Exercise 2

Configuring Auto Scaling and Load Balancing
Deploying a node.js server connected to an RDS database.

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
AutoScaling (without 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:
    a. We could set up a server by hand and then save the configuration to a new AMI image and use that in future.
    b. We could utilize Docker and containers (we'll talk more about this later)
    c. We could use configuration management tools like Puppet, Chef, Salt or Ansible. Or Amazon's own OpsWorks (which uses Chef)
    d. 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

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 18.04 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-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:

i. Installing node.js, the node package manager (npm) and git
ii. Using npm to install some node packages (mysql, express.js and forever)
iii. Using git to install our source code: https://github.com/pzfreo/auto-deploy-node-js/blob/master/clustertest.js

iv. Setting up the URL, userid and password for the database.

> 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.

v. Using the **forever** toolkit to run our node.js code

13. Click **Next: Add Storage,** then **Next: Add Tags**

14. Add the **Name** tag as before. This time use <*your userid*>-node.
    E.g. oxclo02-node

15. Click **Next: Configure Security Group**

16. Select an existing security group in the dropdown menu

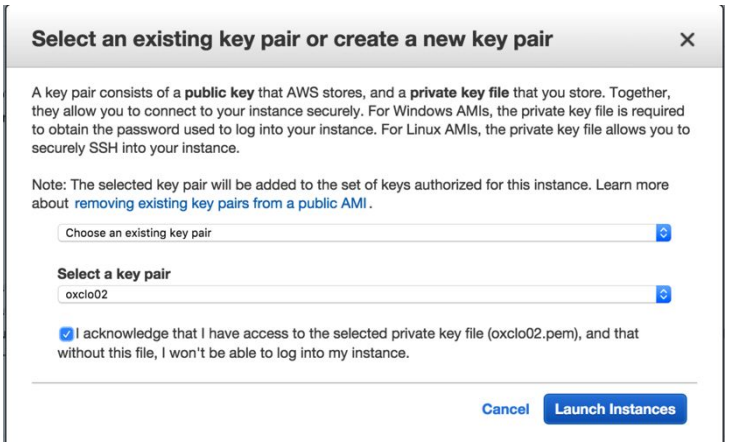
17. Choose the "node-security-group"

    a. *This is important because this group is allowed to access the database. We'll take a look shortly.*

18. Click **Review and Launch**

19. Click **Launch**

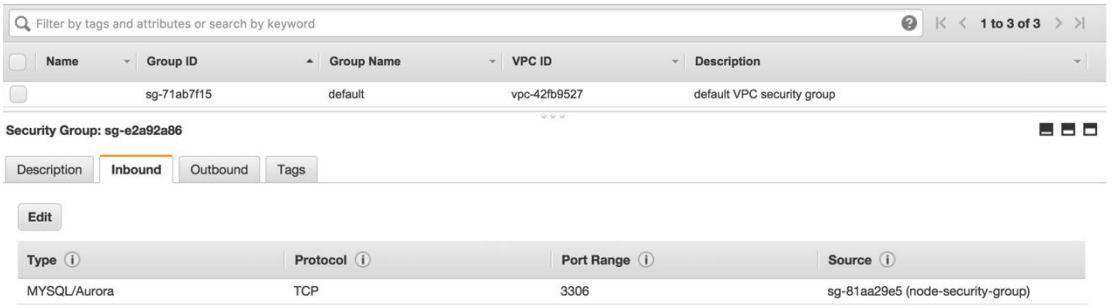
20. This time select to use an existing keypair, and find your own key pair. Check the box that says you have access to the PEM file:



21. Now select **Launch Instances**

22. As before, go take a look at your instance status by clicking on the instance link.

23. 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:



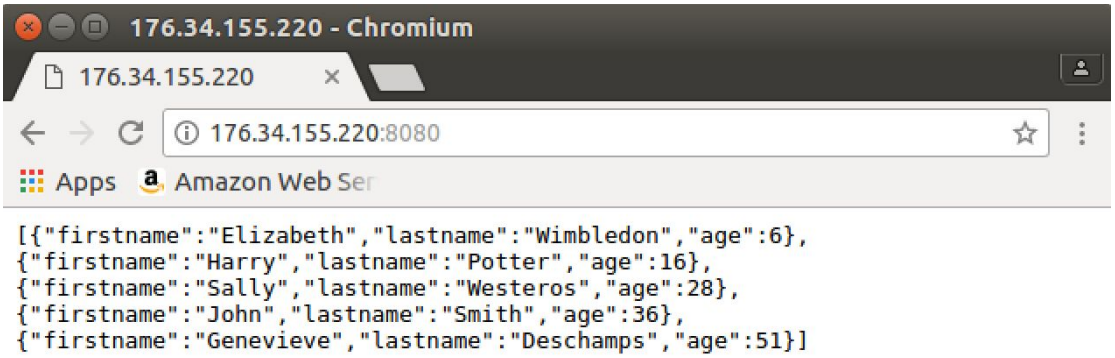
What this shows is that only instances started in the node-security-group can access the RDS instances port 3306.

24. Go back and find your instance running (e.g. tagged oxclo0*n*-node).

25. 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.

26. Copy the public IP address of the instance and try browsing to http://ww.xx.yy.zz:8080 (where the ww.xx.yy.zz are replaced with the public IP of your instance).

27. If everything is running then you should see some json returned.



28. If there is a problem, you can see the state of your startup by SSH-ing into your instance and doing:

```
tail -f /var/log/cloud-init-output.log
```

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

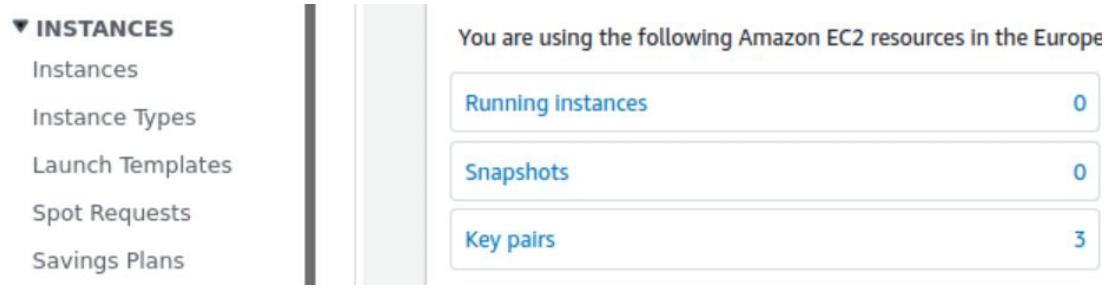
29. Once the server has started fully, try browsing again.

30. Once you have tested this, please **terminate** your instance through the AWS console.

PART B. Creating a Launch Template

31. In order to auto-scale this we are going to create a template for launching new servers. It is similar to creating a server but then we let Amazon decide when to start new servers.

32. In the **EC2 console**, you need to scroll the left hand menu to the bottom where you will find Launch Templates:



33. Click on **Launch Templates**

34. Now click on **Create Launch Template**

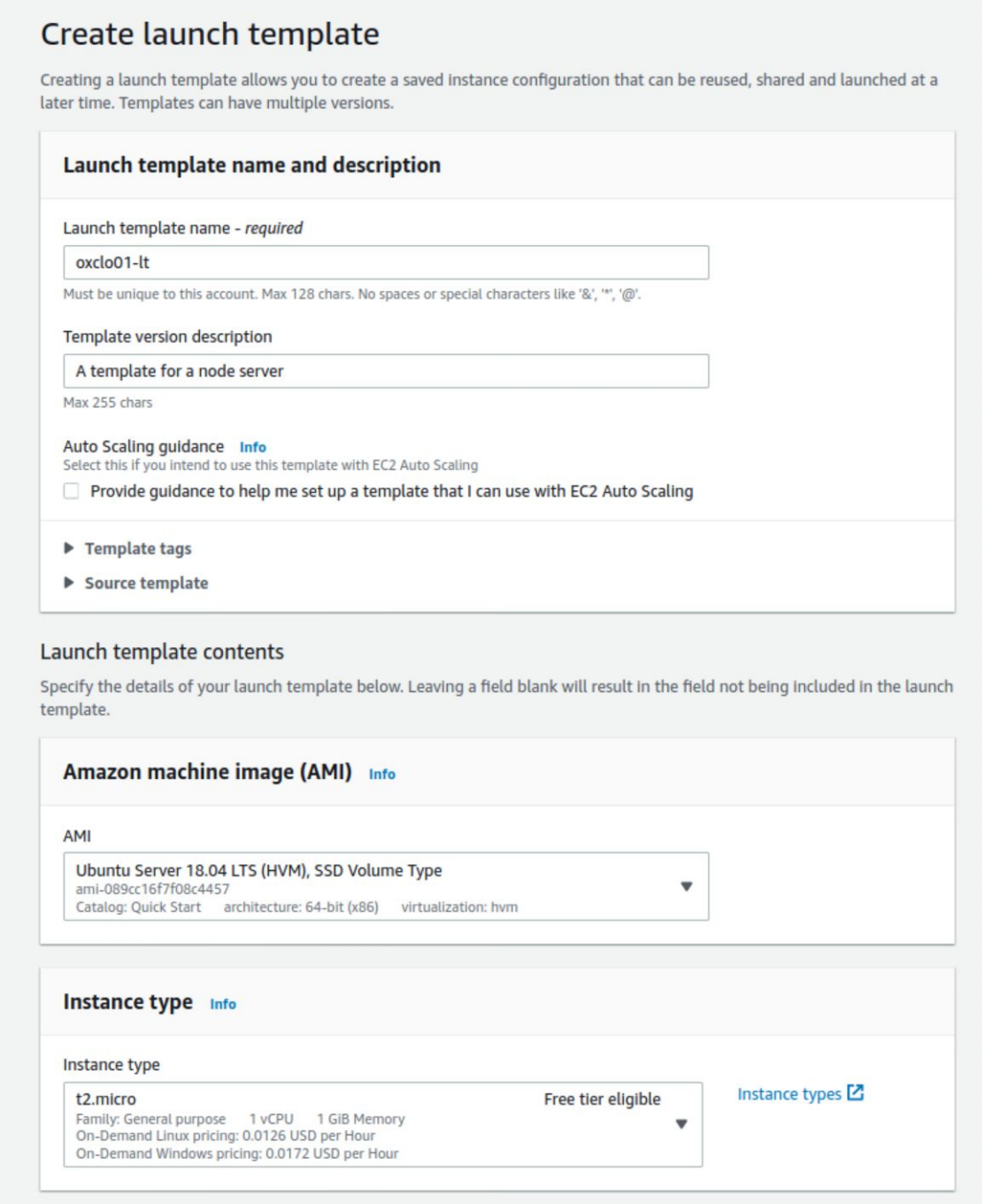
35. Use the name *userid-lt* (e.g. oxclo02-lt).

36. Give the template a description.

37. Choose AMI: Ubuntu Server 18.04 LTS (HVM), SSD

38. Choose Instance type t2.micro

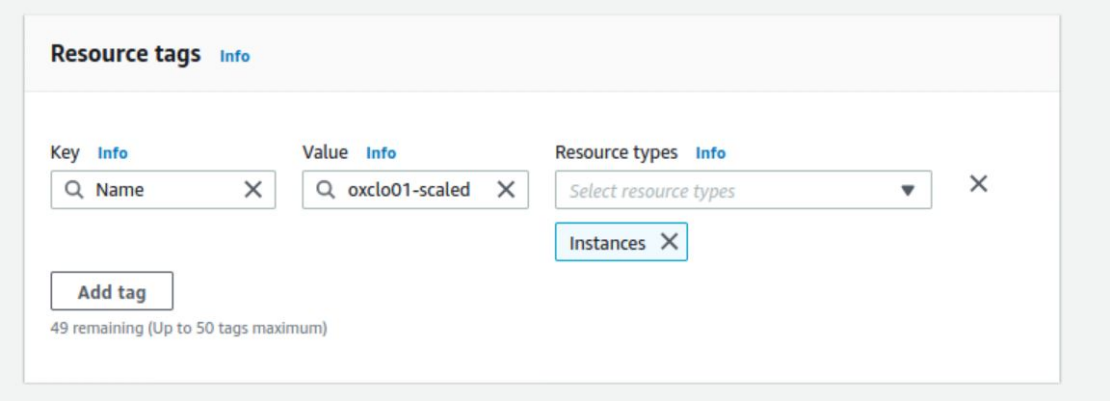
39. So far your screen should look like:



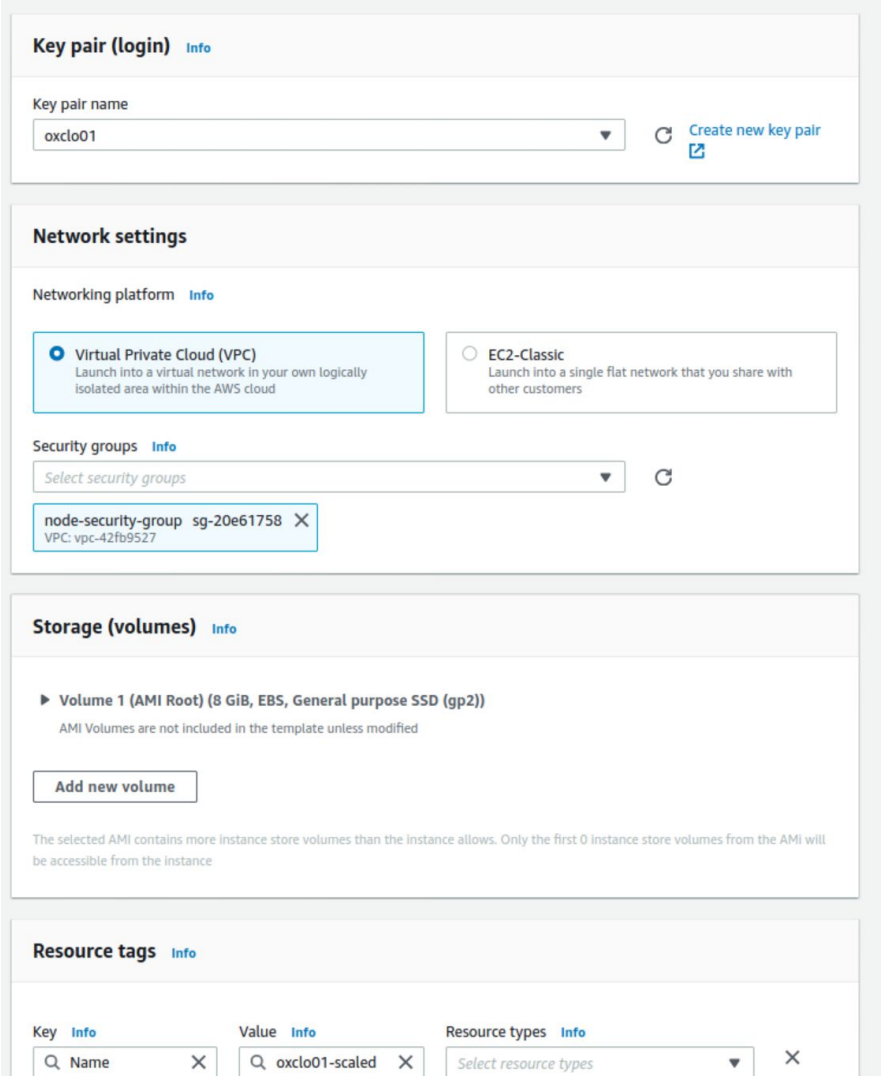
40. Choose **your** key pair

41. Choose the **node-security-group**

42. Add a ResourceTag of:
    Name: oxcloXX-scaled
    Tag instances



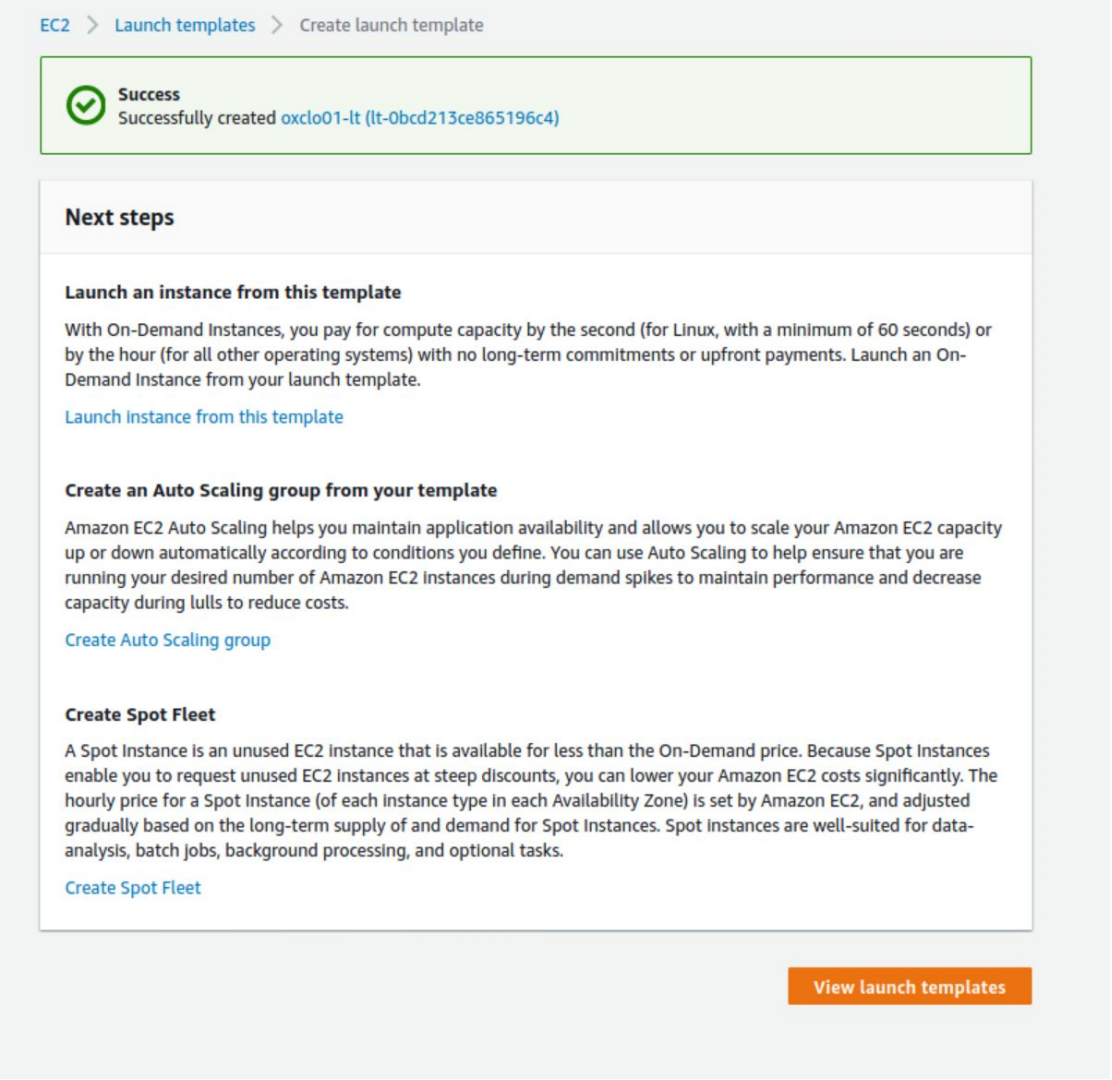
43. Your next part of the screen should now look like:



44. Expand **Advanced Details** and add the same **userdata** as before.
    http://freo.me/oxclo-userdata

45. Click **Create Launch Template**

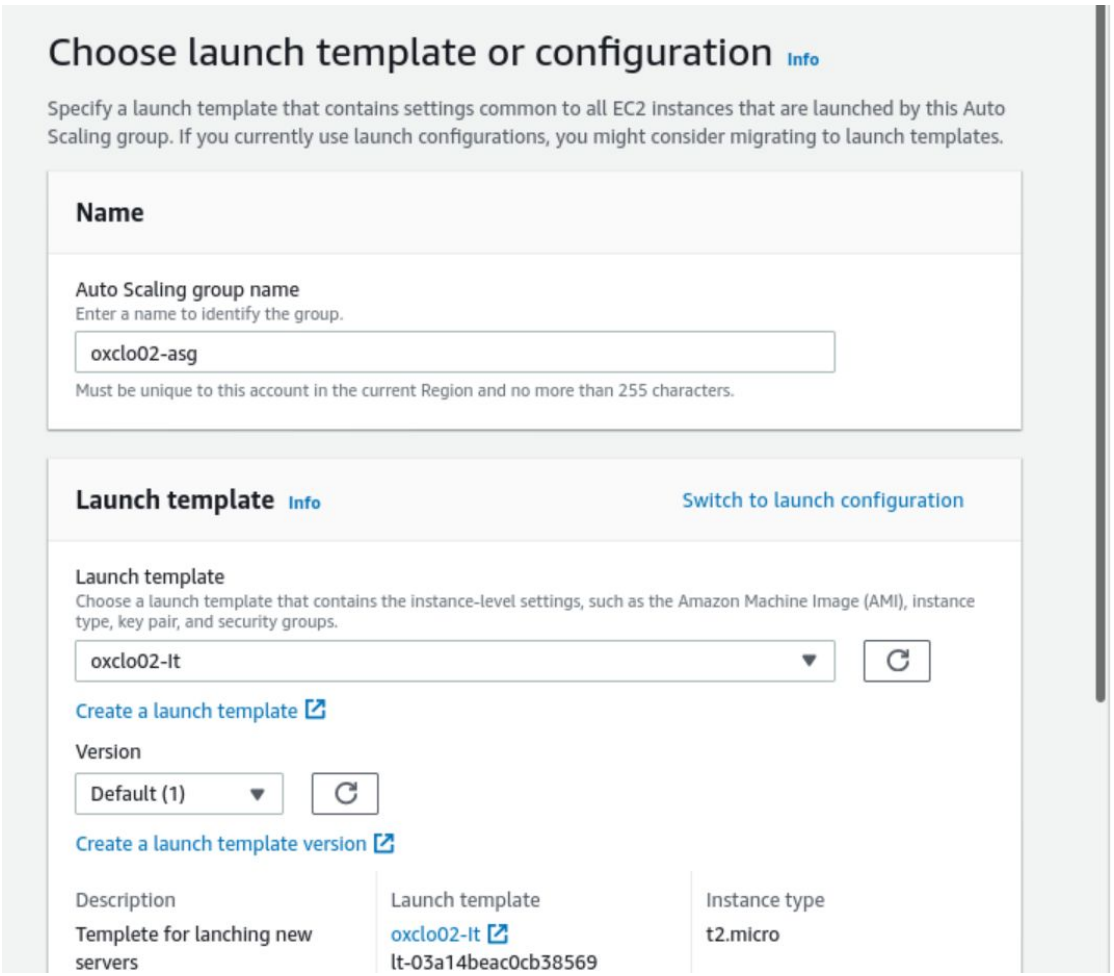
46. Once you have completed the process you can create an Auto Scaling group with this template.



Click **Create Auto Scaling Group**

47. Give the Group name as: *userid*-asg (e.g. oxclo02-asg).

48. Choose your launch template.
    You should see:



49. Click Next

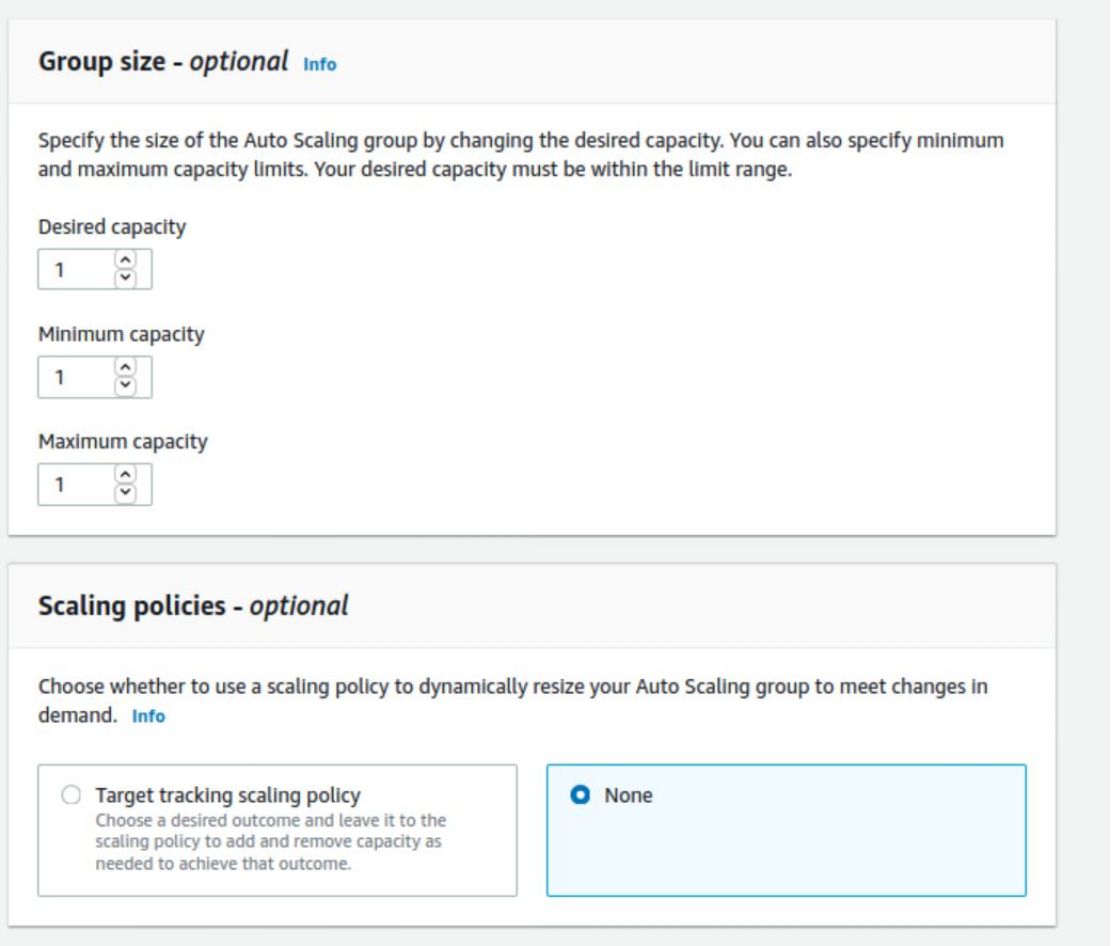
50. **Adhere to Launch Template**

51. Choose **a subnet** from the options that drop down when you select that box. Any one will do, or you can select multiple.

52. Click **Next**

53. You will see that there is a “Grace Period” of 300 seconds. Read the description.

54. Click **Next** to Configure Scaling Policies

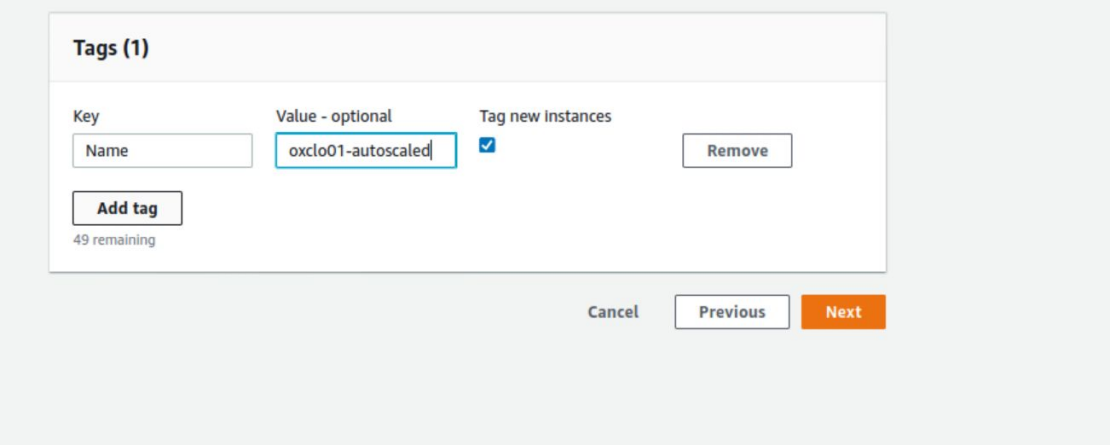


Leave the desired, min, max all at **1**, and Scaling Policy as **None**

55. Click **Next** to Configure Notifications
Leave alone

56. Click **Next** to Configure Tags

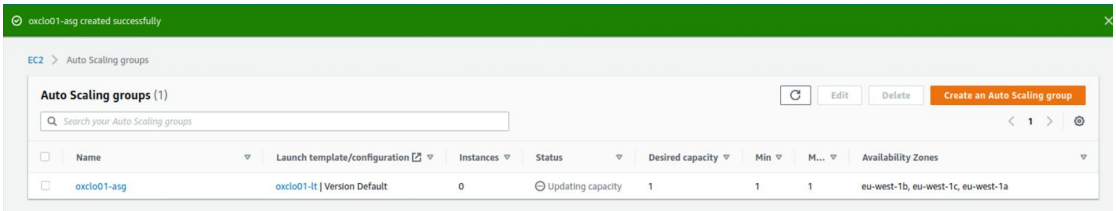
57. In the tags Key, specify **Name,** and in the Value field *userid-autoscaled* (e.g. oxclo01-autoscaled). Make sure that "tag new instances" is selected.



58. Click **Next** to review

59. Click **Create Auto Scaling Group**

You should see

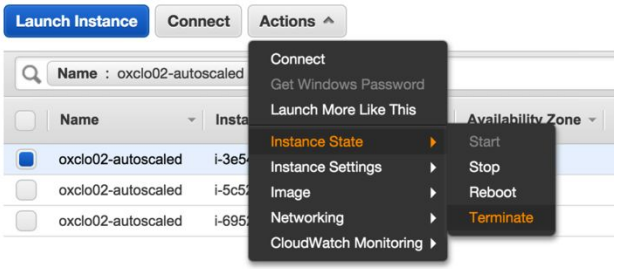


60. Now if you go to your EC2 Instances dashboard (**EC2 Dashboard -> Running Instances**), you should see a new instance starting up. Once it is started it will be tagged with your *userid-autoscaled* so you can see which ones are yours.

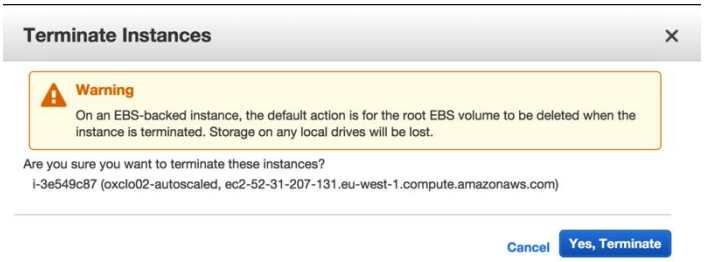
61. First check that your server is working properly by browsing http://<*ip-address*>:8080
You may need to be patient while the server starts up.

Don't continue to the next step until you get a proper response.

62. Now using the **EC2 Dashboard -> Running Instances** screen you can terminate this instance:



63. Click **Yes Terminate** on the next screen:



64. Now wait up (based on the previous grace period) 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).

65. Check the new server is correctly serving the data.

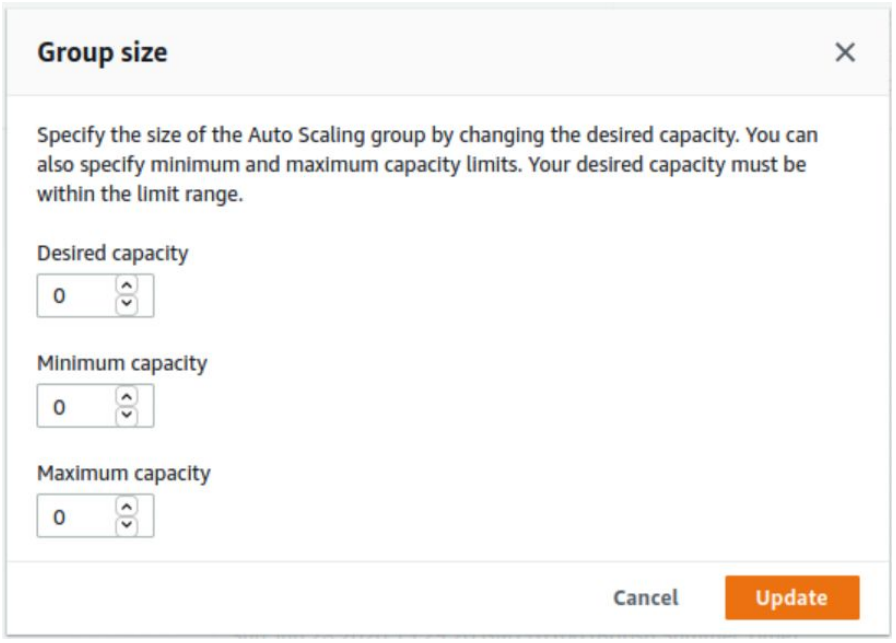
66. Is this newly created server much use to anyone? What would you need to do to make it more useful?

(PS I don't mean the app: let's assume that the app is in fact useful!)

67. What would you need to do to update the code on this system?

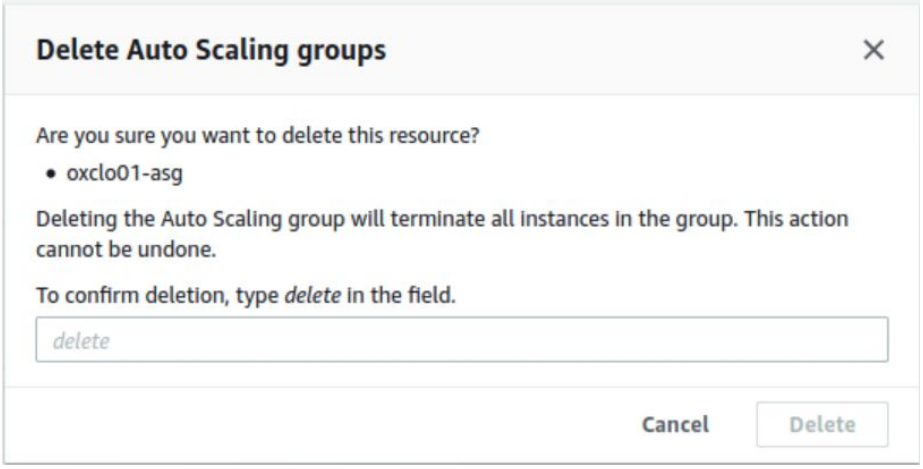
68. Go to **Auto Scaling Groups** in the left hand side of the EC2 menu. Hint it is right at the bottom!

69. You can turn off your ASG by changing the instances to 0. Click Edit on the Group Details and change the desired instances to zero (and the min/max)



70. Go and check in **Instances** that your autoscaled instance is being deleted!

71. If you just needed to stop the instance for a while, this is a simple method that doesn't involve deleting the ASG. However, we are now done, so delete the ASG!



72. In the next lab we will reconfigure this ASG to *auto-scale* instead of having just one instance, and then test it under load.

73. **Congratulations, lab complete.**