

## Lab 2: Using Amazon EC2 Auto Scaling groups to deploy ephemeral environments v1.0.5

Sunday, August 9, 2020 7:05 PM

### Lab 2:

- Step 5, I didn't have the Create Backup Plan. This system took me straight to the dashboard and I had to go into Manage Backup Plans to get the Create Backup Plan button. Can we get a note about this added to this step?
- Step 6, GUI changes, and the steps are very confusing for learners. Need to have the Lifecycle bullet changed to reflect the new GUI for setting the retention period.
- Step 12, GUI changes, and the steps are very confusing for learners. Need to have the Lifecycle bullet changed to reflect the new GUI for setting the retention period.
- Step 57, GUI changes, and the steps are very confusing for learners. Need to have the Lifecycle bullet changed to reflect the new GUI for setting the retention period.

# Lab 2: Cost optimization: Using Amazon EC2 Auto Scaling groups to deploy ephemeral environments



In this lab, you will experiment with how to reduce the resources cost by deploying ephemeral test environments automatically when needed. Then, you will create daily backups of the production environment using AWS Backup. Next, you will then use those daily backups to create a test environment that is launched and terminated as needed with an Amazon EC2 Auto Scaling Group. Finally, in order to make sure that drift between the production and test environments are kept to a minimum, you will use an AWS Lambda function to update the Auto Scaling Group and schedule the function to run daily.

### Objectives

After completing this lab, you will be able to:

- Automate snapshot creation and retention using AWS Backup.
- Create a Launch Configuration and Auto Scaling Group with a desired schedule.

- Automate snapshot creation and retention using AWS Backup.
- Create a Launch Configuration and Auto Scaling Group with a desired schedule.
- Create and test an AWS Lambda function to update AMI in the auto-scaling group.
- Configure a custom schedule with Amazon CloudWatch to trigger the lambda function.

### Prerequisites

This lab requires:

- Access to a computer with Internet Access and Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat)
- The lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the student guide.
- An Internet browser such as Chrome, Firefox, or IE9 (previous versions of Internet Explorer are not supported)
- The ability for your computer to access various websites using HTTP and HTTPS protocols over port 80 and port 443

### Duration

This lab will require **60** minutes to complete.

## Start Lab

1. At the top of your screen, launch your lab by choosing [Start Lab](#)

This starts the process of provisioning your lab resources. An estimated amount of time to provision your lab resources is displayed. You must wait for your resources to be provisioned before continuing.

**i** If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. Open your lab by choosing [Open Console](#)

This opens an AWS Management Console sign-in page.

3. On the sign-in page, configure:

- **IAM user name:** `awsstudent`
- **Password:** Paste the value of **Password** from the left side of the lab page
- Choose [Sign In](#)

**⚠ Do not change the Region unless instructed.**

## Common Login Errors

Error: You must first log out

### Amazon Web Services Sign In

You must first log out before logging into a different AWS account.

To logout, [click here](#)

If you see the message, **You must first log out before logging into a different AWS account:**

- Choose **click here**
- Close your browser tab to return to your initial lab window
- Choose [Open Console](#) again

## Introduction

Previously, you established reactive and proactive controls supporting consistent resource tagging. You are able to use cost and utilization reports in AWS to determine that there was a large amount of waste associated with running non-production environments. Since these environments were running when not in use, significant manual work was required to maintain configuration consistency with production.

In this lab, you will want to create test environments at the start of each workday and terminate the resources at the end of each workday, all based on the latest production configurations.

To start, you have two servers, one production web app server and one test web app server.



Amazon EC2

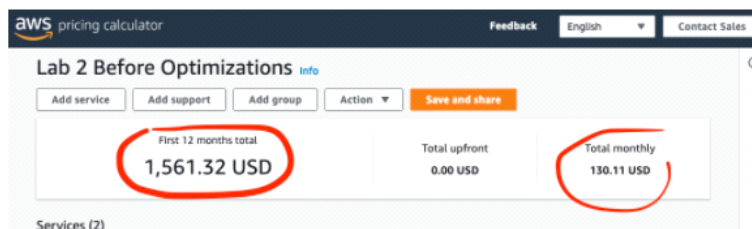
Production Web App Server  
**Name:** APAPP-P



Amazon EC2

Test Web App Server  
**Name:** APAPP-T

Below is a screenshot of the AWS Pricing Calculator showing the cost of the environment at the start of the lab. The prices are as of 5/12/2020. You can select the link below see an estimate online with current prices.

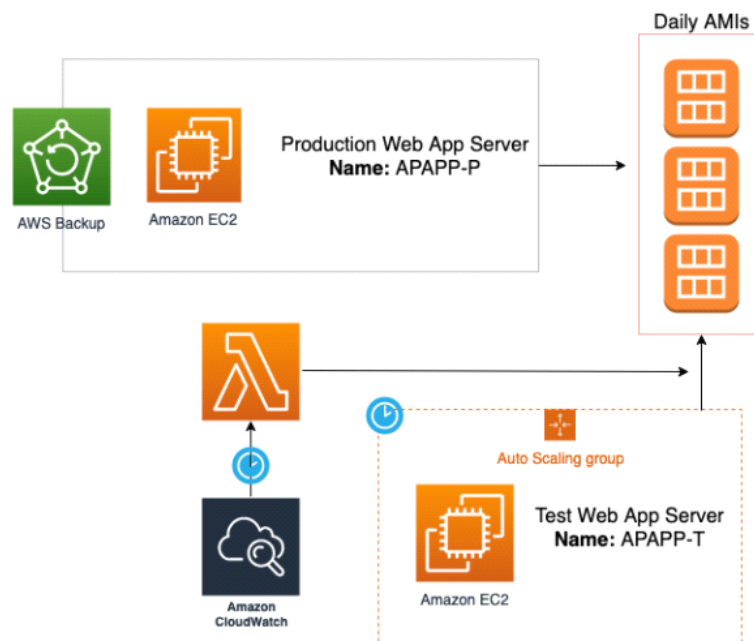


<b>Amazon EC2</b> Region: US East (N. Virginia)		Edit Action
<b>Advanced estimate</b>  Operating system (Linux). Storage for each EC2 instance (General Purpose SSD (gp2)). Storage amount (8 GB). Snapshot Frequency (No snapshot storage). Amount charged per snapshot (3 GB). Inbound (from: Not selected) 0 TB per month. Outbound (to: Internet) 1 GB per month. Intra-Region 0 TB per month.		<b>Test Instance</b>  Monthly: 62.85 USD Upfront: 0.00 USD
<b>Amazon EC2</b> Region: US East (N. Virginia)		Edit Action
<b>Advanced estimate</b>  Operating system (Linux). Storage for each EC2 instance (General Purpose SSD (gp2)). Storage amount (8 GB). Snapshot Frequency (No snapshot storage). Amount charged per snapshot (3 GB). Inbound (from: Not selected) 0 TB per month. Outbound (to: Internet) 50 GB per month. Intra-Region 0 TB per month.		<b>Production Instance</b>  Monthly: 67.26 USD Upfront: 0.00 USD

<https://calculator.aws/#/estimate?id=c6501c330d2a9d386f8f16ea1c9b36b6e03172f9>

⚠ The lab uses t2.micro instances to illustrate a point while keeping costs down for the lab environments. The calculator uses more real world examples of c5.large. The estimates provided use instance types which are more realistic for a customer deployment, yet the percentage of cost savings between the start and ending estimates remain the same.

You will use AWS Backup to create daily backups of your production environment. Once AWS Backup generates your first Amazon Machine Image (AMI), you can use it to build a Launch Configuration and an Amazon EC2 Auto Scaling Group, which will create your Test environment. Next, to ensure your Launch Configuration and Auto Scaling Group are always using the latest AMI from your Backup process, you will create an AWS Lambda function. This identifies the latest AMI ID, creates a new Launch Configuration, and updates the Auto Scaling Group to reference the new Launch Configuration. Finally, an Amazon CloudWatch Event will trigger the Lambda function's execution based on a CRON schedule that will make these updates prior to the recurring Scheduled Actions in your Auto Scaling Group.



## Task 1: Set up AWS Backup to protect your production server


First, schedule the creation of an Amazon Machine Image (AMI) of your production server. You pay for the storage of the EBS snapshots that your AMI uses, therefore you want to make sure that you only keep the snapshots for the amount of time required by your company's policies and no longer. Your company's current policy for the web application states that you must keep daily snapshots of the web app server for 2 weeks. Instead of writing something custom to do this, you will use AWS Backup instead.

AWS Backup is a fully managed backup service that centralizes and automates the backup of data across AWS services in the cloud and on premises. Using AWS Backup, you can configure backup policies and monitor backup activity for your AWS resources in one place without having to create custom scripts and manual processes. With a few selections on the AWS Backup console, you can create backup policies that automate backup schedules and retention management. This helps to avoid unnecessary costs associated with storing unneeded resource backups while you to meet your business and regulatory backup compliance requirements.

Your Compliance and Security teams have already created a Backup Vault and IAM Roles for you use. You will configure the backup plan in the steps below.

### Task 1.1: Configure AWS Backup to protect the Web Application production resources

In this task, you will configure a backup plan to create daily backups and retain them for 2 weeks. You also want to tag your backups properly so you can filter them with future scripts and allocate their costs appropriately.

4. On the navigation bar, choose **Services** , and choose **AWS Backup**.
5. Choose **Create Backup plan**, then select **Build a new plan** enter `prod-web-app-daily` as the **Backup plan name**.
6. In the **Backup rule configuration** section, configure the following settings:
  - For **Rule name**, enter `daily`
  - Under **Lifecycle**, for **Expire**, select **Weeks after creation** and enter `2`
  - For **Backup vault**, select `Lab-Vault`
  - Expand **Tags added to recovery points** and add the following:

• Key: `Department` Value: `Finance`

- **Key:** `Department` **Value:** `Finance`

Choose  for each additional tag.

- **Key:** `Application` **Value:** `Accounts Payable`
- **Key:** `Environment` **Value:** `Prod`

**Note:** Tags are case-sensitive, remember to enter as suggested.

## 7. Choose

Now that you have the backup plan, you need to assign resources to be protected. You can either use tags or assign resources manually. Tags are much more scalable, if for example the application team decides to start using other services like Amazon DynamoDB or Amazon EFS in the future, it would be automatically protected if they tag it appropriately. The application team already created a backup tag for you to use so you will configure it.

## 8. In the card **Resource assignments**, choose , and configure:

- **Resource assignment name:** `prod-web-ap-resources`

Your backup plan needs permission to write to the Backup Vault as well as backup your resources. You need to specify which IAM role the job should use.

- **Choose an IAM role:** and select `LabBackupServiceRole`
- For **Assign resources**, enter the following:

- **Assign by:** `Tags` **Key:** `Backup` **Value:** `Web-App-Prod`

**⚠** Do not forget that Tags are case sensitive so it should be **Backup** and **Web-App-Prod** exactly.

## 9. Choose

# Task 1.2: Create an on-demand backup of the production web app resources

Your daily backup plan is set up to protect the application in the future, but now you need a backup of the production instance to create the initial ephemeral test environment. AWS Backup has functionality to create on-demand backups for use cases such as this. The advantage of using AWS Backup instead of creating an AMI in the Amazon EC2 console is that AWS Backup will manage the retention of that AMI for you.

## 10. On the left navigation pane, select **Dashboard**.

## 11. Choose

## 12. Configure the following settings:

- For **Resource type**, select `EC2`



- For **Instance ID**, select the Instance ID that appears by the tag name of `Web-App-Prod` from the dropdown list.
- Under **Lifecycle**, for **Expire**, select **Weeks after creation** and enter `2`
- For **Backup vault**, select `Lab-Vault`
- In **IAM Role** section, choose **Choose an IAM role:** and select `LabBackupServiceRole`
- Expand **Tags added to recovery points** and add these tags to the backup:
  - **Key:** `Department` **Value:** `Finance`

Choose `Add tag` for each additional tag.

  - **Key:** `Application` **Value:** `Accounts Payable`
  - **Key:** `Environment` **Value:** `Prod`

13. Choose **Create on-demand backup**

This on-demand backup will take approximately 7 minutes to fully complete, but you do not have to wait. The AMI it created is available within 2-3 minutes.

14. Refresh the page and proceed to the next step after the backup job's **Status** changes from **Created** to **Running**.

## Task 2: Create your Auto Scaling group to automate the test environment

Next, you need to create a test environment that will be launched and terminated automatically. Amazon EC2 Auto Scaling is perfect for this.

Amazon EC2 Auto Scaling helps ensure you have the correct number of Amazon EC2 instances available to handle the load for your application. You will create collections of Amazon EC2 instances, called Auto Scaling groups. Scaling policies can ensure that your environment size can support demand without over provisioning resources throughout the workday. Recurring Scheduled Actions will allow you to launch the environment each morning and terminate it every evening.

Before you create the new test environment, there's no need to keep paying for the old system. For now, stop the old system. Stopping an EC2 instance will keep you from paying for the compute capacity of the instance but you will still pay for the EBS storage. You will want to terminate the instance after your application team determines that there is no relevant data on it.

❗ Instead of waiting for someone to go back and terminate the instance, create an AWS Backup on-demand backup of the instance with an agreed upon retention window and then terminate the instance that you just backed up. That is not a task for this lab, but you have steps above to do that.

## Task 2.1: Stop the static test instance

15. On the navigation bar, choose **Services ▾**, and choose **EC2**.
16. Choose **Instances** from the left navigation pane.
17. To stop your application test server, choose the instance with the name **Web-App-Test** and follow:

- Choose **Instance state**
- Select **Stop instance**
- Confirm by choosing on **Stop**

## Task 2.2: Create the new Auto Scaled test instance

A launch configuration is an instance configuration template that an Auto Scaling group uses to launch EC2 instances. When you create a launch configuration, you specify information for the instances. This includes the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups, and a block device mapping. If you have launched an EC2 instance before, you specified the same information in order to launch the instance.

You can use your launch configuration with multiple Auto Scaling groups. However, Auto Scaling groups can only use one launch configuration at a time, and you cannot modify a launch configuration after you have created it. To change the launch configuration for an Auto Scaling group, you must create a new launch configuration and then update your Auto Scaling group with it.

There should be as little difference as possible between your production server and ephemeral test environment. In the User data for your launch configuration, you are going to change the application home page to reflect that this is the test environment and not the production system. A more complicated application would potentially require more changes to make sure that the system points to the proper resources but your simple demonstration only requires one file to be changed.

For this step, you will be using the existing IAM role and Security Group for your Launch Configuration and Auto Scaling Group that the Test server was previously configured for.

18. Scroll to the bottom of the left navigation pane, and under **Auto Scaling**, choose **Launch Configurations**



19. Choose **Create launch configuration**

20. Configure the following settings:

⚠ Leave the other settings as is.

- **Launch configuration name:**

- For **Name:** `Web-App-Test-LC-V1`

⚠ Launch configurations for this function should be named with the following pattern **[LC Name]-V[version number]**. Your AWS Lambda function will use the **V** character to create new versions when it runs.

- **Amazon machine image (AMI):**

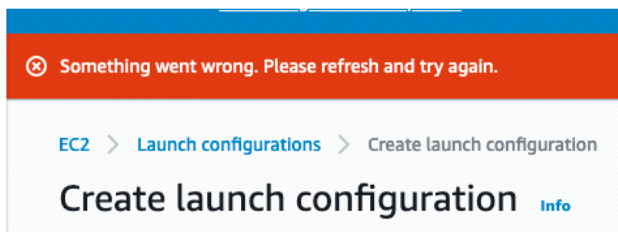
- For **AMI**, press **Choose an AMI**.
- Under **My AMIs**, select the AMI that the name begins with **AwsBackup**.

⚠ If you do not see any AMIs listed under **MY AMIs**, make sure that one is available. Navigate to the EC2 dashboard and select AMIs on the left navigation pane.

- **Instance type:**

- For **Instance Type**, press **Choose Instance Type**
- Select `t2.micro`, and press **Choose**

ⓘ Important: You can ignore the "Something went wrong" error. This error is shown because you do not have permissions to view spot pricing in this lab.



- **Additional configuration - optional:**

- Expand ▶ **Advanced Details**.
- For **User data:** enter the following text:

```
#!/bin/bash
sed -i 's/Production/Test/g' /var/www/html/index.php
```

- **Security Groups:**

- Choose **Select an existing security group**.
- Select the security group with the description `Security Group for the Test App Server`

- **Key pair (login):**

```
#!/bin/bash sed -i 's/Production/Test/g' /var/www/html/index.php
```

- For **Key pair options**, select **Proceed without a key pair**.
- Check ☒ for **I acknowledge that I will not be able to connect to this instance unless I already know the password built into this AMI**.

21. Choose **Create launch configuration**

Now that you have your Launch Configuration you can create the Auto Scaling group that will launch your test environment.

22. Select the launch configuration named **Web-App-Test-LC-V1**.

23. Choose **Actions** ▾

24. Choose **Create Auto Scaling group**.

25. Configure the following and leave the other settings as is:

- **Auto Scaling Group Name:** **Web-App-Test-ASG**

⚠ The Group Name can be anything you would like but for this lab please use **Web-App-Test-ASG** to make sure future steps work as expected.

26. Choose **Next**

27. For **Network**, configure the following:

- **VPC:** **Lab-VPC**
- **Subnets:** **Lab-Pub-a**

28. Choose **Skip to Review**

29. For **Step 6: Add tags** choose **Edit**

30. Choose **Add tag** for each tag and configure the following:

- **Key:** **Department** **Value:** **Finance**
- **Key:** **Application** **Value:** **Accounts Payable**
- **Key:** **Environment** **Value:** **Test**
- **Key:** **Name** **Value:** **Web-App-Test**

31. Choose **Next**

32. Choose **Create Auto Scaling Group**

## Task 2.3: Schedule your test environment to only be provisioned when needed

During the previous steps, you set the Auto Scaling group to use a minimum, maximum and desired capacity of one, which is correct for when you want

your test environment available. If you only wanted to create an environment when your application teams need it, you can change these values manually. However, you received instruction the application team needs the environment to be available 9:00 AM EST until 5:00 PM EST Monday-Friday.

⚠ When you use Cron to schedule your tasks, it is in UTC which is why you entered 14 for the hour instead of 9.

37. Choose **Create**

Now create the task to terminate the environment every afternoon during the work week.

38. Choose **Create Scheduled Action**

39. Configure the following settings:

- **Name:** TurnOff-M-F-5PM-EST
- **Desired Capacity:** 0
- **Min:** 0
- **Max:** 0
- **Recurrence** Select **Cron** and enter 0 22 \* \* MON-FRI

40. Choose **Create**

## Task 3: Create an AWS Lambda function to automate updating the Auto Scaling group

You now have the framework for your ephemeral test environment built, but it is still just creating a static test environment from your initial on-demand backup. You need a way to update the launch configuration to the latest Amazon Machine Image (AMI) created for your production instance. One of your system administrators wrote a python script for you that expect the name of the Auto Scaling group to pass to it. It then does the following:

- Read the Launch Configuration used by the Auto Scaling group.
- Pull the instance name from the tags on the AMI that the Launch Configuration is using.
- Get a list of all AMIs that are available for that instance and sort by the CreationDate to return the latest AMI id available.
- If your Launch Configuration is not already using the latest AMI, it creates a new Launch Configuration using values from the old Launch Configuration, replacing the AMI id and incrementing the version number on that Launch Configuration name.
- Updates the Auto Scaling group to use the latest Launch Configuration.

You now need a cost-efficient way to run this script. You could run it on an EC2 instance but that does not make sense from a cost or efficiency point of view. AWS Lambda is the correct service for this task.

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda runs your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running. With AWS Lambda, you can run code for virtually any type of application or backend service - all with zero administration, making it a cost-effective solution for updating resources in AWS.

### Task 3.1: Create your lambda function

41. On the navigation bar, choose **Services** , and choose **Lambda**.

42. Choose **Create Function**

43. Create function by selecting **Author from scratch**.

44. Under **Basic Information**, configure the following settings:

- **Function name:** `updateASG`
- **Runtime:** `Python 3.8`

Just as you can assign roles to Amazon EC2 instances for their installed applications to access AWS resources, AWS Lambda allows you to do the same. Your security team has already created a role that has the proper permissions that your function needs to update Auto Scaling Groups.



Expand  **Change default execution role.**

- **Execution role:** Choose **Use an existing role**.
- **Existing role:** Select `LabLambdaRole`

45. Choose **Create Function**

AWS Lambda gives you many options to upload your code. Since the file is not that large, you will just copy and paste it into the console. The inline editor is a great way to handle small functions and make quick edits.

46. Scroll down to the **Function code** section and replace the text inside of `lambda_function` with the following code:

 Instead of trying to highlight and copy this entire block of code, make sure to select the  icon on the code block when you hover over the top right of the code block. That will copy the entire contents for you to your clipboard.

```
##Lambda function expects one value passed in the event, targetASG
```

```

##Example, {"targetASG": "ASG-NAME"}

from __future__ import print_function
from dateutil import parser

import json
import boto3
import base64

print('Loading function')
autoscaling = boto3.client('autoscaling')
ec2 = boto3.client('ec2')

#function to sort images by CreationDate
def newest_image(list_of_images):
    latest = None

    for image in list_of_images:
        if not latest:
            latest = image
            continue

        if parser.parse(image['CreationDate']) >
parser.parse(latest['CreationDate']):
            latest = image

    return latest

def lambda_handler(event, context):
    print("Received event: " + json.dumps(event, indent=2))

    #get Auto Scaling group by CloudWatch Event data
    response =
autoscaling.describe_auto_scaling_groups(AutoScalingGroupNames=
[event['targetASG']]).get('AutoScalingGroups')[0]

    #make sure that you found an ASG by the name passed in the
event
    if not response:
        return 'No such Auto Scaling Group'

    #get the LC from the ASG
    LC_name = response['LaunchConfigurationName']

    #pull the Launch Configuration
    response =
autoscaling.describe_launch_configurations(LaunchConfigurationNames=
[LC_name]).get('LaunchConfigurations')[0]

    #save the Launch Configuration in a variable to be used when
copying the LC
    LC = response
    #save the ami in a variable
    LC_ami = response['ImageId']

    #get the information about the AMI currently being used in the
LC
    response = ec2.describe_images(ImageIds=[LC_ami],Owners=
['self']).get('Images')[0]
    #Loop through the tags on the AMI and set the serverName
variable on the one with
    # Name as the tag Key
    for tag in response['Tags']:
        if tag['Key'] == 'Name':
            serverName = tag['Value']

    #Make sure that the serverName variable was created
    if not serverName:
        return 'Image is not tagged properly'

    #get a list of AMIs that you own that has the same Name tag
    response = ec2.describe_images(Owners=['self'],Filters=
[{'Name':'tag:Name','Values':[serverName]})].get('Images')

```

```

##Lambda function expects one value passed in the event,
targetASG ##Example, {"targetASG": "ASG-NAME"} from
__future__ import print_function from dateutil import parser
import json import boto3 import base64 print('Loading
function') autoscaling = boto3.client('autoscaling') ec2 =
boto3.client('ec2') #function to sort images by CreationDate
def newest_image(list_of_images): latest = None for image in
list_of_images: if not latest: latest = image continue if
parser.parse(image['CreationDate']) >
parser.parse(latest['CreationDate']): latest = image return
latest def lambda_handler(event, context): print("Received
event: " + json.dumps(event, indent=2)) #get Auto Scaling
group by CloudWatch Event data response =
autoscaling.describe_auto_scaling_groups(AutoScalingGrou
pNames=[event['targetASG']]).get('AutoScalingGroups')[0]
#make sure that you found an ASG by the name passed in
the event if not response: return 'No such Auto Scaling
Group' #get the LC from the ASG LC_name =
response['LaunchConfigurationName'] #pull the Launch
Configuration response =
autoscaling.describe_launch_configurations(LaunchConfigur
ationNames=[LC_name]).get('LaunchConfigurations')[0]
#save the Launch Configuration in a variable to be used
when copying the LC LC = response #save the ami in a
variable LC_ami = response['ImageId'] #get the information
about the AMI currently being used in the LC response =
ec2.describe_images(ImageIds=[LC_ami],Owners=['self']).get
('Images')[0] #Loop through the tags on the AMI and set the
serverName variable on the one with # Name as the tag Key
for tag in response['Tags']: if tag['Key'] == 'Name':
serverName = tag['Value'] #Make sure that the serverName
variable was created if not serverName: return 'Image is not
tagged properly' #get a list of AMIs that you own that has the
same Name tag response =
ec2.describe_images(Owners=['self'],Filters=[{'Name':'tag:Na
me','Values':[serverName]})].get('Images') #find the latest
AMIid with a help from the newest_image function latestAMI
= newest_image(response['ImageId']) #check to see if you're
already using the latest AMI, exit if you are if LC_ami ==
latestAMI: print('No newer AMIs are available') return 'No
newer AMIs are available' #launch configurations for this
function should be named with the following #pattern <LC
Name>V<version number>. This line will increment the
version number #creating a new unique name,
new_LC_name = "V".join(LC_name.split("V")
[0],str(int(LC_name.split("V")[-1])+1)) #check to see if the LC
is using Spot Pricing spot = False try: if LC['SpotPrice']: spot
= True print('Spot Price found!') except KeyError: print('On
Demand Pricing') #create new LC from values of the previous
one #if the current LC uses Spot pricing, pass that to the new
one #else, create a new one that does not use spot pricing if
spot: response =
autoscaling.create_launch_configuration( ImageId=latestAMI,
InstanceType=LC['InstanceType'],
LaunchConfigurationName=new_LC_name,
UserData=base64.b64decode(LC['UserData']),
SecurityGroups=LC['SecurityGroups'],
SpotPrice=LC['SpotPrice']) else: response =
autoscaling.create_launch_configuration( ImageId=latestAMI,
InstanceType=LC['InstanceType'],
LaunchConfigurationName=new_LC_name,
UserData=base64.b64decode(LC['UserData']),
SecurityGroups=LC['SecurityGroups']) #update ASG with

```



```

#find the latest AMIID with a help from the newest_image
function
    latestAMI = newest_image(response)['ImageId']

    #check to see if you're already using the latest AMI, exit if
    you are
    if LC_ami == latestAMI:
        print('No newer AMIs are available')
        return 'No newer AMIs are available'

    #launch configurations for this function should be named with
    the following
    #pattern <LC Name>V<version number>. This line will increment
    the version number
    #creating a new unique name.
    new_LC_name = "V".join([LC_name.split('V')
[0],str(int(LC_name.split('V')[1])+1)])

    #check to see if the LC is using Spot Pricing
    spot = False
    try:
        if LC["SpotPrice"]:
            spot = True
            print('Spot Price found!')
    except KeyError:
        print('On Demand Pricing')

    #create new LC from values of the previous one
    #if the current LC uses Spot pricing, pass that to the new one
    #else, create a new one that does not use spot pricing
    if spot:
        response = autoscaling.create_launch_configuration(
            ImageId=latestAMI,
            InstanceType=LC['InstanceType'],
            LaunchConfigurationName=new_LC_name,
            UserData=base64.b64decode(LC['UserData']),
            SecurityGroups=LC['SecurityGroups'],
            SpotPrice=LC["SpotPrice"]
        )
    else:
        response = autoscaling.create_launch_configuration(
            ImageId=latestAMI,
            InstanceType=LC['InstanceType'],
            LaunchConfigurationName=new_LC_name,
            UserData=base64.b64decode(LC['UserData']),
            SecurityGroups=LC['SecurityGroups']
        )

    #update ASG with new LC
    response = autoscaling.update_auto_scaling_group(
        AutoScalingGroupName='Web-App-Test-ASG',
        LaunchConfigurationName=new_LC_name
    ).get('ResponseMetadata')

    #if the update was successful return a success.
    if response['HTTPStatusCode']==200:
        print('Successfully Updated Auto Scaling group: ' +
str(event['targetASG']))
        return 'Successfully Updated Auto Scaling group: ' +
str(event['targetASG'])

```

```

new LC response =
autoscaling.update_auto_scaling_group( AutoScalingGroupN
ame='Web-App-Test-ASG',
LaunchConfigurationName=new_LC_name ).get('Response
Metadata') #if the update was successful return a success. if
response['HTTPStatusCode']==200: print('Successfully
Updated Auto Scaling group: ' + str(event['targetASG']))
return 'Successfully Updated Auto Scaling group: ' +
str(event['targetASG'])

```

47. Choose **Deploy** to publish changes.

## Task 3.2: Create a test event for your function

Now that you have your Lambda function created, you will test it to make sure there are no errors while copying the code. The function, is made to



accept one value in the Event that invokes it, the Auto Scaling Group's name. This allows one function to handle multiple Auto Scaling groups, if it needs to. First, set up your test event, passing the expected variable.

48. Choose **Test**

49. For **Event name**, enter `TestUpdateASG`

50. Replace the event with the following code:

```
{
  "targetASG": "Web-App-Test-ASG"
}
```

51. Choose **Create**

You can test the function now.

52. Choose **Test**, again, with **TestUpdateASG** selected to the left of the Test button.

The Execution result should show that it succeeded, and if you expand **Details** you should see that "No newer AMIs are available" was returned from the function. You will create another On-demand backup so that you can fully test the function.

⚠ if your test failed, go back and verify that the code copied correctly, and check to make sure that you have the correct execution role selected on the Permissions tab.

### Task 3.3: Create another on-demand backup to test your function

Most of this task is repeating what you did before but with one exception. Since you created an on-demand backup of an already protected resource, you will not need to specify the instance or add the tags. You will be specifying a shorter retention period for this snapshot to make sure you are not paying for anymore snapshot storage than what you need.

53. On the navigation bar, choose **Services**, and choose **AWS Backup**.

54. Choose **Protected Resources** on the left navigation pane.

55. Choose the Resource ID of you one instance that you have protected.

56. Choose **Create on-demand backup**

57. Configure the following settings:

- Under **Lifecycle**, for **Expire**, select **Days after creation** and enter `1`
- For **Backup vault**, select `Lab-Vault`
- In **IAM Role** section, choose **Choose an IAM role:** and select `LabBackupServiceRole`

58. Choose **Create on-demand backup**

The status will initially say **Created**, once it changes to **Running** continue to the next step. This could take up to 10 minutes and you may have to refresh the box to see the change.

59. In the AWS Backup Console, on the **Services** menu, select **EC2**.

60. In the left navigation pane, under **Images**, choose **AMIs**.

61. Verify that the latest AMI from your on-demand backup is available. If it is still pending then wait. It should take 2-3 minutes to complete.

62. In the AWS Console, on the **Services** menu, select **Lambda**.

63. Choose the Function name **updateASG**.

64. Choose **Test** in the upper right-hand corner of your screen.

The Execution result should show that it succeeded again but this time when you expand **Details** the results returned should say, "Successfully Updated Auto Scaling group: Web-App-Test-ASG". If you would like, you can go back to the EC2 service and verify that a new Launch Configuration created is using the new AMI and the Auto Scaling group updated using the new Launch Configuration.

## Task 4: Create an AWS CloudWatch event to invoke your AWS Lambda function daily

The lambda function appears to be working correctly, but you will want to avoid going in and manually invoke it every morning. The Final Task is to automate the invocation of your AWS Lambda Function. You will schedule the lambda function to update the Launch Configuration every day before it scales up your test environment. Since you set the Auto Scaling group to scale up your test environment at 9 AM EST Monday thru Friday above, you will set the AWS Lambda function to run at 8am on those same days.

Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources. Using simple rules that you can quickly set up, you can match events and route them to one or more target functions or streams. CloudWatch Events becomes aware of operational changes as they occur. In addition, it responds to these operational changes and takes corrective action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information.

You can also use CloudWatch Events to schedule automated actions that self-trigger at certain times using Cron or rate expressions, as you will be doing here.

65. In the AWS Console, on the **Services** menu, select **CloudWatch**.
66. In the left navigation pane, under **Events**, choose **Rules**.
67. Choose **Create rule**
68. Choose **Schedule** under **Event Source** section.
69. Choose **Cron expression** and enter `0 13 ? * MON-FRI *`
70. On the right choose **+ Add target\***
71. **Lambda Function** should already be selected in the dropdown list, if not then select it.
72. For **Function**, select `updateASG`
73. Expand **Configure input** by choosing ▶
- Select **Constant (JSON text)** and enter `{ "targetASG": "Web-App-Test-ASG" }`
74. Choose **Configure details**
75. **Name** `update-Web-App-Test-Environment`
76. Choose **Create rule**

This Event will invoke your AWS Lambda function every day at 8:00 AM EST and update the Launch Configuration for the Web-App-Test-ASG Auto Scaling group if there is a new AMI available. If you need to update multiple Auto Scaling groups you can create more Rules to invoke the lambda function passing the Auto Scaling group name to the function.

⚠ Remember, the default Service Quota for **Launch configurations per region** is 200. You can view that quota or Request a quota increase by selecting **Services**, going to the **Service Quotas** Service, **AWS services**, on the left navigation pane, and selecting **Amazon EC2 Auto Scaling**. Instead of requesting a service quota increase though, it would be better to clean out old Launch Configurations if you do not need them. You could set up an AWS Lambda function that cleans up launch configurations when backups expire, but that is out of scope for this lab.

## Optional Challenge: Test the CloudWatch rule

You have everything you need to figure out how to test this CloudWatch rule on your own.

### Clues:

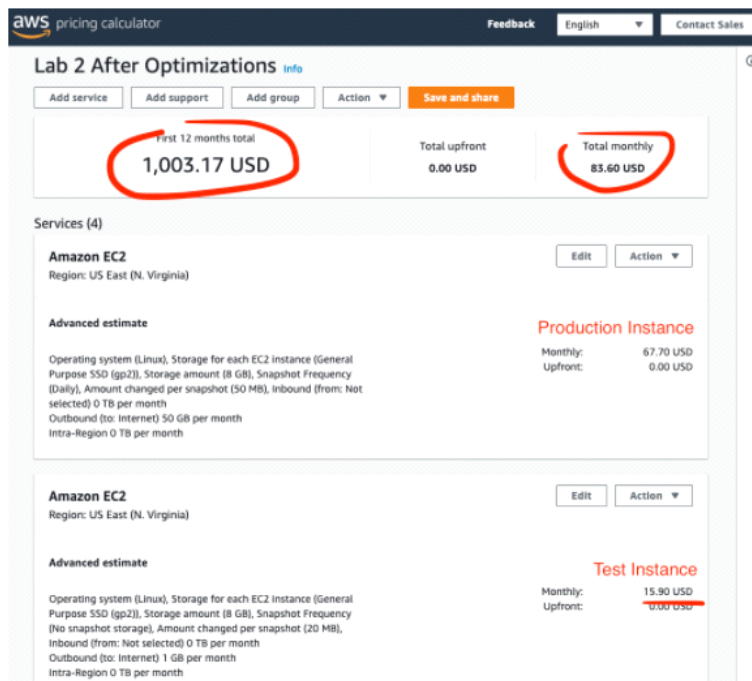
- Change the rule to trigger at a 1 minute fixed rate.
- Create another backup.

- Confirm that the Auto Scaling group was updated.

## Summary

In this lab you saved costs by only running resources when they are needed. Utilizing services that have generous perpetual free tiers, you were also able to fix the issues that you were having with your test environment's configuration drifting from the production environment.

Below is a screenshot of the AWS pricing calculator showing the cost of the environment after implementing the optimizations in this lab. The prices are as of 5/12/2020. You can select the link below see an estimate online with current prices.



<https://calculator.aws/#/estimate?id=b23280f6192a8300a480ddc4464b454d95002c78>

⚠ The lab uses t2.micro instances to illustrate a point while keeping costs down for the lab environments. The calculator uses more real world examples of c5.large. The estimates provided use instance types which are more realistic for a customer deployment, yet the percentage of cost savings between the start and ending estimates remain the same.


## End Lab

Follow these steps to close the console, end your lab, and evaluate the experience.

77. Return to the AWS Management Console.

78. On the navigation bar, choose **awsstudent@<AccountNumber>**, and then choose **Sign Out**.

79. Choose 

80. Choose 

81. (Optional):

- Select the applicable number of stars ☆
- Type a comment
- Choose **Submit**
  - 1 star = Very dissatisfied
  - 2 stars = Dissatisfied
  - 3 stars = Neutral
  - 4 stars = Satisfied
  - 5 stars = Very satisfied

You may close the window if you don't want to provide feedback.

For more information about AWS Training and Certification, see <http://aws.amazon.com/training/>.

*Your feedback is welcome and appreciated.*

If you would like to share any feedback, suggestions, or corrections, please provide the details in our [AWS Training and Certification Contact Form](#).

-----

```
{ "targetASG": "Web-App-Test-ASG" }
```

```
{"targetASG": "Web-App-Test-ASG"}
```