**Exercise 2**

*Configuring Auto Scaling and Load Balancing*

*Deploying a node.js server connected to an RDS database.*

*Monitoring services with CloudWatch*

**Prior Knowledge**

Unix Command Line Shell

EC2 starting servers

**Learning Objectives**

How servers interconnect in EC2

Passing configuration and automating setup of services in EC2

Scaling

Load Balancing

**Software Requirements**

* AWS CLI

**Part A: Starting an instance with a userdata configuration**

1. In our previous lab, we installed and started Apache by hand in the EC2 instance. Obviously that is not a tenable approach for a real production system. There are several options that could replace this:
   1. We could set up a server by hand and then save the configuration to a new AMI image and use that in future.
   2. We could utilize Docker and containers (we’ll talk more about this later)
   3. We could use configuration management tools like Puppet, Chef, Salt or Ansible. Or Amazon’s own OpsWorks (which uses Chef)
   4. But those go beyond the scope of this class, so we are going to use a simpler approach based on Amazon’s “userdata” which allows us to pass a startup script to the newly launched instance.
2. EC2 allows us to pass a script that is run as root. This is passed in a format called userdata.
3. Go back to the console (and login if you need to again)  
   [**https://ox-clo.signin.aws.amazon.com/console**](https://ox-clo.signin.aws.amazon.com/console)
4. There is already an Amazon Aurora (MySQL compatible) database running in the cloud. It has a small amount of data in it that we will query from a node.js application. If you go to the RDS section of the AWS management console you can take a look at this instance. Please do not modify it!
5. Now let’s try the instance manually before we create an auto-scaling version.
6. Go to the EC2 console, and Launch a new instance.
7. Choose the Ubuntu Server 14.04 LTS (HVM)
8. Once again choose a t2.micro instance and then **Next: Configure Instance Details**
9. At the bottom of the page you will find a section called **Advanced Details**. Expand this.
10. This is where our script will go. We are going to paste it into the **User Data** section. (You could also create a file and upload that if you prefer)
11. In your browser go to <http://freo.me/oxclo-userdata>
12. Now copy and paste the startup script into the user data section. It looks like this:

#!/bin/bash

# verbosity

set -e -x

# update the package list

apt-get update

# install node, node package manager and git.

apt-get -y install nodejs npm git

# some node packages including forever expect nodejs to be called node

ln -s /usr/bin/nodejs /usr/local/bin/node

# use the node package manager to install express.js and mysql support

npm install express mysql

# forever is a daemon for running node.js code

npm install forever -g

# change to the ubuntu home directory

cd /home/ubuntu

# use git to copy the node.js code into the system

git clone https://github.com/pzfreo/auto-deploy-node-js.git

cd auto-deploy-node-js

# pass the DB connection parameters into the code

export DBURL=oxclo-db-cluster.cluster-citfamc1edxs.eu-west-1.rds.amazonaws.com

export DBUSER=node

export DBPW=node

# start the server as a daemon

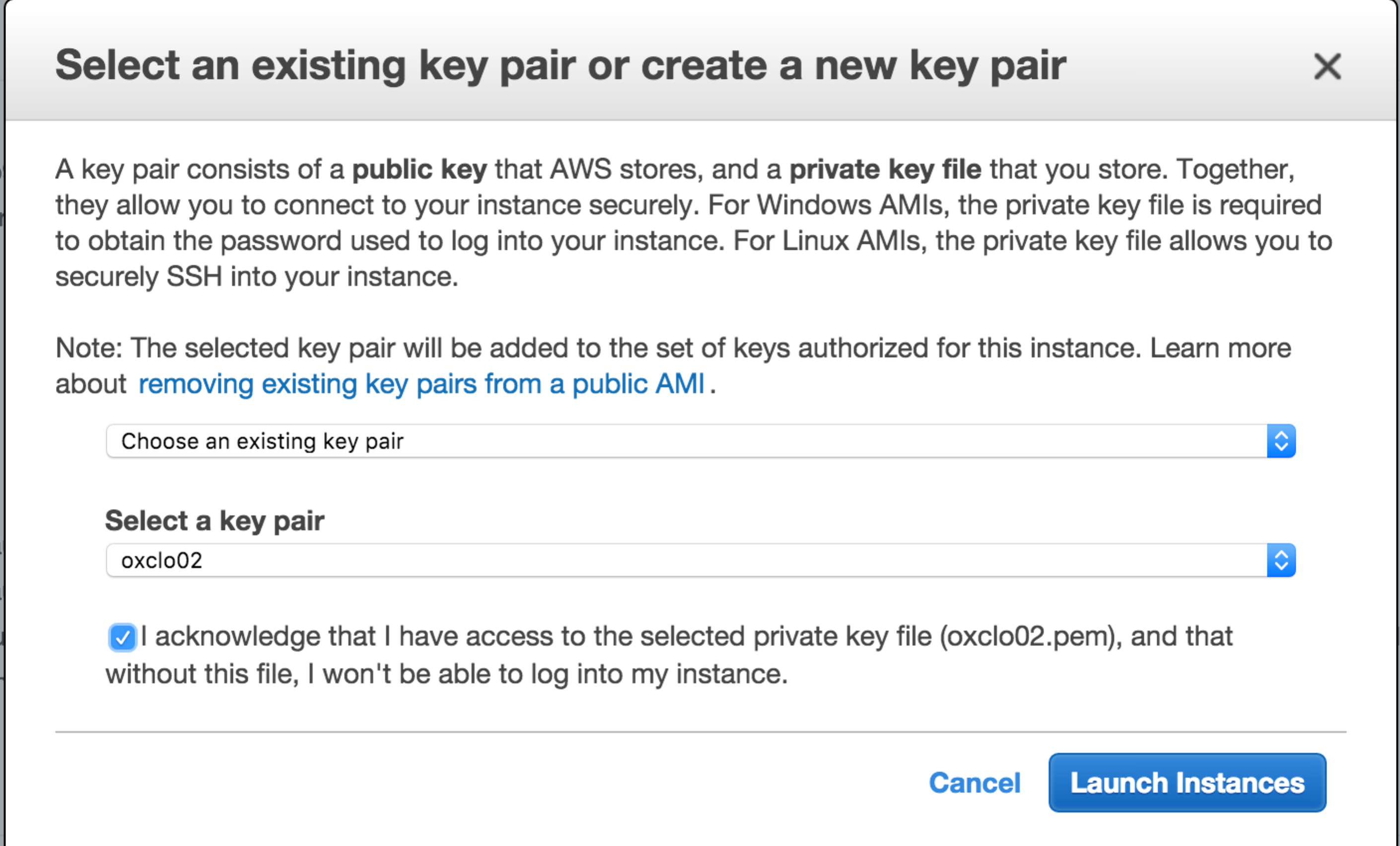
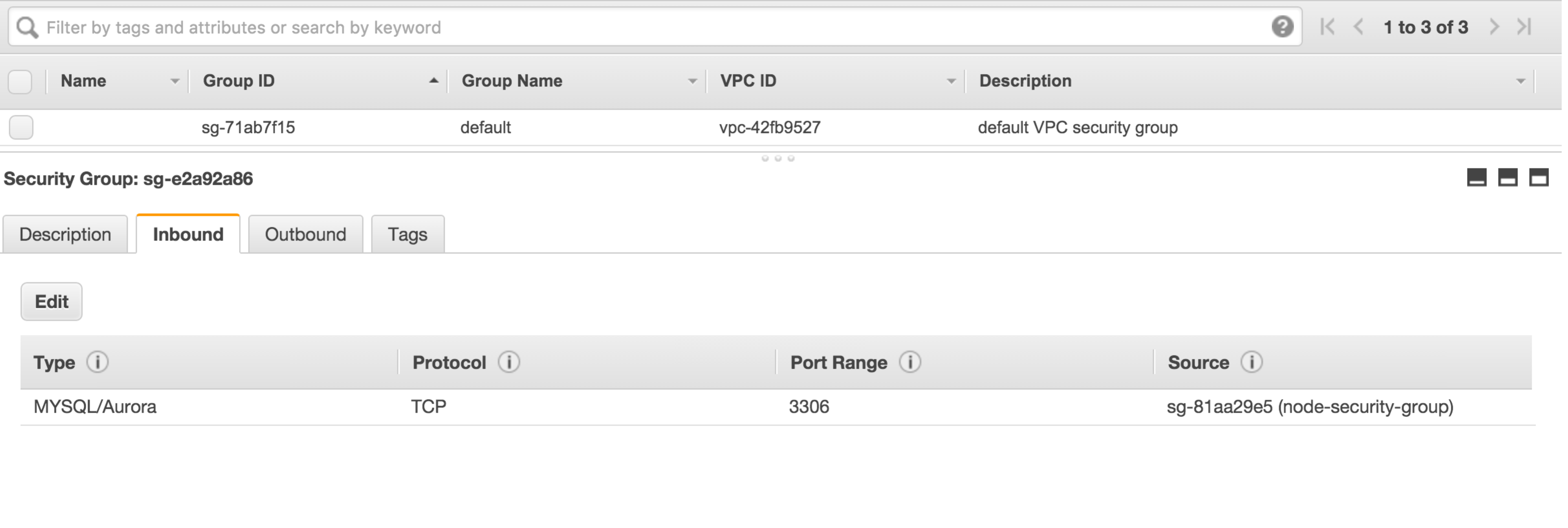
forever start --minUptime=1000 --spinSleepTime=1000 clustertest.js

#that's all

The script is doing the following:

* + 1. Installing node.js, the node package manager (npm) and git
    2. Using npm to install some node packages (mysql, express.js and forever)
    3. Using git to install our source code:   
       <https://github.com/pzfreo/auto-deploy-node-js/blob/master/clustertest.js>
    4. Setting up the URL, userid and password for the database.
    5. Using the **forever** toolkit to run our node.js code

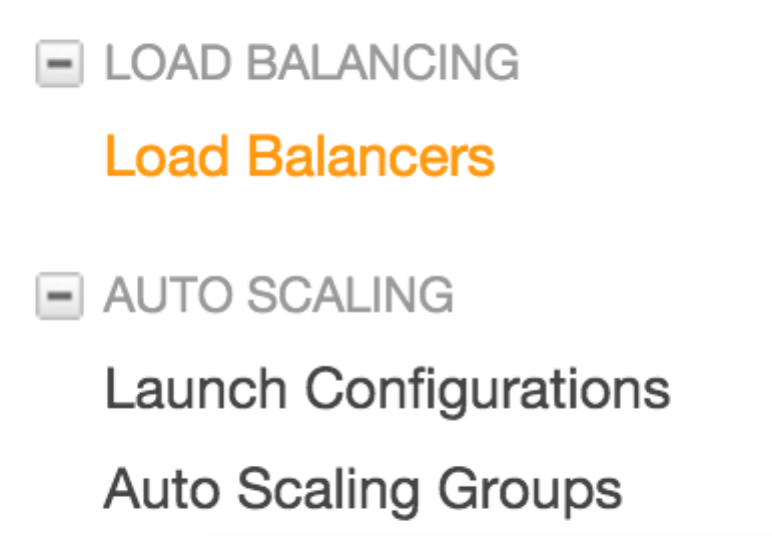
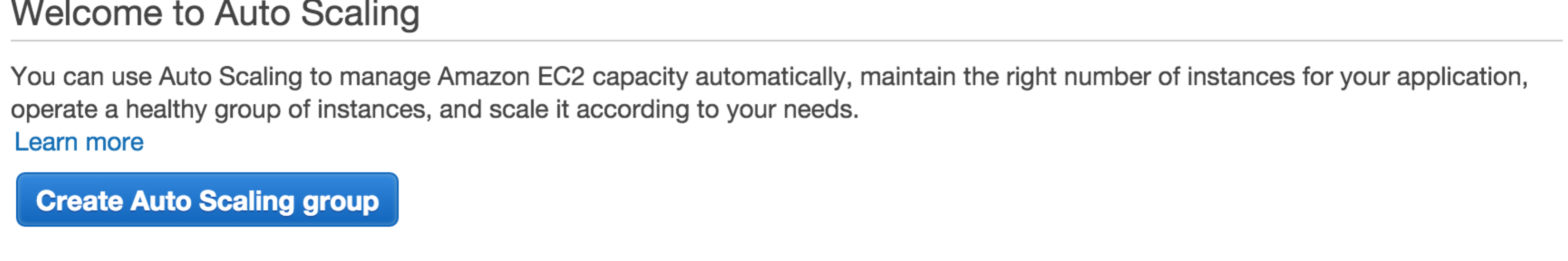
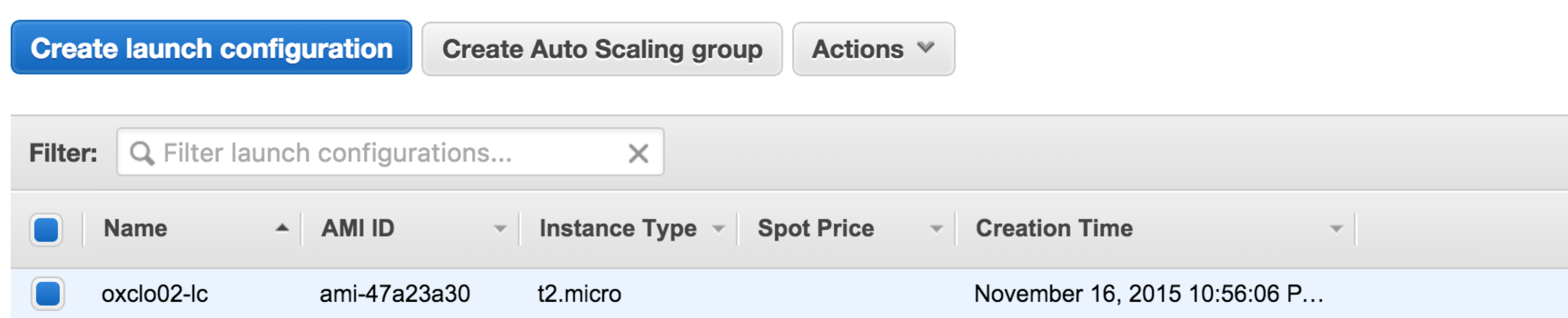
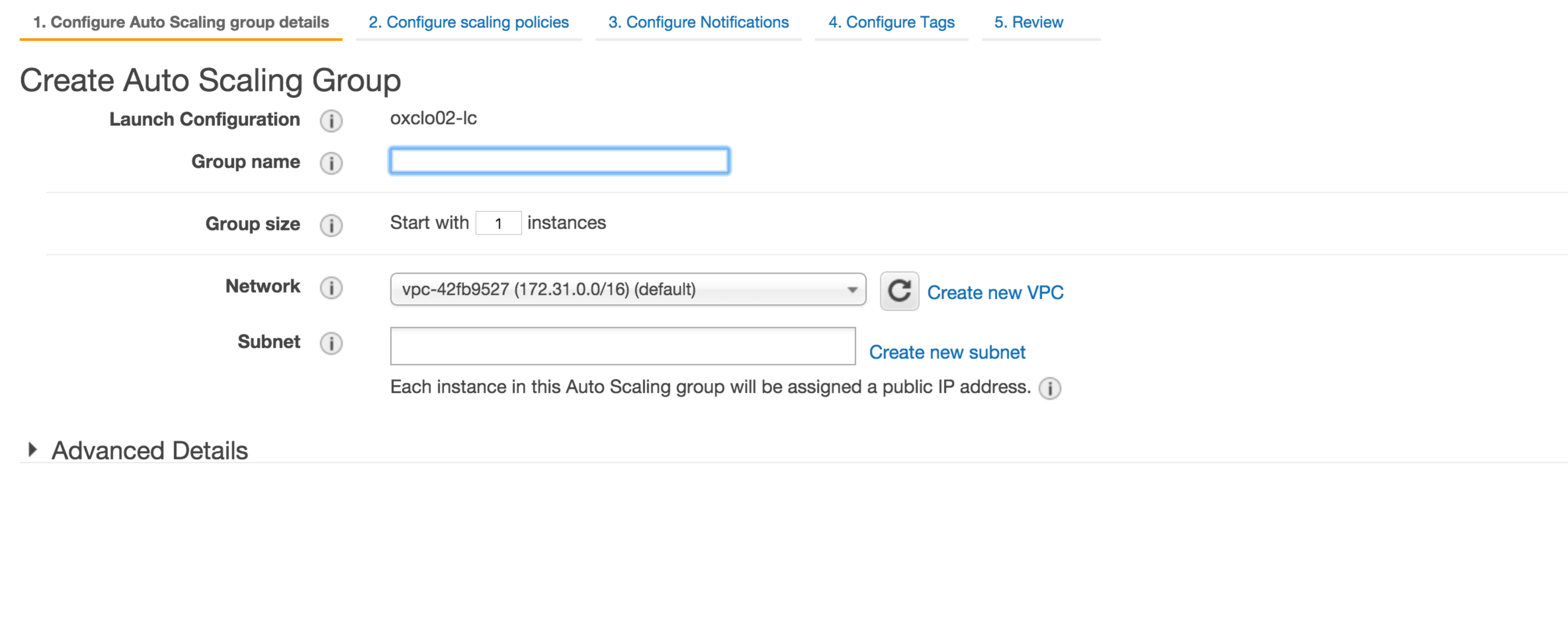
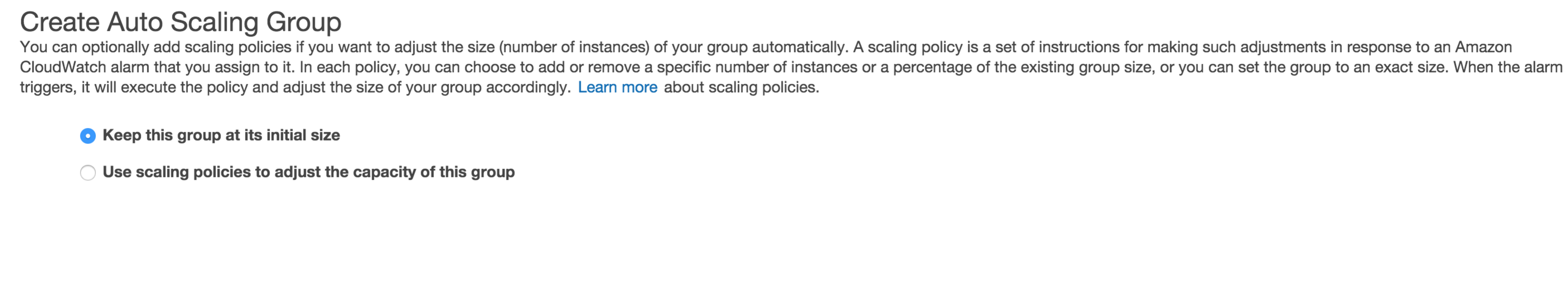
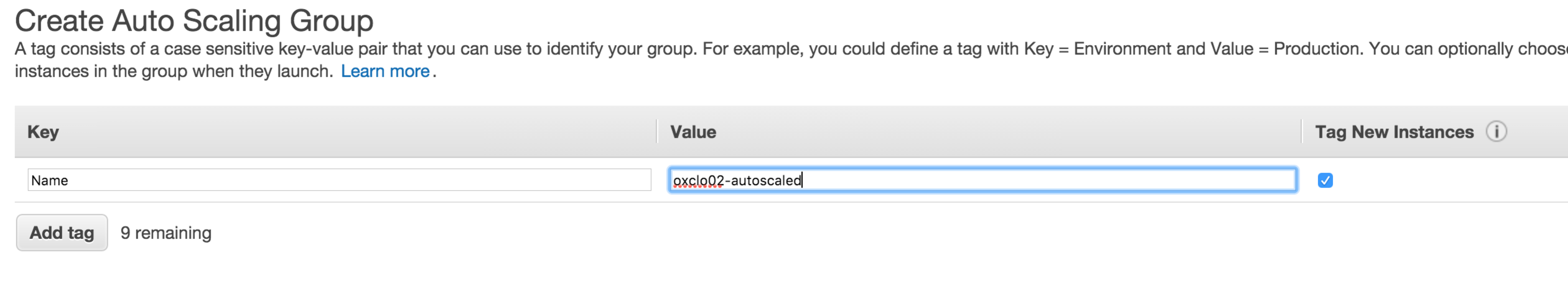
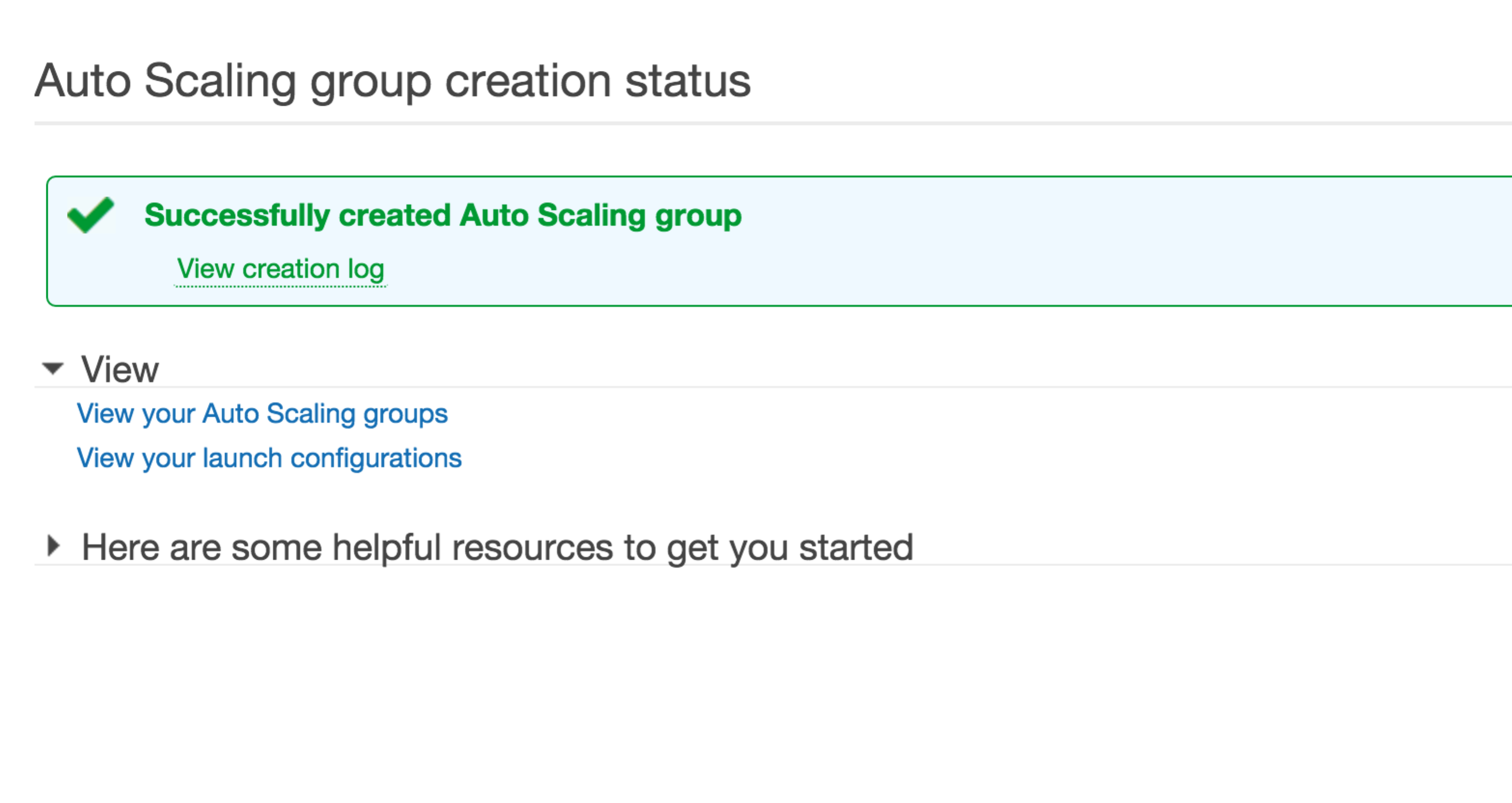
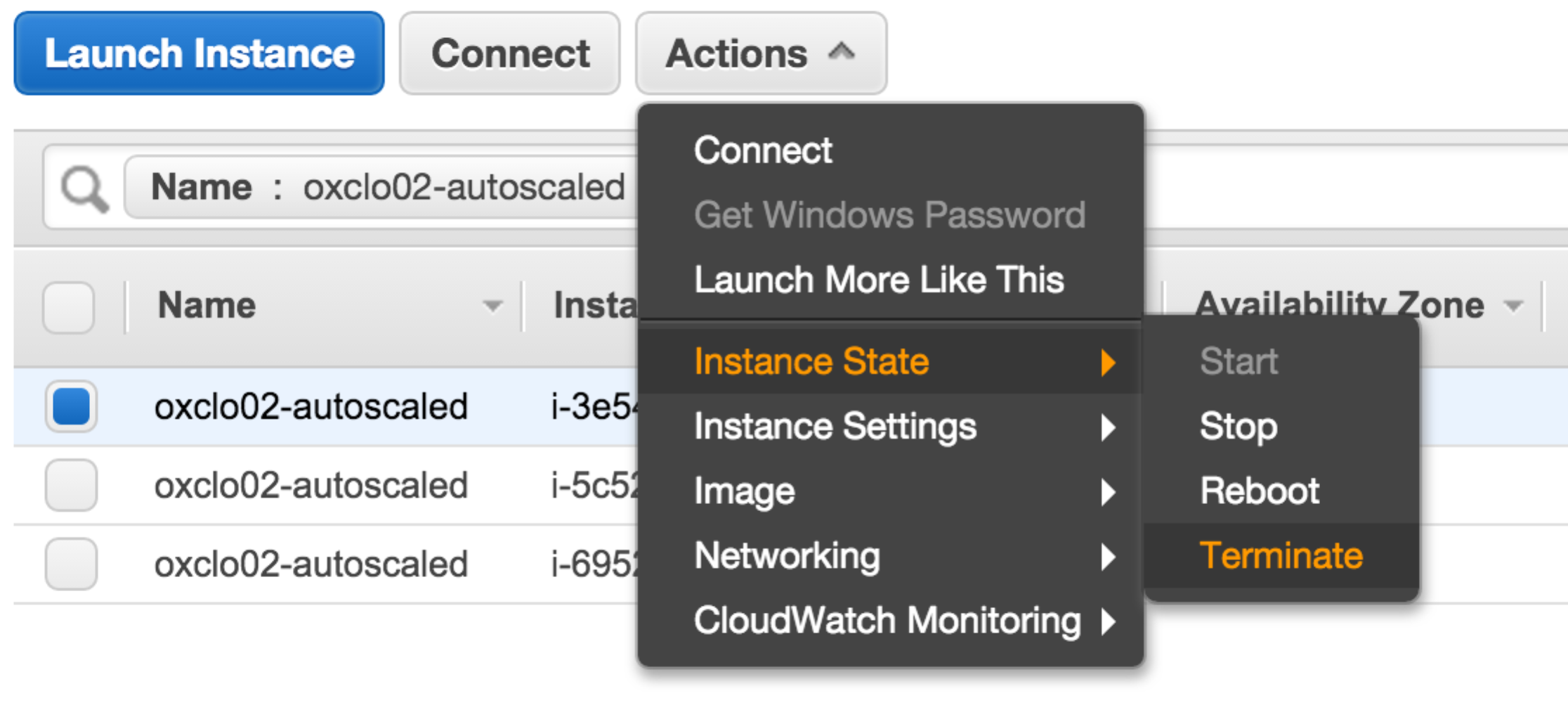
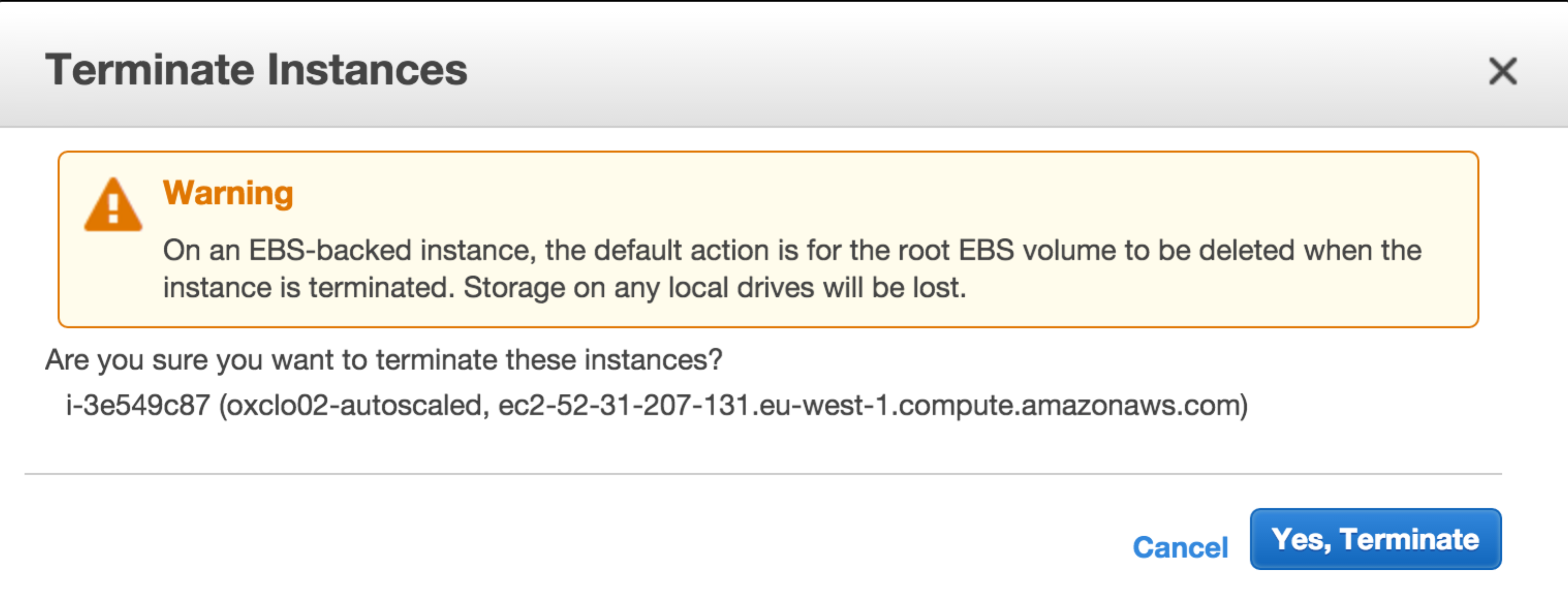
Hint: If you are worried about the security of this password (which you should be) then consider this. Firstly, you would certainly not normally put this into a public git repository! Secondly, only instances in Amazon can connect to the database (I’ll explain shortly). Thirdly, this userid/password only has the access rights to read a single table.

1. Click **Next: Add Storage,** then **Next: Tag Instance**
2. Add the name tag made up of *<your userid>*-node. E.g. oxclo02-node
3. Click **Next: Configure Security Group**
4. Select an existing security group in the dropdown menu
5. Choose the “node-security-group”   
   1. This is important because this group is allowed to access the database. We’ll take a look shortly.
6. Click **Review and Launch**
7. Click **Launch**
8. This time select to use an existing keypair, and find your own key pair. Check the box that says you have access to the PEM file:  
   
9. Now select **Launch Instances**
10. As before, go take a look at your instance status by clicking on the instance link.
11. While you wait for your instance to get going, you can take a look at the Security Groups. If you look at the **rds-security-group** and take a look at the inbound rules, you will see the following:  
      
    What this shows is that only instances started in the node-security-group can access the RDS instances port 3306.
12. Go back and find your instance running (e.g. tagged oxclo02-node, but with your userid).
13. If it has started and the status checks are finished, it may have completed its startup script. But this is a lot of work for a poor old micro instance to manage, so don’t expect miracles.
14. Copy the public IP address of the instance and try browsing to   
    <http://ww.xx.yy.zz:3001> (where the ww.xx.yy.zz are replaced with the public IP of your instance).
15. If everything is running then you should see some json returned. If there is a problem, you can see the state of your startup by SSH-ing into your instance and doing:
    1. tail -f /var/log/cloud-init-output.log

If this is still scrolling past then your server hasn’t started up yet.

* 1. If this shows something like:  
     cloud-init v. 0.7.5 finished at Mon, 16 Jun 2015 22:30:26 +0000. Datasource DataSourceEc2. Up 72.17 seconds  
     then the server init has started. Press Ctrl-C to exit.
  2. Once the server has started fully, try browsing again.

**PART B. Creating a Launch Configuration**

1. In order to auto-scale this we are going to create a template for launching new servers. It is just like creating a server but then we let Amazon decide when to start new servers.
2. In the EC2 console, you need to scroll the left hand menu to the bottom where you will find Launch Configurations:  
   
3. Click on **Launch Configurations**
4. If you are the first to get here then this will start a sort of wizard:  
     
   
   1. Click on **Create Auto Scaling group**
5. However, it is more likely that there is already at least one Launch Configuration in the system, in which case it will look like:  
   ****
6. Now click on **Create Launch Configuration**
7. Repeat the process from step 7 through step 20. There are some minor differences. When you are configuring the instance details, you need to give the launch configuration a name.   
   Use *userid-lc* (e.g. oxclo02-lc).
8. Once you have completed the process you will be prompted to create an Auto Scaling group with this configuration. Click on it.  
   
9. You will see.  
   
10. Give the Group name as: *userid*-group (e.g. oxclo02-group).
11. Choose a subnet from the options that drop down when you select that box. Any one will do.
12. Click **Next: Configure Scaling Policies**
13. Leave this as **Keep this group at its original size  
    **
14. Click **Next: Configure Notifications,** then **Next: Configure Tags**
15. In the tags Key, specify **Name,** and in the Value field *userid-autoscaled* (e.g. oxclo02-autoscaled)
16. Click **Review**
17. Click **Create Auto Scaling Group:  
    **
18. Now if you go to your EC2 Instances dashboard, you should see a new instance starting up. Once it is started it will be tagged with your userid-autoscaled so you can see which ones are yours.
19. First check that your server is working properly by browsing <http://ip-address:3001>   
    You may need to be patient.
20. Now using the **EC2 Dashboard -> Running Instances** screen you can terminate this instance:
21. Click **Yes Terminate** on the next screen:  
    ****
22. Now wait up to 5 minutes and you should see a new instance spawned to replace the one you killed. Amazon is ensuring that you have an instance running at all times (give or take a little bit of startup time!).

Part B: Creating a Launch Configuration and setting up Auto Scaling

**Part C: Load Balancing the workload**

**Part D: Stress testing**