2  var prerender = require('prerender');
3
4  var server = prerender({
5    workers: process.env.PRERENDER_NUM_WORKERS,
6    iterations: process.env.PRERENDER_NUM_ITERATIONS
7  });
8
9
10
11  server.use(prerender.sendPrerenderHeader());
12  // server.use(prerender.basicAuth());
13  // server.use(prerender.whitelist());
14  server.use(prerender.blacklist());
15  // server.use(prerender.logger());
16  server.use(prerender.removeScriptTags());
17  server.use(prerender.httpHeaders());
18  // server.use(prerender.inMemoryHtmlCache());
19  // server.use(prerender.s3HtmlCache());
20
21  server.start();
22
```

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 to determine the health of our Node.js layer
- ▶ The load balancer also needs to have a path for health checks

CUSTOMIZE SERVER

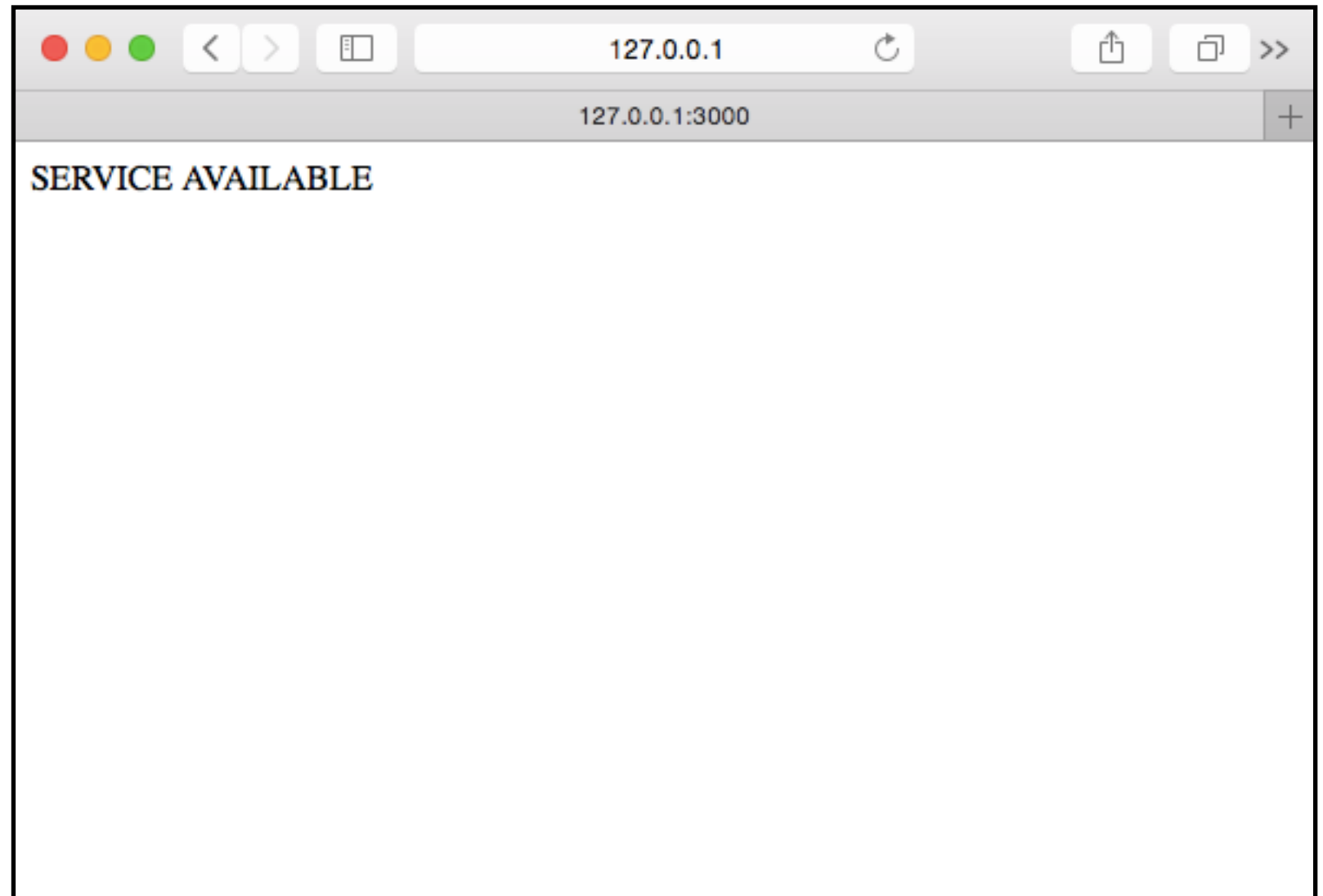
- ▶ Use the prerender **beforePhantomRequest** plugin handler to intercept requests without a url
- ▶ Return a success message and 200 status code

```
JS server.js x
1  #!/usr/bin/env node
2  var prerender = require('prerender');
3
4  var server = prerender({
5    workers: process.env.PRERENDER_NUM_WORKERS,
6    iterations: process.env.PRERENDER_NUM_ITERATIONS
7  });
8
9  server.use({
10    beforePhantomRequest: function(req, res, next) {
11      if(!req.url || req.url == '/') {
12        req.prerender.documentHTML = "SERVICE AVAILABLE";
13        return res.send(200);
14      }
15      next();
16    }
17  });
18
19  server.use(prerender.sendPrerenderHeader());
20  // server.use(prerender.basicAuth());
21  // server.use(prerender.whitelist());
22  server.use(prerender.blacklist());
23  // server.use(prerender.logger());
24  server.use(prerender.removeScriptTags());
25  server.use(prerender.httpHeaders());
26  // server.use(prerender.inMemoryHtmlCache());
27  // server.use(prerender.s3HtmlCache());
28
29  server.start();
30
```

SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

TEST LOCALLY

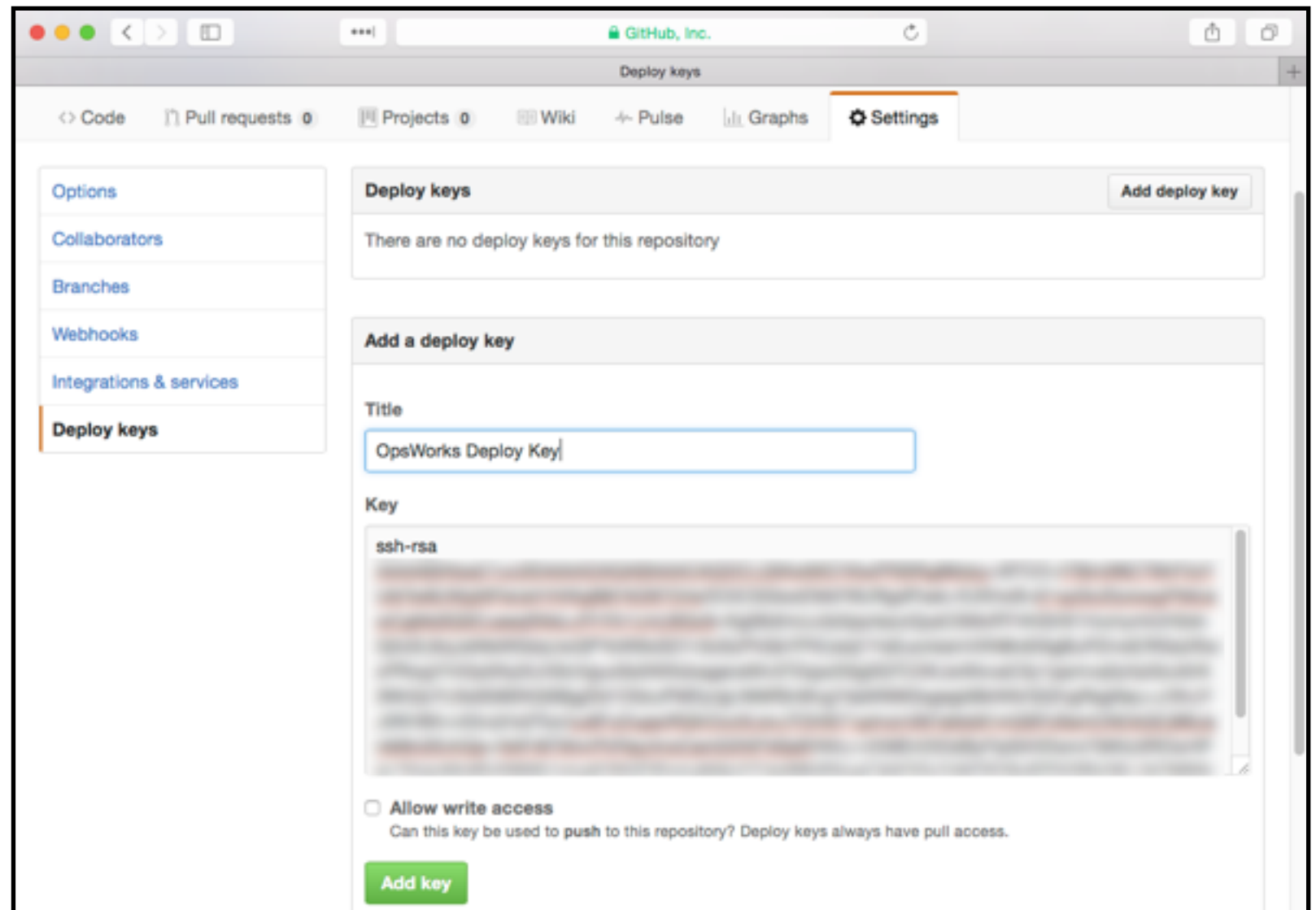
- ▶ `> node server.js`
- ▶ Open the local server in your browser
- ▶ Try adding a valid web URL as the path



SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

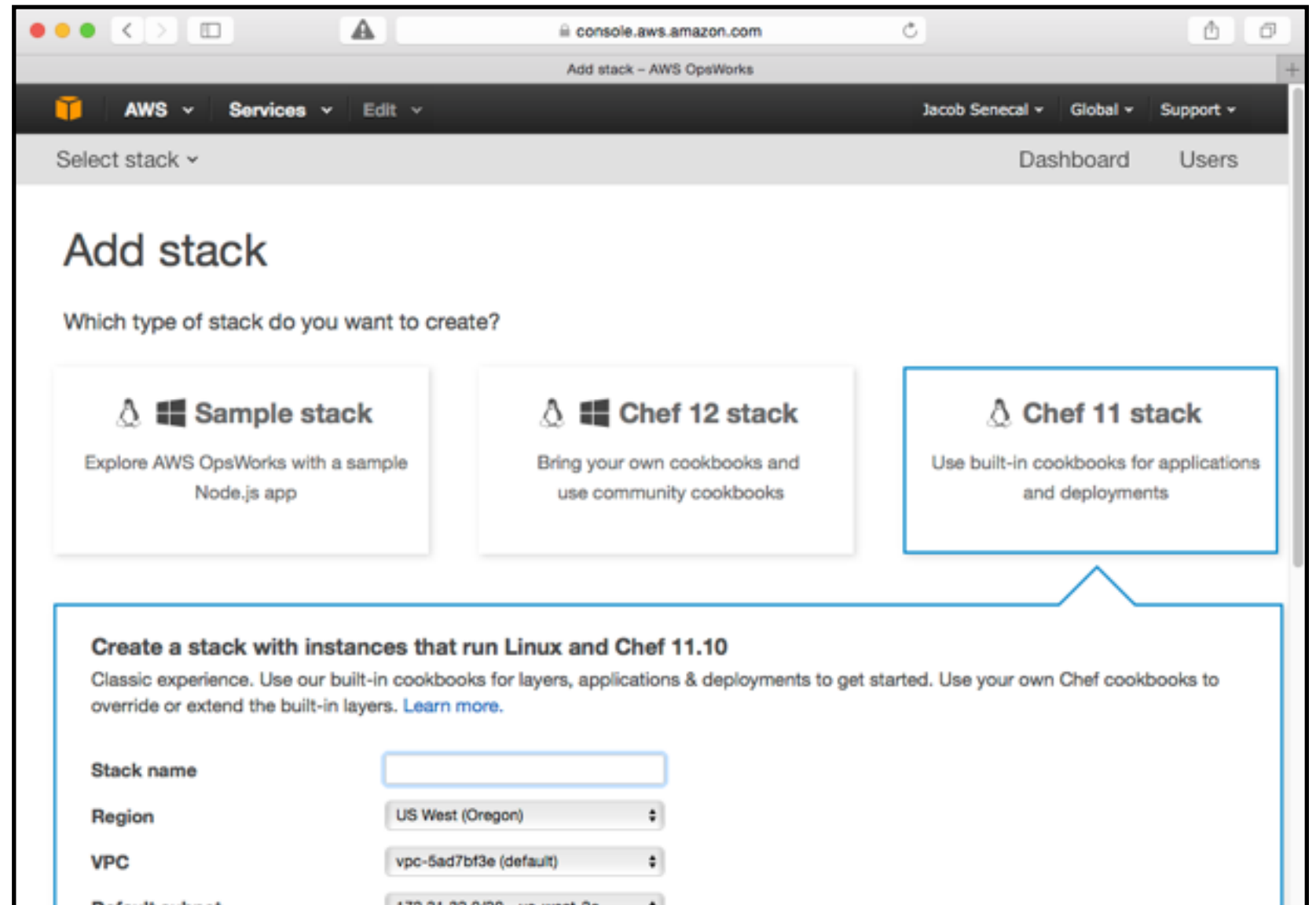
SET UP DEPLOY KEY

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



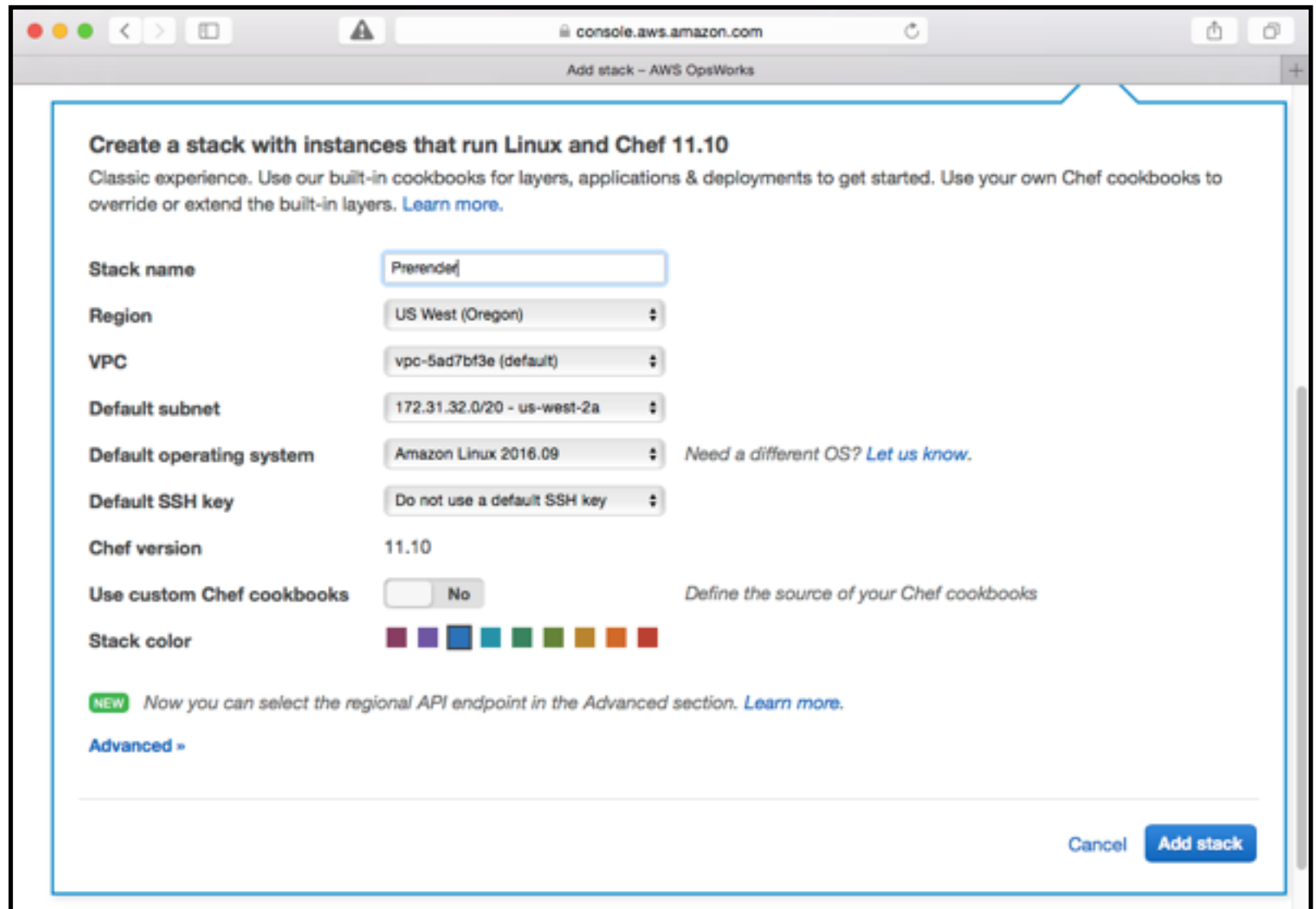
ADD OPSWORKS STACK

- ▶ 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)



The screenshot shows the 'Add stack - AWS OpsWorks' console window. The title is 'Create a stack with instances that run Linux and Chef 11.10'. Below the title is a description: 'Classic experience. Use our built-in cookbooks for layers, applications & deployments to get started. Use your own Chef cookbooks to override or extend the built-in layers. [Learn more.](#)'

The configuration fields are as follows:

- Stack name:** Pre-render
- Region:** US West (Oregon)
- VPC:** vpc-5ad7bf3e (default)
- Default subnet:** 172.31.32.0/20 - us-west-2a
- Default operating system:** Amazon Linux 2016.09. A link 'Need a different OS? [Let us know.](#)' is present.
- Default SSH key:** Do not use a default SSH key
- Chef version:** 11.10
- Use custom Chef cookbooks:** No. A link 'Define the source of your Chef cookbooks' is present.
- Stack color:** A row of eight colored squares (purple, blue, teal, green, yellow, orange, red, brown).

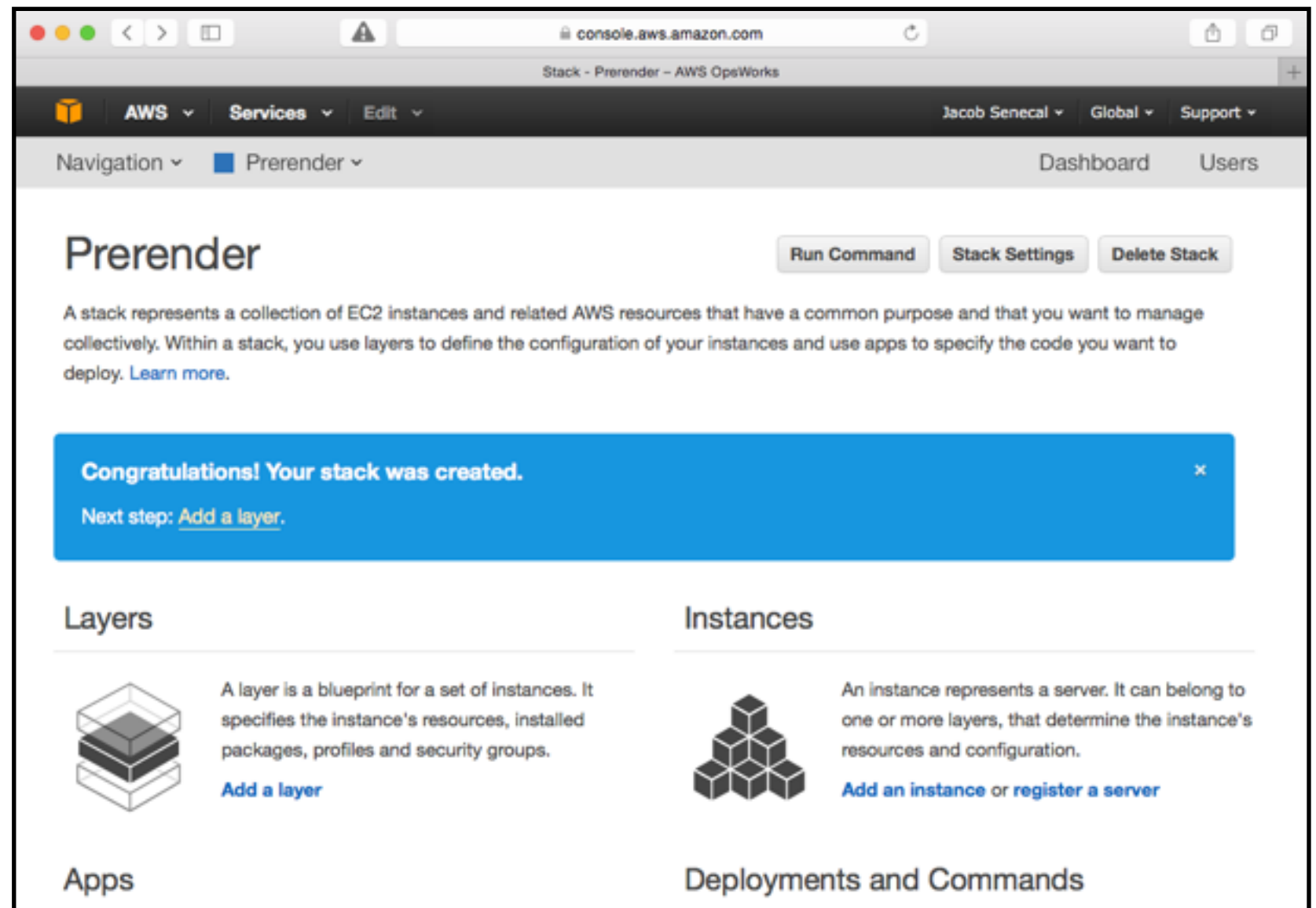
At the bottom, there is a 'NEW' badge and a message: 'Now you can select the regional API endpoint in the Advanced section. [Learn more.](#)' Below this is a link 'Advanced »'.

At the bottom right, there are two buttons: 'Cancel' and 'Add stack'.

SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

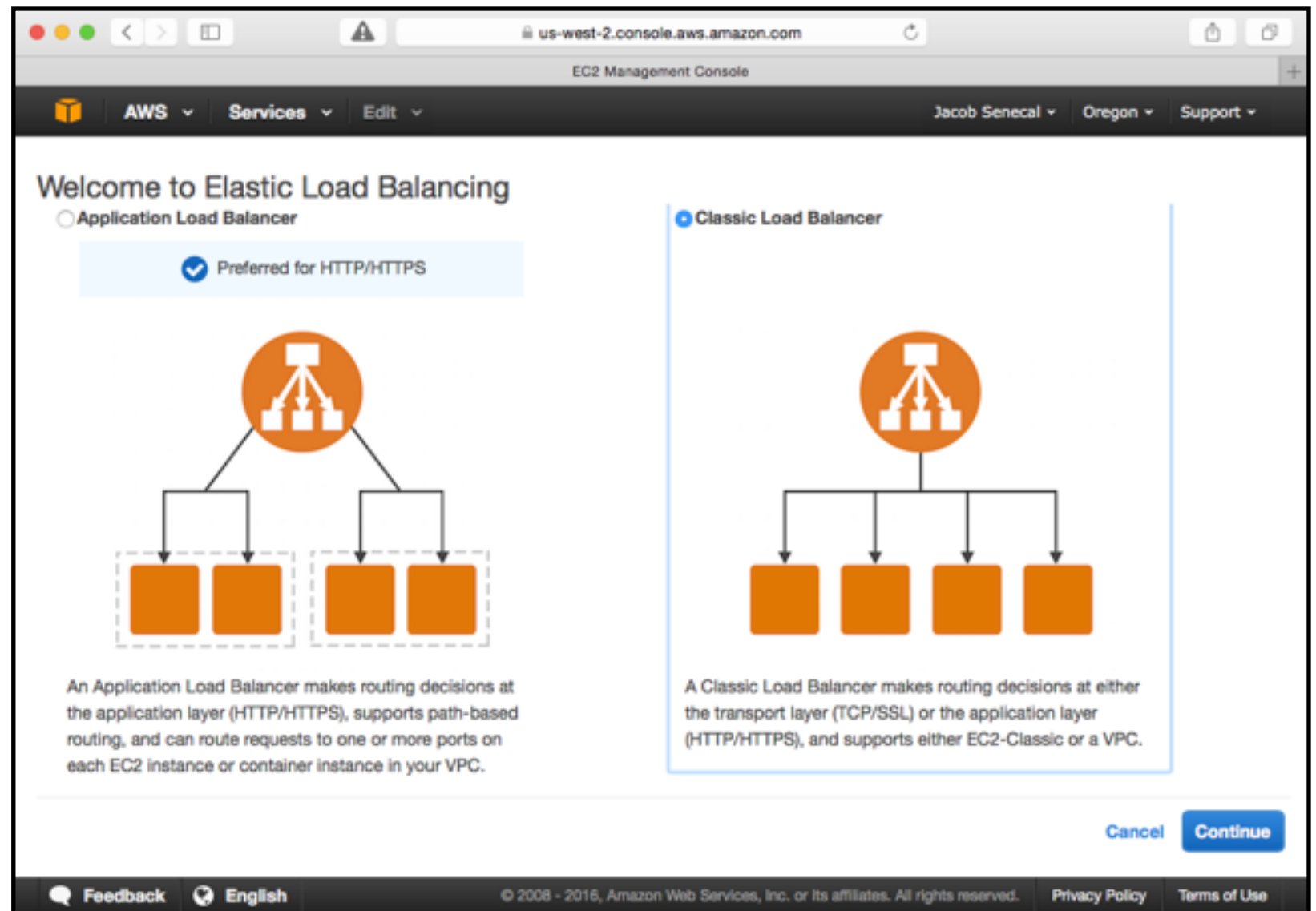
ALL DONE! (JK)

- ▶ 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)

us-west-2.console.aws.amazon.com

EC2 Management Console

AWS Services Edit

Jacob Senecal Oregon Support

1. Define Load Balancer 2. Assign Security Groups 3. Configure Security Settings 4. Configure Health Check 5. Add EC2 Instances 6. Add Tags

Step 1: Define Load Balancer

Basic Configuration

This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a unique name so that you can identify it from other load balancers you might create. You will also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed from any load balancer port to any port on your EC2 instances. By default, we've configured your load balancer with a standard web server on port 80.

Load Balancer name:

Create LB inside:

Create an internal load balancer: ☐ (what's this?)

Enable advanced VPC configuration: ☐

Listener Configuration:

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port
<input type="text" value="HTTP"/>	<input type="text" value="80"/>	<input type="text" value="HTTP"/>	<input type="text" value="80"/>

Add

Cancel Next: Assign Security Groups

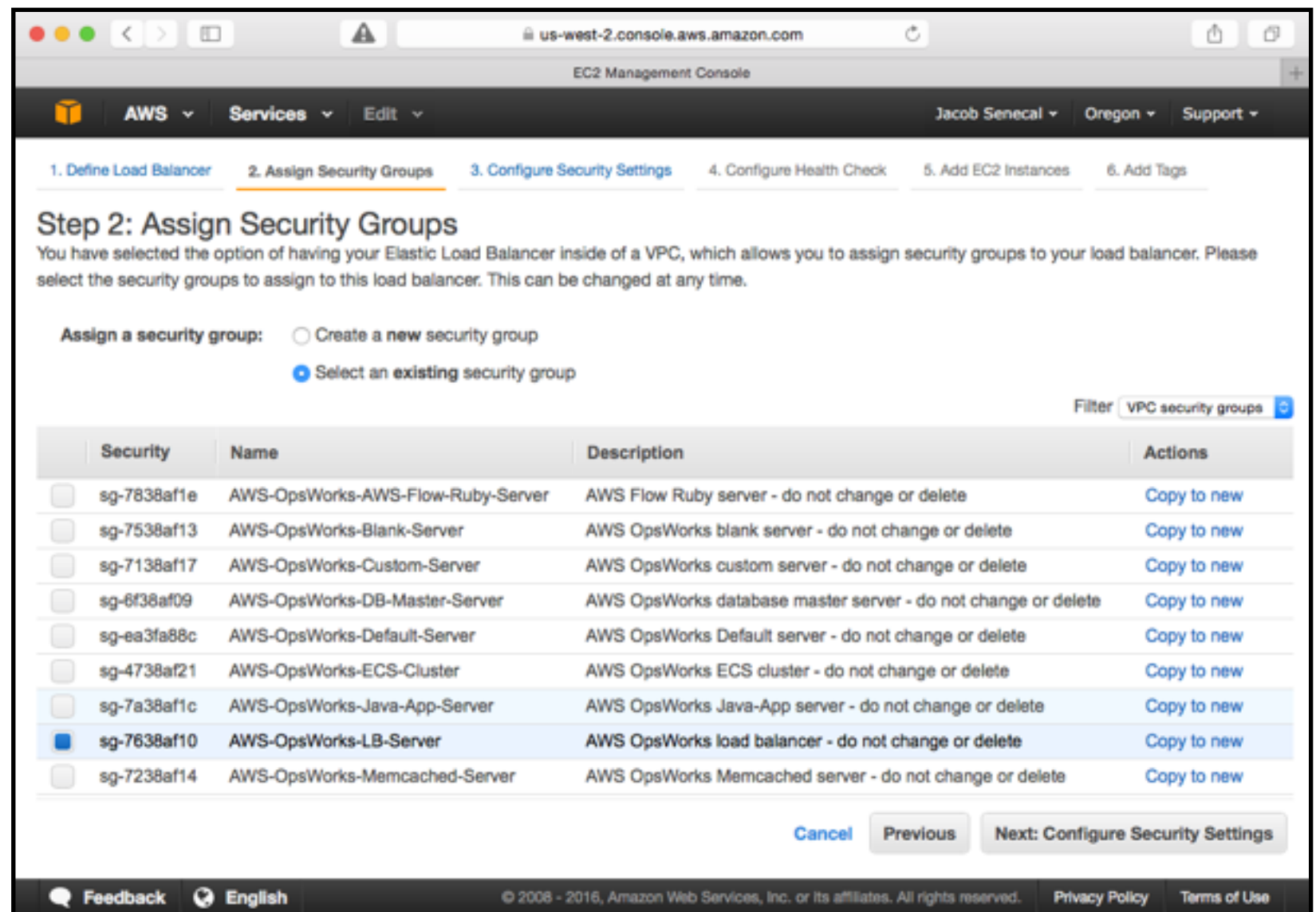
Feedback English

© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

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...

The screenshot shows the AWS Management Console interface for configuring a health check. The browser address bar indicates the URL is `us-west-2.console.aws.amazon.com`. The console header shows the user is logged in as Jacob Senecal in the Oregon region. The navigation bar includes tabs for '1. Define Load Balancer', '2. Assign Security Groups', '3. Configure Security Settings', '4. Configure Health Check' (which is the active step), '5. Add EC2 Instances', and '6. Add Tags'.

Step 4: Configure Health Check
Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

The configuration fields are as follows:

- Ping Protocol:** HTTP (selected from a dropdown)
- Ping Port:** 80
- Ping Path:** /

Advanced Details

- Response Timeout:** 30 seconds
- Interval:** 60 seconds
- Unhealthy threshold:** 2 (selected from a dropdown)
- Healthy threshold:** 2 (selected from a dropdown)

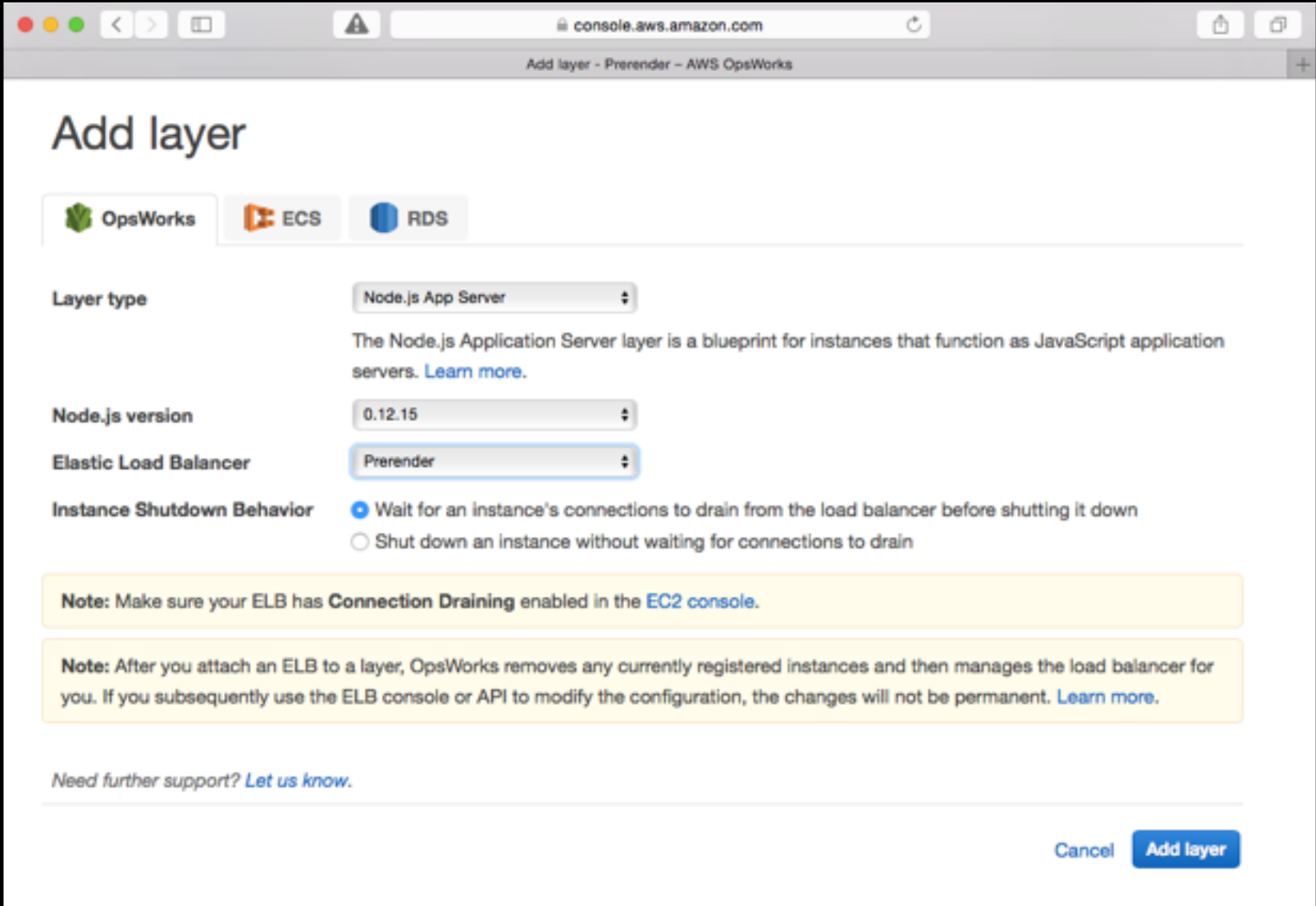
At the bottom right, there are three buttons: 'Cancel', 'Previous', and 'Next: Add EC2 Instances'.

The footer of the console includes a 'Feedback' link, the language 'English', and copyright information: '© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.' It also links to the 'Privacy Policy' and 'Terms of Use'.

SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

ADD NODE.JS LAYER

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



The screenshot shows the 'Add layer' page in the AWS OpsWorks console. The browser address bar shows 'console.aws.amazon.com'. The page title is 'Add layer - Prerender - AWS OpsWorks'. There are three tabs: 'OpsWorks' (selected), 'ECS', and 'RDS'. The 'Layer type' is set to 'Node.js App Server'. A description states: 'The Node.js Application Server layer is a blueprint for instances that function as JavaScript application servers. [Learn more.](#)'. The 'Node.js version' is set to '0.12.15'. The 'Elastic Load Balancer' is set to 'Prerender'. Under 'Instance Shutdown Behavior', the option 'Wait for an instance's connections to drain from the load balancer before shutting it down' is selected. Two yellow note boxes are present: the first says 'Note: Make sure your ELB has **Connection Draining** enabled in the [EC2 console](#).'; the second says 'Note: After you attach an ELB to a layer, OpsWorks removes any currently registered instances and then manages the load balancer for you. If you subsequently use the ELB console or API to modify the configuration, the changes will not be permanent. [Learn more.](#)'. At the bottom, there is a link 'Need further support? [Let us know.](#)' and two buttons: 'Cancel' and 'Add layer'.

console.aws.amazon.com

Add layer - Prerender - AWS OpsWorks

Add layer

OpsWorks ECS RDS

Layer type: Node.js App Server

The Node.js Application Server layer is a blueprint for instances that function as JavaScript application servers. [Learn more.](#)

Node.js version: 0.12.15

Elastic Load Balancer: Prerender

Instance Shutdown Behavior: ☒ Wait for an instance's connections to drain from the load balancer before shutting it down
☐ Shut down an instance without waiting for connections to drain

Note: Make sure your ELB has **Connection Draining** enabled in the [EC2 console](#).

Note: After you attach an ELB to a layer, OpsWorks removes any currently registered instances and then manages the load balancer for you. If you subsequently use the ELB console or API to modify the configuration, the changes will not be permanent. [Learn more.](#)

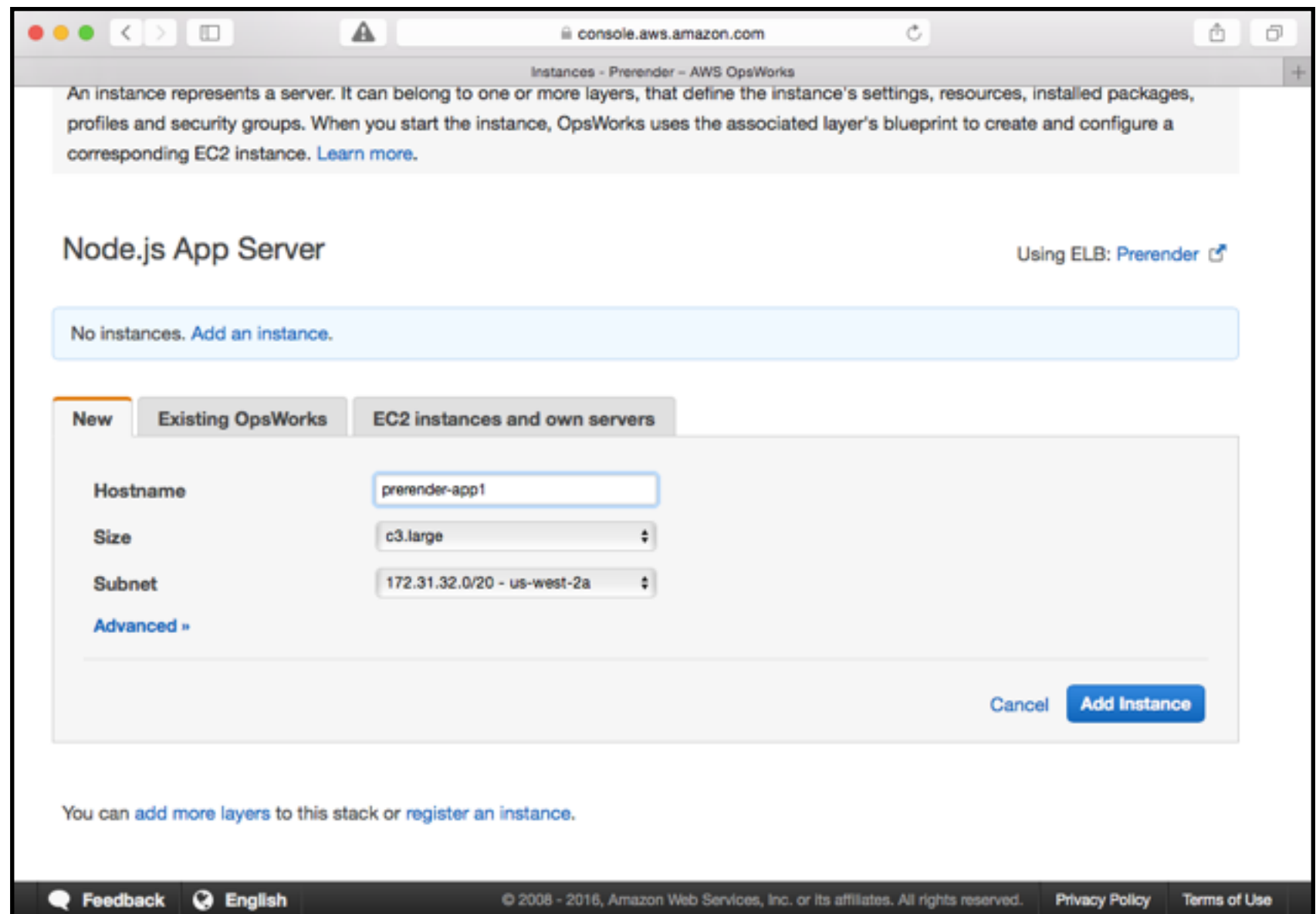
Need further support? [Let us know.](#)

Cancel Add layer

SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

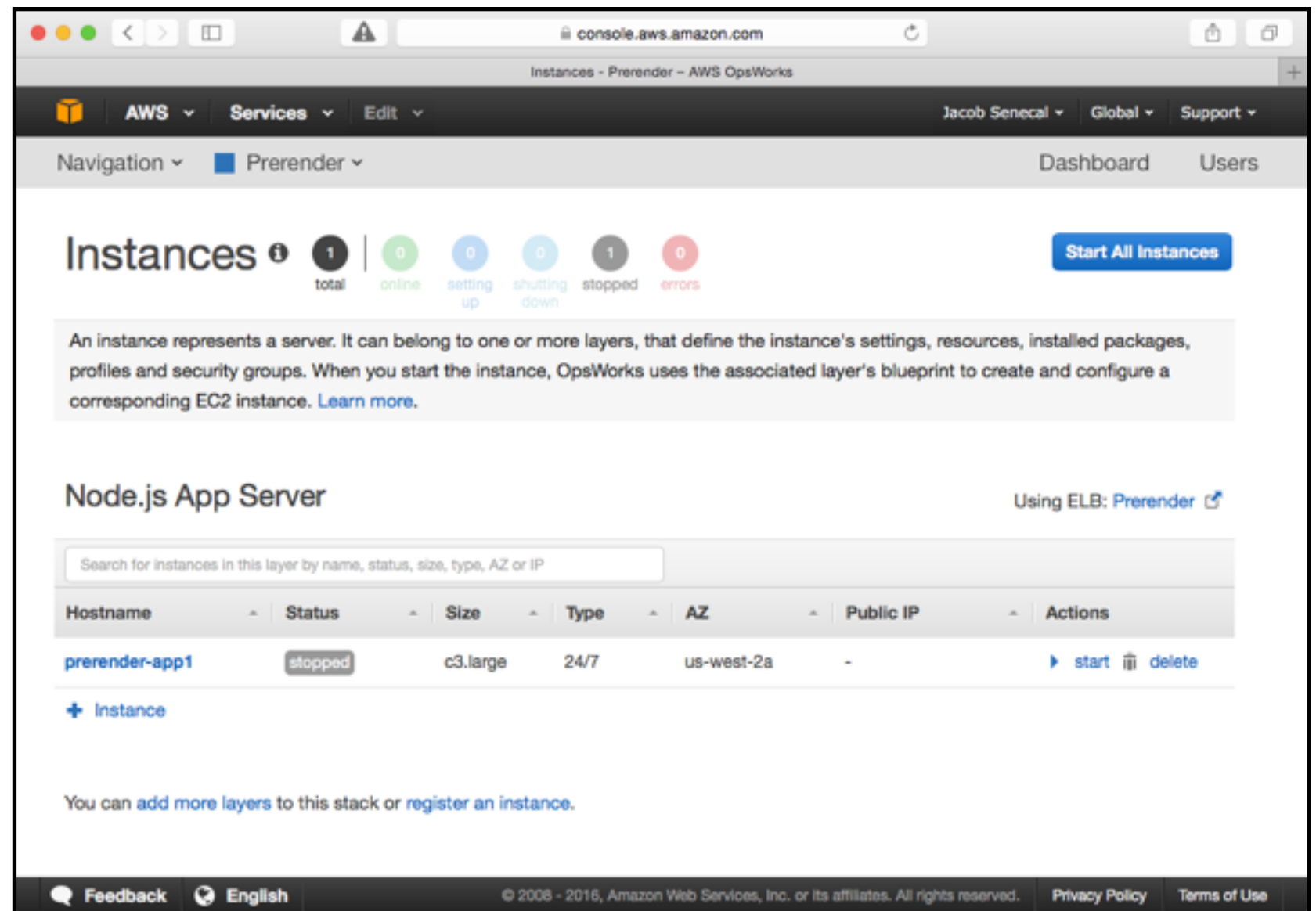
ADD NEW INSTANCE

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



START NEW INSTANCE

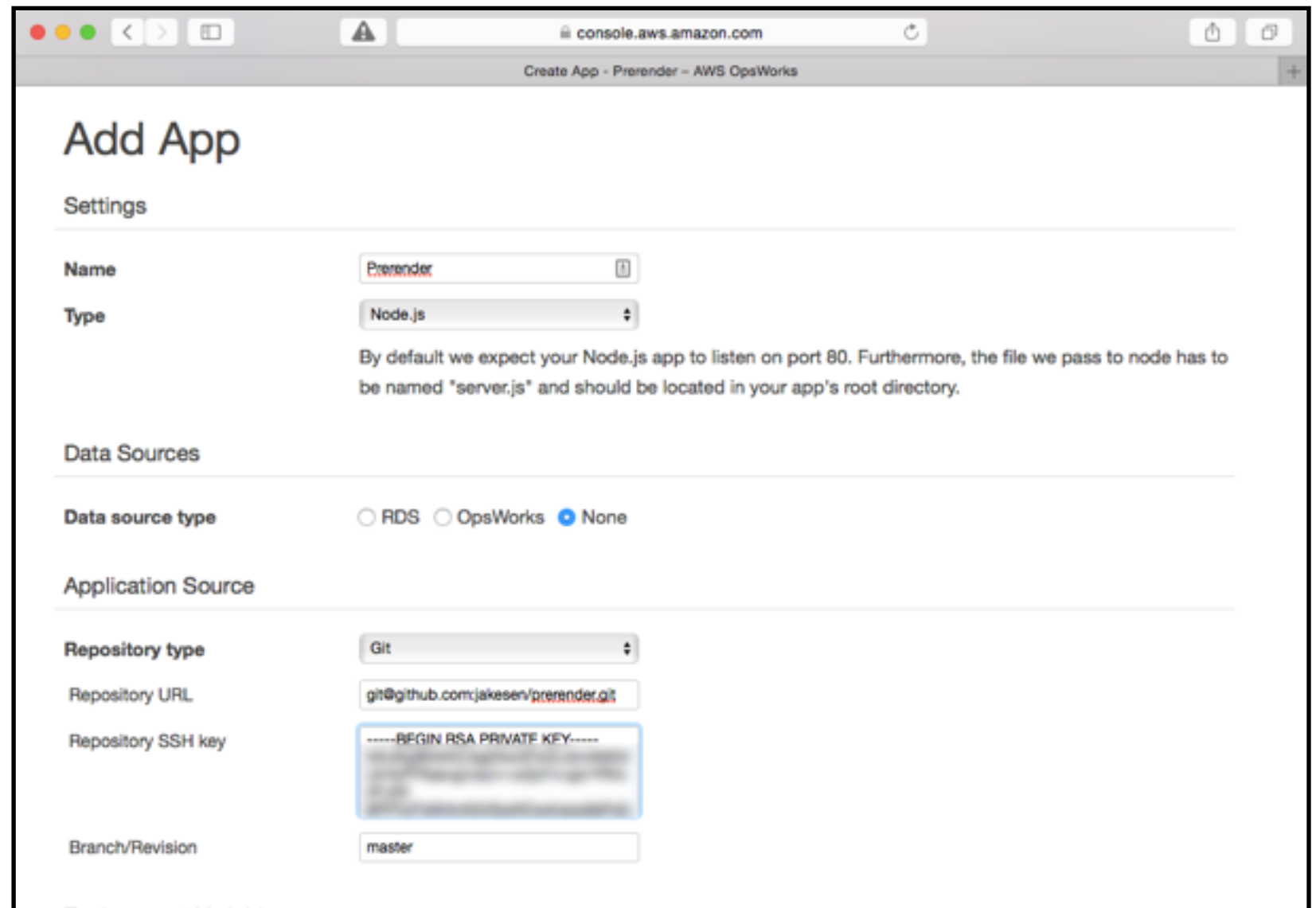
- ▶ lick start to provision and boot new EC2 instance
- ▶ You'll see the status change to Requested, Booting, Running Setup, and finally Online



SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

ADD PRERENDER APP

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



The screenshot shows the 'Add App' page in the AWS OpsWorks console. The browser address bar shows 'console.aws.amazon.com'. The page title is 'Create App - Prerender - AWS OpsWorks'. The 'Add App' section is divided into three main areas: Settings, Data Sources, and Application Source.

Settings

- Name:** A text input field containing 'Prerender'.
- Type:** A dropdown menu set to 'Node.js'. Below it, a note states: 'By default we expect your Node.js app to listen on port 80. Furthermore, the file we pass to node has to be named "server.js" and should be located in your app's root directory.'

Data Sources

- Data source type:** Three radio buttons are present: 'RDS', 'OpsWorks', and 'None'. The 'None' option is selected.

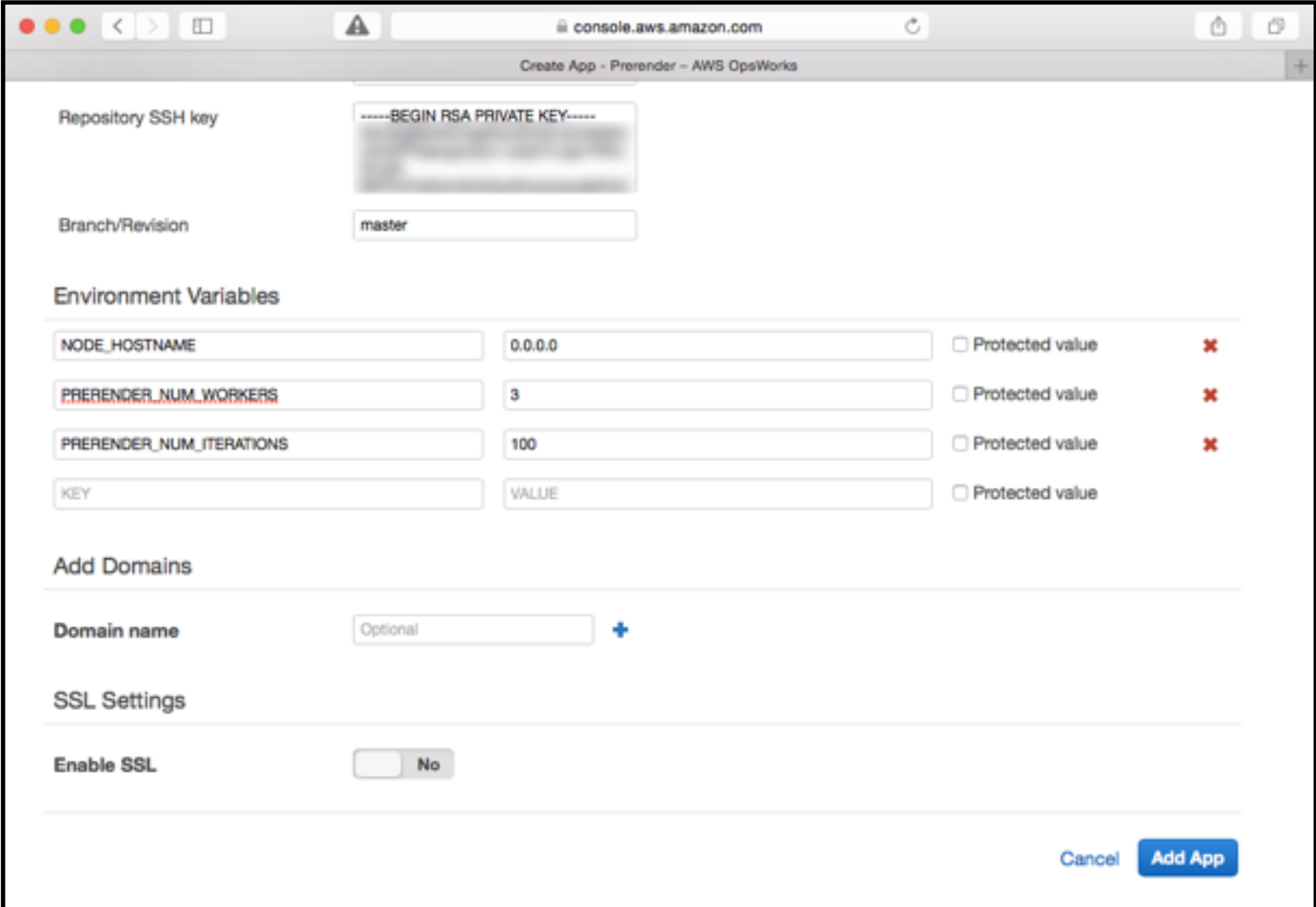
Application Source

- Repository type:** A dropdown menu set to 'Git'.
- Repository URL:** A text input field containing 'git@github.com:jakesen/prerender.git'.
- Repository SSH key:** A text input field containing '-----BEGIN RSA PRIVATE KEY-----' followed by a blurred private key.
- Branch/Revision:** A text input field containing 'master'.

SET UP YOUR OWN PRERENDER SERVER WITH AWS OPSWORKS

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



The screenshot shows the AWS OpsWorks console interface for creating a new application named 'Prerender'. The page is titled 'Create App - Prerender - AWS OpsWorks'. It contains several sections for configuration:

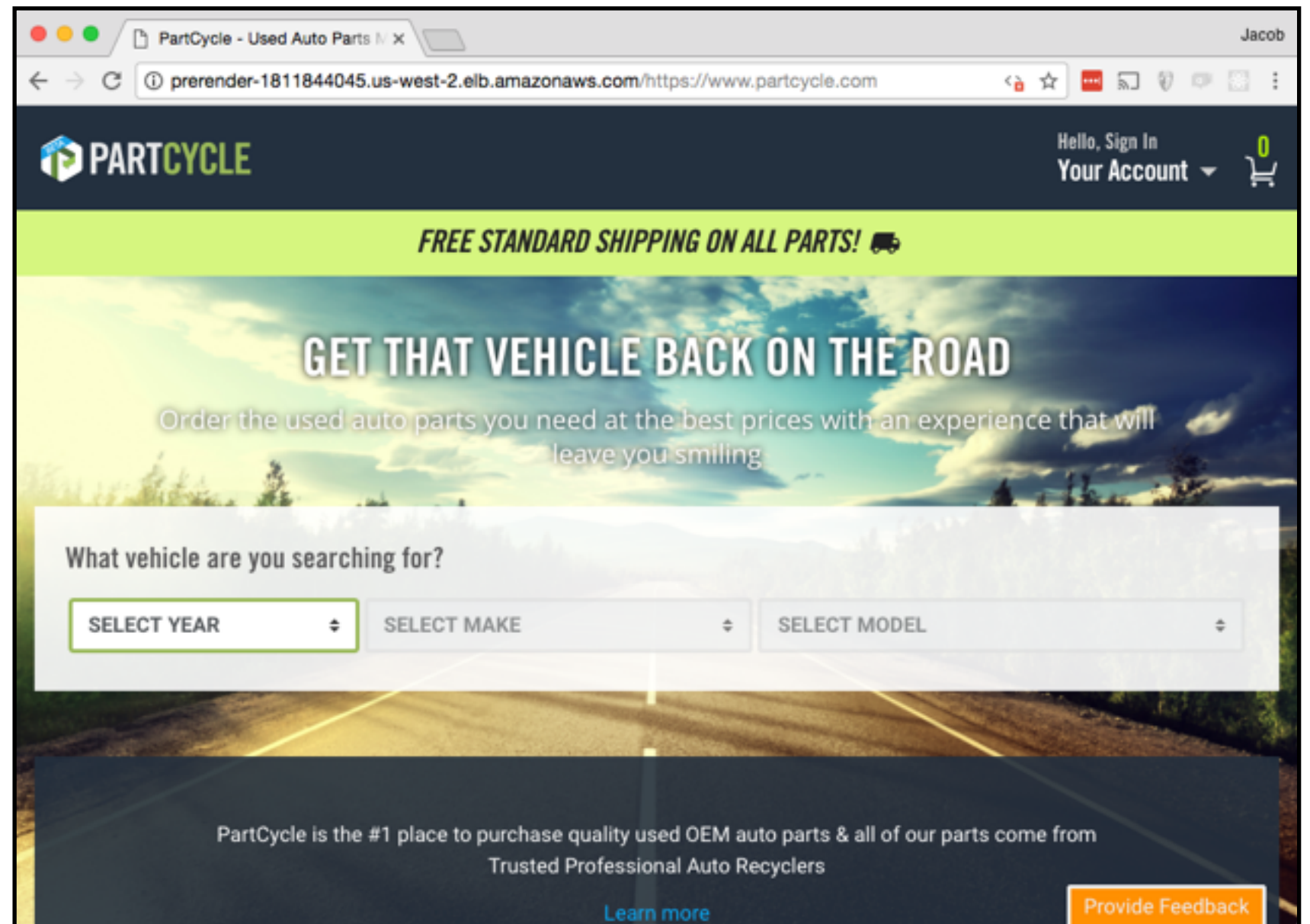
- Repository SSH key:** A text area containing a blurred RSA private key, with a label '-----BEGIN RSA PRIVATE KEY-----' at the top.
- Branch/Revision:** A text input field with the value 'master'.
- Environment Variables:** A table with four rows for defining environment variables. Each row has a 'KEY' column, a 'VALUE' column, and a 'Protected value' checkbox with a red 'X' icon to its right.

KEY	VALUE	Protected value
NODE_HOSTNAME	0.0.0.0	<input type="checkbox"/>
PRERENDER_NUM_WORKERS	3	<input type="checkbox"/>
PRERENDER_NUM_ITERATIONS	100	<input type="checkbox"/>
KEY	VALUE	<input type="checkbox"/>
- Add Domains:** A section with a 'Domain name' input field containing the word 'Optional' and a blue plus icon to its right.
- SSL Settings:** A section with an 'Enable SSL' toggle switch currently set to 'No'.

At the bottom right of the form, there are two buttons: 'Cancel' and 'Add App'.

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

