

LAUNCH CONFIGURATION AND AUTOSCALING GROUPS

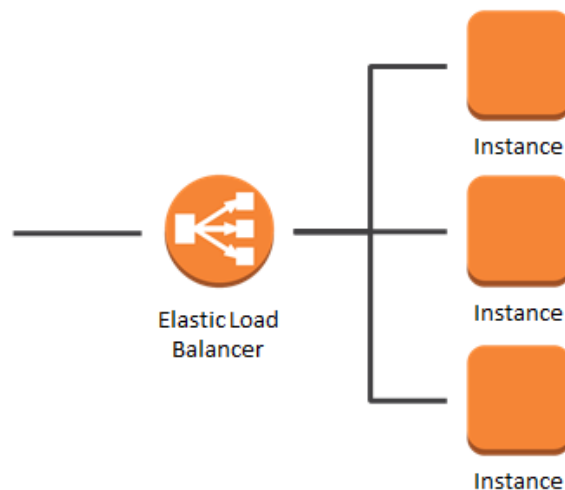
WHAT WE'LL COVER

In this section we are going to learn about **Launch Configuration** and **AutoScaling Groups**.

AWS **Auto Scaling** monitors your applications and automatically adjusts capacity to maintain steady, predictable performance.

Earlier on in the course we used a **Load Balancer** to spread traffic across instances in multiple **Availability Zones**, but we created the instances manually. With **AutoScaling Groups** we can automate this process based on criteria that we will define, for example when **CPU Utilization** is high, and then terminate unneeded instances when **CPU Utilization** is back to normal values.

Autoscaling can detect if there is heavy load on your website and run instances automatically to handle the load. The instances will register with the **Load Balancer** automatically which will spread the traffic across them.



The **Launch Configuration** is the instructions to automatically launch a new instance. Creating a **Launch Configuration** is similar to creating a new instance, only that it will not create it now, just save the steps to when it will need to create it.

AutoScaling Groups are groups of instances that can grow or shrink based on criteria which we define. They use the **Launch Configuration** to automatically create more instances when needed, also known as **Scale Up**, but they can also terminate instances when demand is low, which is also known as **Scale Down**.

1. Create an **Application Load Balancer**, name it **MyELB**, make it **internet-facing**, and add 4 availability zones. Don't register any instances at the moment.

Step 1: Configure Load Balancer

Basic Configuration

To configure your load balancer, provide a name, select a scheme, specify one or more listeners, and select a network. The default configuration is an Internet-facing load balancer in the selected network with a listener that receives HTTP traffic on port 80.

Name

Scheme ☒ internet-facing ☐ internal

IP address type

Listeners

A listener is a process that checks for connection requests, using the protocol and port that you configured.

Load Balancer Protocol	Load Balancer Port
HTTP	80

Add listener

Availability Zones

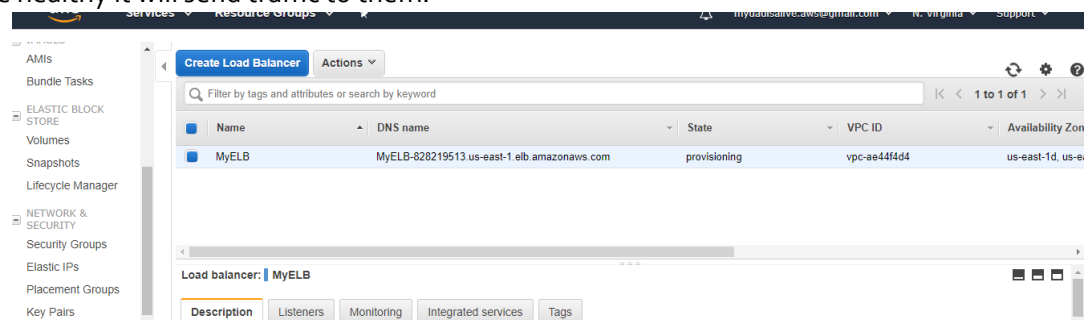
Specify the Availability Zones to enable for your load balancer. The load balancer routes traffic to the targets in these Availability Zones only. You can specify only one subnet per Availability Zone. You must specify subnets from at least two Availability Zones to increase the availability of your load balancer.

VPC

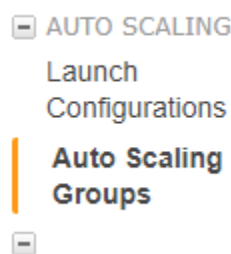
Availability Zones ☒ us-east-1a ☐ subnet-4506646b

Cancel Next: Configure Security Settings

2. When you're done hit **Create**
3. Reminder: The **Load Balancer** checks for the instances whether they are healthy or not if they are healthy it will send traffic to them.



4. Under **Auto Scaling** go to **AutoScaling Groups**, and then hit **Create Auto Scaling Group**



5. To create **AutoScaling Groups** you will be asked to create **Launch Configuration**.
6. When you create a **Launch Configuration** name it **MyAutoScalingGroup**, use the **Amazon Linux AMI**, attach the **IAM Roles** of **S3-Admin-Access**, and in **User data** type in the following commands taken from the last lecture when we learned about **User data** for running a bootstrap scripts.

```
#!/bin/bash
echo "LC_ALL=en_US.utf8" > /etc/environment
yum update -y
yum install httpd -y
```

```
aws s3 cp s3://mydadisalive-website/index.html /var/www/html/
systemctl start httpd
systemctl enable httpd
```

Here is how it should look:

7. Hit **Next**
8. Use default settings for **Storage**, and use our **Web-DMZ** for Security Group settings.
9. **Review** and then **Create launch configuration**

10. This is very similar to creating a new instance, but we're **NOT** creating one now, just creating the **Launch configuration** but just the recipe for it.
11. Now straight away we are requested to create the **Auto Scaling Group**
12. Name is **MyAutoScalingGroup** as well. Start with 2 instances.
13. For network just use the **default VPC**.

14. For subnets click inside and choose the same availability zones that you chose for the **Load Balancer**

1. Configure Auto Scaling group details 2. Configure scaling policies 3. Configure Notifications 4. Configure Tags 5. Review

Create Auto Scaling Group

The name of the launch configuration associated with this Auto Scaling group

Group name MyAutoScalingGroup

Launch Configuration MyAutoScalingGroup

Group size Start with 2 instances

Network vpc-ae44fd4 (172.31.0.0/16) (default) Create new VPC

Subnet subnet-4506646b(172.31.80.0/20) | Default in us-east-1a
subnet-f65166bc(172.31.16.0/20) | Default in us-east-1b
subnet-1035554c(172.31.32.0/20) | Default in us-east-1c
subnet-02dfbb65(172.31.0.0/20) | Default in us-east-1d Create new subnet

Each instance in this Auto Scaling group will be assigned a public IP address.

Advanced Details

15. The **Auto Scaling Group** will know to spread evenly the creation of new instances across the different Availability Zones.
16. In **Advanced Settings** you have to make some changes which are important
17. Under **Load Balancing**, make sure you check **Receive Traffic from one or more load balancers**
18. In **Target Groups** make sure you choose your target group which we named **MyTg**
19. For **Health Check Type** choose **ELB** which will let our Load Balancer do the checking for health of the instances.
20. **Health Check Grace Period** is basically the grace period to let an instance boot before start checking it. 300 seconds is the default which is 5 minutes. If you remember we run commands like **yum update -y** in our **Bootstrap script**, and then we copy some files from the bucket to our instance, and until the **httpd** will start it may take some time. This time is the **Health Check Grace Period**. For our example let's use **150 seconds**.

Advanced Details

Load Balancing Receive traffic from one or more load balancers Learn about Elastic Load Balancing

Classic Load Balancers

Target Groups MyTG

Health Check Type ELB EC2

Health Check Grace Period 150 seconds

Monitoring Amazon EC2 Detailed Monitoring metrics, which are provided at 1 minute frequency, are not enabled for the launch configuration MyAutoScalingGroup. Instances launched from it will use Basic Monitoring metrics, provided at 5 minute frequency. Learn more

Instance Protection

Service-Linked Role AWSServiceRoleForAutoScaling View Role in IAM

Cancel Next: Configure scaling policies

21. Hit **Next** to configure **Scaling Policies**
22. Here you should choose **Use Scaling Policies** to adjust the capacity of this group
23. Then choose **scale between 2 to 3 instances**. This will be the minimum and the maximum size of this group.
24. Name is **Scale Group Size**, choose **Metric Type: Average CPU Utilization**

25. **Target Value: 75**

26. **Instance need: 150 seconds to warm up**

- ☐ Keep this group at its initial size
 - ☒ Use scaling policies to adjust the capacity of this group
- Scale between and instances. These will be the minimum and maximum size of your group.

Scale Group Size

Name:

Scale Group Size

Metric type:

Average CPU Utilization

Target value:

75

Instances need:

150

seconds to warm up after scaling

Disable scale-in:

☐

[Scale the Auto Scaling group using step or simple scaling policies](#)

Cancel

Previous

Review

Next: Configure Notifications

27. Now hit **Next** to configure notifications

Send a notification to:

CPU-High (mydadisalive@gmail.com)

create topic

Whenever instances:

☒ launch

☒ terminate

☒ fail to launch

☒ fail to terminate

Add notification

28. This will send a notification whenever an instance **launches, terminates, fails to launch, or fails to terminate** to the recipients we have chosen.

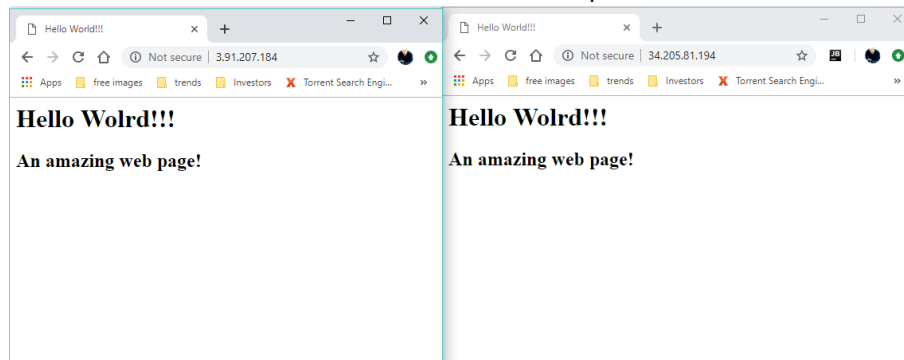
29. Create next, and then add tags, and then **Create Auto Scaling Group**

30. If we go back to our instance dashboard we can see already two instances are being provisioned. This is because we used a minimum of 2 in the **Scaling Policies**

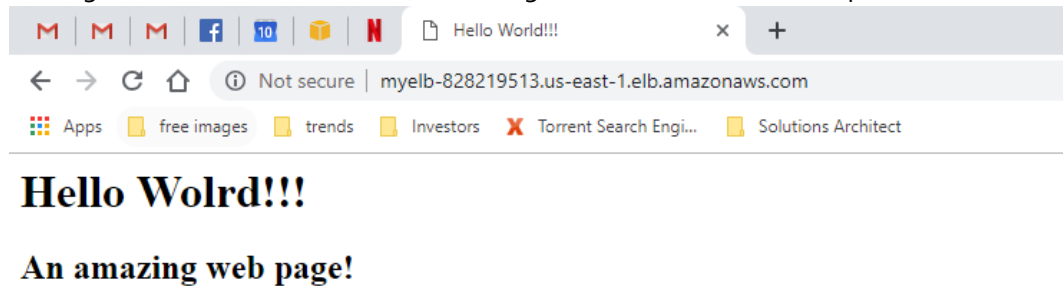
31. It will take them some time to run the **bootstrap scripts** that if you recall are now installing updates, copying a page from S3, and installing, starting and enabling **httpd**

32. We can see in the description that they are on different **Availability Zones**, they have different **Public IP Addresses**

33. Let's open a new **Web Browser Window** for each of those public addresses



34. As you can see they are both showing the same page exactly, which means that they were both automatically installed and configured by the **Launch Configuration** we set up and using the **bootstrap** script.
35. Now let's go to our **Elastic Load Balancer** and grab the **Public DNS** and open it in a browser.



36. As you can see it's going to one of the two instances which means they all registered to it properly. You can also see that in **Target Groups**

registered targets

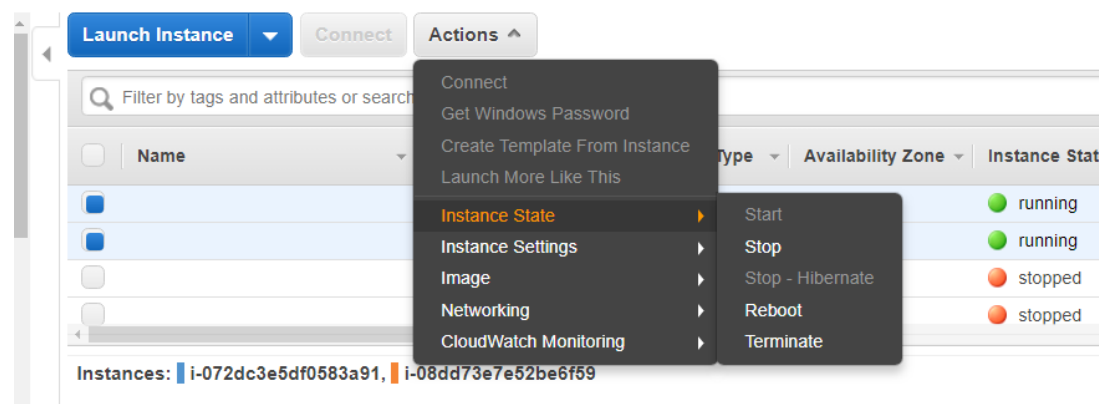
Instance ID	Name	Port	Availability Zone	Status
i-08dd73e7e52be6f59		80	us-east-1d	healthy ⓘ
i-072dc3e5df0583a91		80	us-east-1a	healthy ⓘ

Availability Zones

SIMULATING AND TESTING FAILURE

What happens if we kill an instance? Well, in our policies we said minimum number of instances should be 2. Let's try to terminate the instances and see what happens.

37. Choose the instances and then **Actions** and **Terminate**. We do this to simulate a situation of a crash that can occur in real-life scenarios.



38. Now wait and see what happens.
39. Once the **ELB** detects that the instances are not **Healthy** (as we terminated them), two new instances will be automatically provisioned (again, we set the minimum number to be 2).

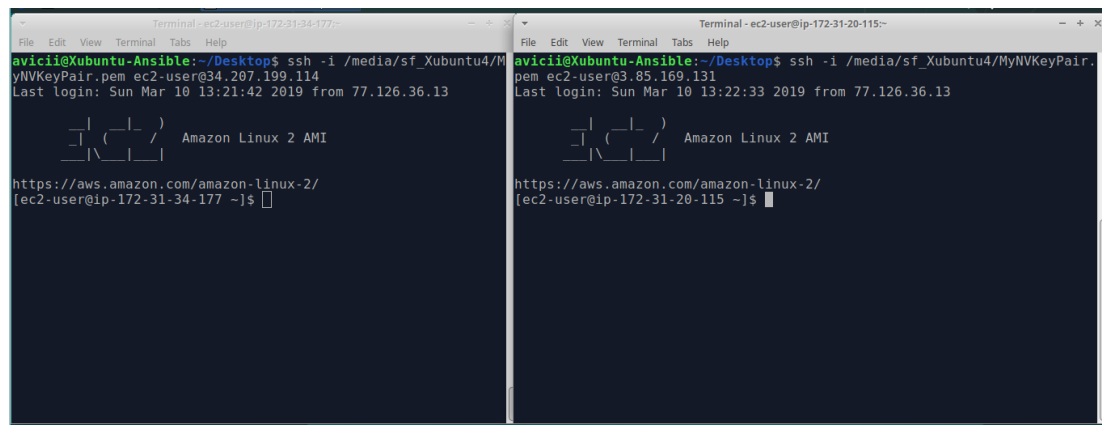
	i-0d0af81c1c3132d55	t2.micro	us-east-1c	running	2/2 checks passed
	i-0d975cb2f0fe26163	t2.micro	us-east-1b	running	2/2 checks passed

40. The chances that two failures will occur at the same time are not very high. However we can see that the **Auto Scaling Group** automatically takes care of it and provisions the necessary instances so that the downtime our users will experience is minimal.
41. Our users will use the public DNS which is now <http://myelb-828219513.us-east-1.elb.amazonaws.com/> but in real-life will be something similar to www.example.com

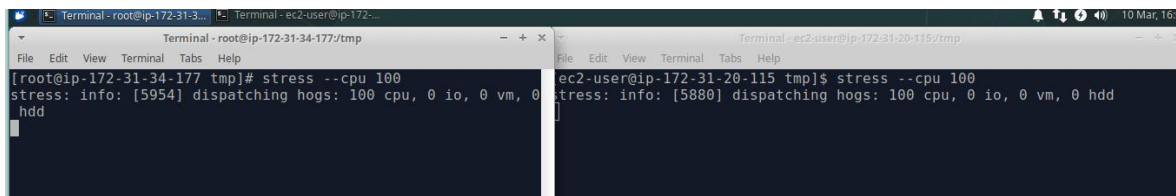
STRESS TEST

Now let's try to simulate a situation of heavy traffic on our website. If you recall we used the metric of **CPU High** to measure that. To simulate **High CPU Scenario**, we will use the Linux command **stress** which will artificially create stress over on the CPU and we will use it to make the **CPU Utilization** go **above 75%** on both instances, which is supposed to trigger our policy even and scale up, which means provision another instance. Then, we'll stop the stress commands which we will see return the **CPU Utilization** levels to normal levels, and the **AutoScaling Group** will as a result terminate one of the instances, but still keep a minimum of 2, as we defined in the policy.

42. Open two terminal windows.
43. You currently have two servers running. On one of the windows **SSH** to the first instance, and on the other windows **SSH** to the second one.












44. We will install **stress** command in each terminal window using these commands (stress is not part of the **yum** repositories so we'll download it using **wget** then use **yum localinstall** command to install it locally on the instance.
45. Install stress by going here <https://gist.github.com/mikepfeiffer/d27f5c478bef92e8aff4241154b77e54>
46. Now on each window run **stress --cpu 100**. This will create a very high load on the CPU of 100% on each of our instances



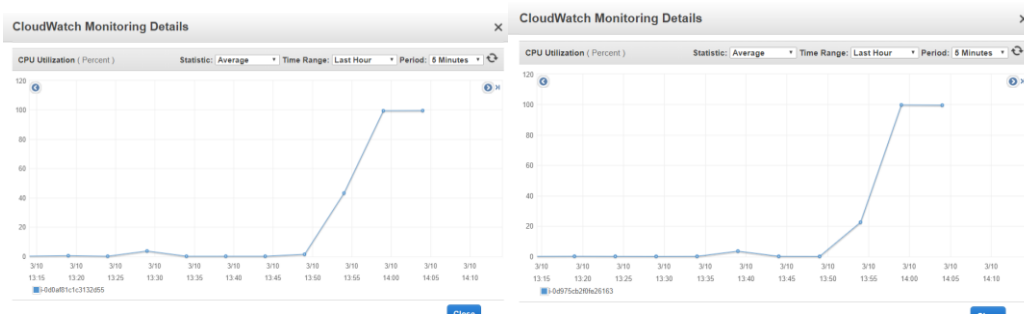
47. Now let's go back to our console and see what's happening there. Let's go to our monitoring and see the CPU Utilization graphs of our instances.

48. After a few minutes we will be able to see a new instance is being provisioned.

	i-082cb261e0bd92979	t2.micro	us-east-1a		pending		Initializing
	i-0d0af81c1c3132d55	t2.micro	us-east-1c		running		2/2 checks passed
	i-0d975cb2f0fe26163	t2.micro	us-east-1b		running		2/2 checks passed

49. This is because our policy rule was CPU and 75%. And since both instances run at 100% then the average is clearly over 75%. The **Auto Scaling Group** launches a new instance. Remember that the maximum we defined was 3 instances. Other than that, no more instances will provision. However, theoretically if we set this up to 100, then scaling up could continue.

50. We can visually see the spiking in **CPU Utilization** using **CloudWatch** graphs. This visual representation is important. It's happening in 2 of our instances since we ran the **stress** command on two of them.

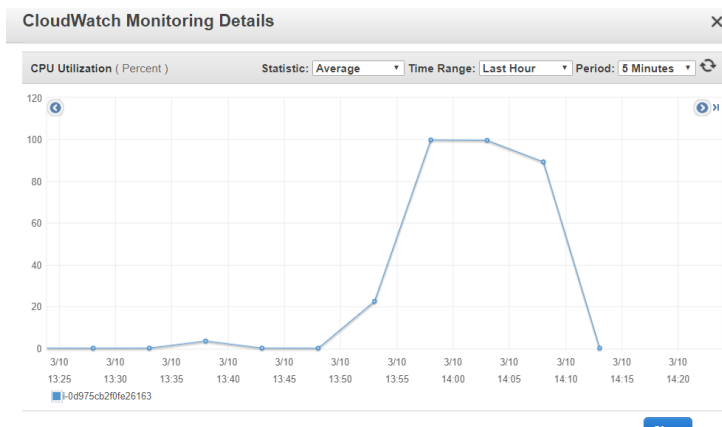


51. Now let's stop the stress command. This will stop creating heavy load on the CPUs. When the **Auto Scaling Group** will detect the change, it's supposed to terminate one of the instances and leave us only with two (which is the minimum we defined) instances running. Just go to the terminal windows and press ctrl-c in both of them to stop the stress command from running.

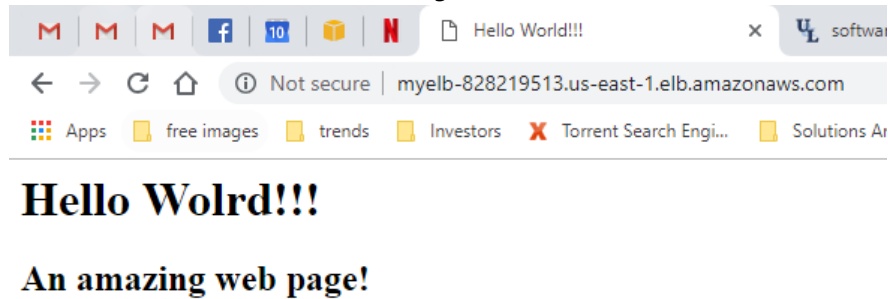
```
Terminal - root@ip-172-31-34-177/tmp
[ec2-user@ip-172-31-20-115 tmp]$ stress --cpu 100
stress: info: [5954] dispatching hogs: 100 cpu, 0 io, 0 vm, 0 hdd
^C
[ec2-user@ip-172-31-20-115 tmp]$
```

52. Now let's wait and in the meantime refresh the graphs page to see the change in the AWS web console taking effect. It takes time for the changes to occur because monitoring occurs only every 5 minutes.


53. After some time we can see that the CPU is back to normal levels



54. Note that the end user continues to get the same user experience as he is not aware of what's happening behind the scenes. He continues to get service from our website



55. When there is no need the AutoScaling group will terminate unneeded instances

	I-0d0af81c1c3132d55	t2.micro	us-east-1c	 shutting-do...
---	---------------------	----------	------------	--

EXERCISE

56. Change the min and max values for autoscaling group.
57. Terminating: Once you're done experimenting delete the Auto Scaling Group. It will delete all the instances with it.

SUMMARY

In this section we have learned what is AWS **AutoScaling**. We have created **Launch Configuration** and use **Auto Scaling Groups** to automatically scale up and scale down our website. We simulated a failure situation and a heavy traffic situation using the **stress** command, and verified that our setup for **AutoScaling** is working, which it did. **Auto Scaling** is very important for system architecture because it creates a seamless experience for the end user and handles the necessary underlying work for the DevOps engineer automatically, once set correctly.