Show Submission Password

Horizontal Scaling and Advanced Resource Scaling Horizontal Scaling and Advanced Resource Scaling

Introduction

Introduction to Resource Scaling

Information

Learning Objectives

This project will encompass the following learning objectives:

1. Design solutions and invoke cloud APIs to programmatically provision and deprovision cloud resources for a dynamic load.
2. Configure and deploy an Elastic Load Balancer and an Auto Scaling Group on AWS.
3. Develop solutions that monitor cloud resource metrics to manage cloud resources with the ability to deal with resource failure.
4. Analyze a workload pattern and develop elasticity policies to maintain the Quality of Service (QoS) of a web service.
5. Account for cost as a constraint when provisioning cloud resources and analyze the performance tradeoffs due to budget restrictions.
6. Orchestrate infrastructure on the cloud using Terraform as part of the deployment process.

Show Submission Password

Danger

Intense Project Warning!

This project requires the understanding of cloud APIs in AWS as well as consistent development and a systematic testing effort where each test will take 24 minutes. This project will require a lot of time to complete successfully and hence we recommend starting as early as possible to be successful.

Danger

Accessing Course Resources

Before you provision any resources, make sure that you have updated your 12-digit AWS account ID in your profile (https://theproject.zone/profile/) of theproject.zone. Failing to do so will get your login blocked when using the AMI image provided by us.

Warning: Make sure that your instance limits (https://piazza.com/class/k562fiaob2hlh?cid=67) for the following types of instances are at least 20 for the following instance type: `m5.large` . You can check your EC2 service limits on the console (https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#Limits:). If you are facing an issue with the vCPUs limit, please **contact Amazon** to increase the limit (https://console.aws.amazon.com/support/v1#/case/create?issueType=service-limit-increase&limitType=service-code-ec2-instances) instead of posting this issue on Piazza. Amazon customer service will handle your request in a timely fashion (typically in a few days).

Information

General Details

The following table contains some general information about this project:

Prerequisites	Java 8 or Python 3

Primers	Git Best Practices; Intro to Maven and Checkstyle (for Java user); Jupyter Notebook (for Python user); Amazon Web Services APIs; Infrastructure as Code
Applicable Languages	We recommend that you use *Java 8 (JDK 1.8 preferred) and Python 3* so that the teaching staff can support you. Otherwise, you may use any language that your AWS instance supports. However, the teaching staff may not be able to help you resolve problems in other languages. Note: If you use Maven, the *Maven central repository* is the only remote repository you are allowed to use.
Applicable Cloud Platform	Amazon Web Service (AWS)

Show Submission Password

Danger

Project Grading Penalties

Violation	**Penalty of the project grade**
Spending more than $20 for this project phase on AWS	-10%
Spending more than $40 for this project phase on AWS	-100%
Failing to tag all your resources in either part (EC2 instances, ELB, ASG) for this project with the tag: key= `Project` , value= `2.1` (AWS only)	-10%
Submitting your AWS/Andrew credentials in your code for grading	-100%
Using instances other than t3.micro (testing only) or m5.large for Horizontal scaling on AWS	-100%
Using instances other than t3.micro (testing only), m5.large for Autoscaling on AWS	-100%
Submitting only executables (`.jar` , `.pyc` , etc.) instead of human-readable code (`.py` , `.java` , `.sh` , etc.)	-100%
Attempting to hack/tamper the autograder in any way	-200%
Cheating, plagiarism or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus)	-200% & potential dismissal

Warning

Show Submission Password

Resource Tagging

For this project, assign the tags with `Key: Project` and `Value: 2.1` for all AWS resources.

In the previous project, you were introduced to using the cloud to perform big data analytics. You used EMR to launch a cluster of machines that processed a large dataset in parallel using only a few lines of code. Unfortunately, PaaS services like AWS EMR, Azure HDInsight, or GCP Dataproc can be expensive and the internals are not transparent to you. Also, although such services support a fair number of frameworks, it is still a limited and inflexible set, which are mostly geared towards the needs of batch processing and analytics. You might need to deploy another framework or your own web service on cloud resources. To enable you to do this, we will now shift gears to look at ways in which we can deploy performant web services that respond to dynamic requests on the cloud.

If you recall in Project 1, we intentionally required you to use a micro instance, specifically `t3.micro`, which had limited hardware resources. Many of you found that code that would run fairly fast on your own machines would take much more time to run on the `t3.micro` instance, or would sometimes even run out of memory if the code was not efficient. As a result, many of you were forced to optimize your code given the resource constraints of the instance.

As you can imagine, there is a limit to how much you can optimize the code. For certain workloads, you simply must be able to assign the appropriate hardware resources in order to complete processing the workload efficiently. In cloud computing, the process of adjusting the number of resources assigned to a particular workload/task/service is known as **scaling**.

Types of Scaling

For the purposes of this course, scaling of resources can be categorized broadly into two main parts:

Vertical Scaling

The simplest technique used to scale an application is vertical scaling, which involves changing the capacity of the resources in the system. For instance, if the resource being used is a virtual machine, vertical scaling could involve changing the number of cores, amount of memory, or CPU speed used by the VM. This typically requires provisioning a new VM with a different amount of resources. Vertical scaling requires the migration of the application or service to another machine. Recall that Vertical Scaling was already explored in Project 0, where you benchmarked the performance of various types of instances on AWS, GCP and Azure.

Horizontal Scaling

Horizontal scaling involves adding or removing resources, such as VMs, from/to the system respectively. For example, if a web service is currently utilizing 5 identical VMs and the load continues to grow. Using horizontal scaling, we will increase the number of identical VMs to 6 to deal with the increased load. Horizontal scaling is more complicated than vertical scaling because you now have a task distribution and assignment problem, for example, to which VM should you send a new incoming request? In the simplest case, resources can be scaled using replicas which can be used interchangeably. Horizontal scaling can also be performed by partitioning the problem domain into smaller "shards", and making each individual resource responsible for a single partition (or shard). The remaining parts of this project all deal with horizontal scaling. In this project module, we will only focus on simple replication of resources. In the days before cloud computing gained broad usage, scaling of resources was a complex task that involved management, engineering and logistics along with months of planning and execution. However, cloud services allow elasticity, where cloud users can rapidly and **dynamically** scale resources based on a number of programmable factors. The theme of this project is elasticity; you will gain first-hand experience with dynamic scaling and elasticity.

Show Submission Password

This project introduces you to the methods which allow you to write programs that manipulate and control resources on a public cloud. Though this project primarily relies on stringing together API calls in the right order, it is important for your code to be reliable and fault-tolerant. Making cloud API calls in a networked program is not as easy as sequential programming. Each API call is merely a request to a cloud provider, which can return either success or failure; you must account for all possibilities. Defensive programming is crucial this week.

At the end of this module, you will learn how to programmatically manage AWS resources and explore a web-service deployment scenario involving infrastructure that is provisioned dynamically using your own code, by an autoscaling platform provided by AWS and finally using a Terraform script.

Scenario

You are a competitive job seeker who is extremely interested in working for clandestine organizations that may or may not have Orwellian goals. Recently, you received the following letter offering you a chance to join one of the top organizations in this domain, the Massive Surveillance Bureau (MSB).

Massive Surveillance Bureau
Quis Custodiet Ipsos Custodes

Show Submission Password

First-round interview invitation letter

Dear Student,

We are pleased to confirm that you have been shortlisted for the first round of selection for the System Architect Program (fast track) at the Massive Surveillance Bureau (MSB).

In this round, you need to pass our Horizontal Scaling and Autoscaling Test to show that you have a clear idea of how to elastically provision your resources to maintain your Quality of Service. You should complete Horizontal Scaling tests on AWS. Then you need to complete an Autoscaling Test on AWS.

These tasks are time intensive! Good luck!

Warmest regards,

Harry Q. Bovik

Human Resources
Massive Surveillance Bureau
Quis Custodiet Ipsos Custodes

Information

Getting Ready

In Project 0 you gained hands-on experience with a form of scaling called **vertical scaling**, by simply changing the type of an instance from one with a smaller number of CPU cores, memory and network bandwidth to another with more hardware resources.

In contrast, **horizontal scaling** involves adding or removing identical virtual machines assigned to a particular task/workload/service. An example on AWS would be ***scaling out*** from a single `m5.large` virtual machine to several `m5.large` virtual machines. Conversely, you can also ***scale in*** if you reduce the number of resources assigned to a particular task/workload/service.

In this interview process (project module), we will use two types of instances, a **Load Generator (LG)**, which generates requests, and **Web Service (WS)** instances, which handle and respond to requests produced by the load generator.

The MSB requires you to write code and build a system that is capable of handling the large volume of requests generated by the Load Generator with minimal cost. To impress the MSB, you will have to write code that launches these instances, initiates a special test from the Load Generator to send requests to the web service instances, and measures the requests per second from the web service instances. Your program will have to scale the number of web service instances until your entire system is able to handle a specified target of requests per second (RPS).

Show Submission Password

You will be scored on your ability to scale out and achieve the target RPS within the allotted time.

Note: You may notice that you are not able to access the Load Generator or Web Service instances via SSH. This is the expected behavior. The project does not require you to perform any operations on the instances.

Note: Please only use the official APIs/SDKs provided by the cloud providers. Do not use third-party or cloud agnostic libraries like libcloud. They defeat the learning objectives of the assignment as you won't be able to effectively learn the differences between the APIs and their complexities.

Information

Starter Code

We provide you with a starter code in Java and Python, which you will find on the instance launched using the student workspace VM image specified below.

*** Note: The starter code is only for your reference. You are welcome to start from scratch or use snippets of it.***

The starter code for `task1` and `task2` contains the basic dependencies you will need and stubs to interact with the APIs on the load generator and the web service.

The starter code also contains the terraform file template you will need.

Put all your configuration parameters for `task1` and `task2` in the json files provided.

AWS Horizontal Scaling

Information

The MSB wants to ensure that the candidates it selects have good skills in AWS. To move to the next phase of the selection process for the MSB, you will need to perform the Horizontal Scaling experiment on AWS. You can use the primers on the AWS APIs as a reference for this section.

Getting the files

1 - You need to provision a `student-vm` using `t3.micro` instance in the `us-east-1` region using the AWS console with the AMI `ami-0795e993eb27953cf`. You will have to open both port 80 and port 22 for this instance. We suggest that you use the EC2 Fleets or On-demand EC2 instance Terraform template provided in **Project 1**. Please remember to update the tags. If you decide not to use Terraform, note that you need to manually tag the EC2 instances launched by Spot requests.

Show Submission Password

2 - Open the URL `http://[YOUR-EC2-DNS]` in your web browser for the instance. Enter your TPZ username and submission password. Then click the **Launch Project** button of the **Horizontal Scaling and Advanced Resource Scaling** section. Wait for several minutes until the log tells you that the instance is ready. You can then log into the instance using the PEM/PPK file as the user `clouduser`.

4 You can find the starter code for each task under `/home/clouduser/VM_Scaling_Project/`.

Tasks to Complete

The folder to work on in this task is in `/home/clouduser/VM_Scaling_Project/<java/python>/task1` on your EC2 VM launched with the Cloud Computing Project Image

Using the Amazon EC2 API, please write code that does the following:

1 - Launch a `m5.large` load generator using `ami-0120179ee1facd28b` with the correct tags

2 - Launch a `m5.large` web service instance using `ami-0496215388838dae0` with the correct tags

3 - Before you start, you must authenticate with the load generator. Your code must submit your submission password and username to the load generator in the following format:

```
   http://[load-generator-dns]/password?passwd=[your-submission-password]&userna
me=[your-username]
```

4 - Next, submit the web service VM's DNS name to the load generator to start the test:

```
   http://[load-generator-dns]/test/horizontal?dns=[web-service-instance-dns]
```

You will be provided with a test id if the launch of the test was successful. The load generator should start sending a large amount of traffic to your web service VM. Unfortunately, a single VM is incapable of handling this traffic.

5 - Your code will now need to monitor the test from the log URL and fetch the total RPS of the web service instance(s):

```
   http://[load-generator-dns]/log?name=test.[test-number].log
```

An example of the log is provided below:

Show Submission Password

```
; 2019-01-05T01:02:25+00:00
; Horizontal Scaling Test
; isTestingThroughCode=true
; Test launched. Please check every minute for update.
; Your goal is to achieve rps=50 in 30 min
; Minimal interval of adding instances is 100 sec
[Test]
type=horizontal
testId=12345678901234
testFile=test.12345678901234.log
startTime=2019-01-05T01:02:25+00:00

[Minute 1]
ec2-10-10-10-02.compute-1.amazonaws.com=10.0
[Current rps=10.00]

[Minute 2]
ec2-10-10-10-02.compute-1.amazonaws.com=10.0
[Current rps=10.00]

[Minute 3]
ec2-10-10-10-02.compute-1.amazonaws.com=10.0
[Current rps=10.00]

[Minute 4]
ec2-10-10-10-02.compute-1.amazonaws.com=10.0
ec2-10-10-10-03.compute-1.amazonaws.com=10.0
[Current rps=20.0]
.
.
.

[Minute 20]
ec2-10-10-10-02.compute-1.amazonaws.com=10.0
ec2-10-10-10-03.compute-1.amazonaws.com=10.0
ec2-10-10-10-04.compute-1.amazonaws.com=10.0
ec2-10-10-10-05.compute-1.amazonaws.com=10.0
ec2-10-10-10-06.compute-1.amazonaws.com=10.0
[Current rps=50.00]

[Load Generator]
username=student@andrew.cmu.edu
platform=AWS
instanceId=i-abcdefghijklmnop
instanceType=m5.large
hostname=ec2-10-10-10-01.compute-1.amazonaws.com
passwd=abcdefghijklmnopqrst

[Web Service 0]
username=student@andrew.cmu.edu
platform=AWS
instanceId=i-abcdefghijklmnop2
instanceType=m5.large
hostname=ec2-10-10-10-02.compute-1.amazonaws.com
.
.
.
```

```
    ; MSB is validating...
    [Test End]
    rps=50.00
    pass=true
    endTime=2019-01-05T01:09:29+00:00
```

Show Submission Password

Example 1: Horizontal Scaling Sample Log

6 - Your code should launch more web service instances, one at a time, in order to achieve a cumulative RPS of `50`. By cumulative RPS, we mean the total RPS of all web service instances added together. For example, in the above log file, the cumulative RPS at minute 4 is `20.00`. Once launched and running, new instances can be added to a running test by sending the following request to the load generator:

```
    http://[load-generator-dns]/test/horizontal/add?dns=[web-service-instance-dn
s]
```

7 - Please read the Testing Rules section carefully to pass the test. Once a test is complete, you should be able to visit the submissions page to see your score for this task.

What to Submit

In addition to the above submission, you also need to upload your source code to TPZ using the submitter. Your test will be graded only if you have uploaded code for that test. The submitter will upload your source code to our autograder for check-style grading. Note that your submission must contain a file called **references**, which should contain a list of references you used to complete the project. A sample reference file can be found here (https://s3.amazonaws.com/15619public/webcontent/references). Please do not include any binaries (.jar) or cloud credentials (.pem,access_key.csv) files.

How to Submit

```
    export TPZ_USERNAME=your_tpz_username
    export TPZ_PASSWORD=your_tpz_pwd
    ./task1_code_submitter
```

Information

Please note that:

1 - All instances must be tagged. The instance tagging for all those steps above must be done in your code, and not manually through the AWS web console. Only the `student-vm` instance is allowed to be tagged from the console.

2 - Make sure that you create a security group with inbound `port 80` and all outbound ports open with code for your load generator and all your web service instances.

3 - You are required to develop and run the program locally to complete the scaling test and you will only use the `student-vm` to submit your source code to TPZ.

Testing Rules

1 - To complete the test, you have a maximum of `30` minutes to reach your target RPS of `50` .

Show Submission Password

2 - Please make sure you start with a blank slate. All your instances must be launched by your code. All web service instances (except for the first one) should be launched **after** the test starts. The autograder checks for this.

3 - Make sure your code processes the current requests per second and then decide whether to launch another instance.

- **Do not hardcode** the number of instances into the code.

4 - There should be a time window of at least `100 seconds` between any pair of web service instance VMs' launch time. That is, if your program just launched a web service VM, your program should wait at least `100 seconds` before it launches another one. This window is known as the **cooldown period**, and the cooldown period helps to ensure that your scaling does not launch or terminate additional instances before the previous scaling activity takes effect.

- **Do not hardcode** a fixed delay between launch operations. This means that you can not explicitly use time.sleep(100) or any similar operation to launch a new instance every 100 seconds. However, you are allowed to record the last time you added an instance and check that 100s have elapsed since that time before adding a new one.
- However, you may use sleep for <= 1 seconds while retrying a request. Please refer to hint 16 for examples of allowed and disallowed uses of sleep.
- The code snippet below demonstrates the correct usages of `sleep`

```
request_success=false
while(!request_success){    //  Usage 1
        try{
                make_http_request()
                request_success = true
        }
        catch (Exception){
                Thread.sleep(<=1 second)
        }
}

while (test is not complete) {    // Usage 2
        Thread.sleep(<=1 second); // Very small sleep time to save clock c
ycles
        if (current_time - previous_launch_time > 100 seconds){
                //check or do something
        }
        current_time = update_current_time()
}
```

The following usage of sleep is **NOT ALLOWED**:

```
if (test is not complete){
        Thread.sleep(100 seconds)
        launchInstance()
}
```

6 - In order to receive score for the test on TPZ, your program should be `fully automated` and `fault-tolerant`. That means, you run your program and the program will launch an LG VM and WS VM, fill in your submission password and username, start the test, add more VMs if needed, and finally exit when the test is done.

Show Submission Password

7 - You can terminate all the web service instances (not necessarily through code).

Hints

1 - Try a dry run without code (using only the browser), if you want to get a feel for the process. Only run the test once you are sure you understand all the different pieces involved. You can also use `t3.micro` instances for initial testing to save budget, but with `t3.micro` you will not be able to reach the desired target for the final submission.

2 - If you are unfamiliar with what requests you need to make and what output to expect, view the process on your browser. You can use developer tools in Chrome and Firefox to analyze the requests that are being sent when you perform some operation on the UI. You can also see the responses received for each and every request.

3 - Launching all instances within the same subnet should yield better performance than having them in different subnets. You do not need to create your own private subnet and VPCs. You can obtain the subnet IDs from the availability zones for usage. You can look up the VPC ID and the Subnet IDs using the AWS console. Make sure that all the IDs are in the same availability zone.

4 - You can modify the sample code provided in the AWS API primer to launch a virtual machine.

5 - You need to use the DNS rather than IP address of the virtual machine in this project.

6 - The Horizontal Scaling Test will end once the sum of the RPS of your web service instance VMs reaches the target threshold. Your program may want to make use of the log to check if the test finished.

7 - The test log files are in the ini format. You might want to consider using `ConfigParser` library in python and `ini4j` in Java to parse them. When using `ConfigParser`, make sure you do `configparser.ConfigParser(strict=False)` since there are sections with same values in the log file.

8 - During the test, you may find it interesting to monitor the instance graphs on the EC2 Console. You can optionally create a custom Dashboard to monitor multiple parameters. `Note: Custom dashboards are chargeable on a monthly basis.`

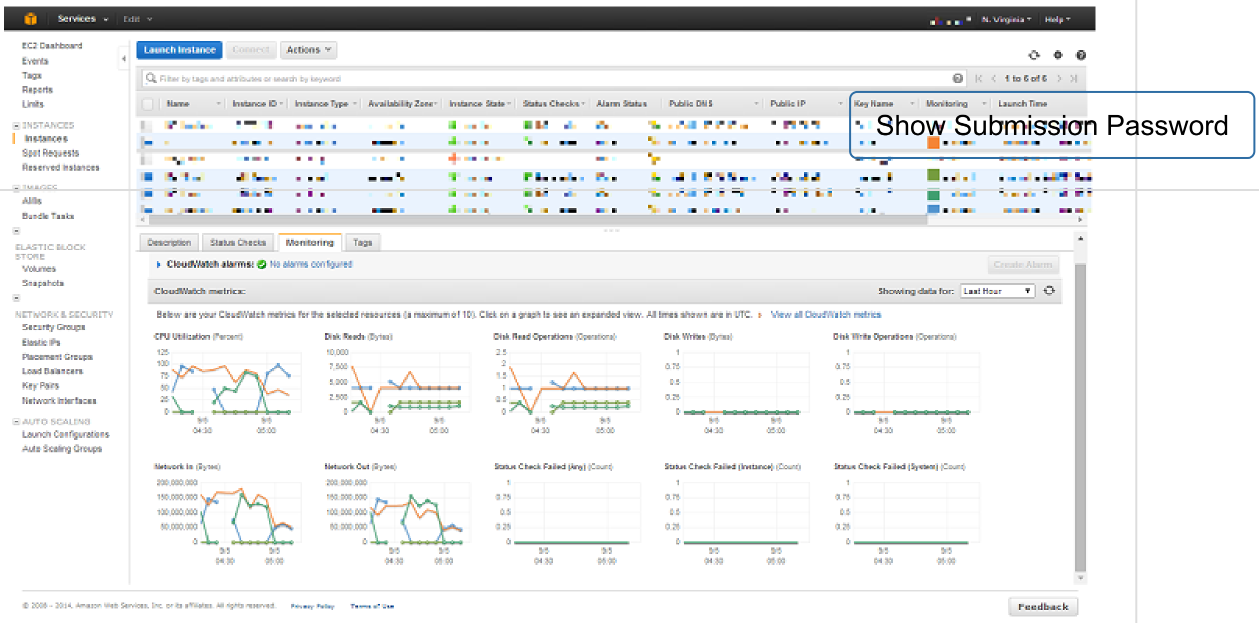


Figure 2: CloudWatch on the EC2 Console

9 - Try to make your code modular and reusable. This can help greatly in future tasks.

10 - **Do not store your Username/Submission password in the code. Read it from a file or environment variables.**

11 If your grading style check fails due to the presence of unnecessary files in your tarball; you can edit the `.fdignore` file in each task folder to include the files that need to be excluded.

12 - You might might need to make an AWS API call to get the latest state once your resource is created completely. For example, when you create an EC2 instance through the API, the EC2 response object does not have an IPv4 address, but it is assigned one upon successful creation.

13 - You will find the **Waiters** class in the AWS SDK for Java and Python particularly useful in waiting for a resource to transition into the desired state, which is a very common task when you're working with services that have a lead time for creating resources (Amazon EC2).

14 - **/test/horizontal?dns=[]** This endpoint is used to start a horizontal scaling test with an initial web service instance. If the test has started successfully, this API returns 200, the test ID and additional test information. If there already is a test running or your web service is not running as expected, this API returns 400 and some error information. This endpoint CANNOT be used to ADD web-service instances to a test that is already running; to add instances, please see the endpoint in the next bullet point.

15 - **/test/horizontal/add?dns=[]** This endpoint is used to add web services instances to an already running test. If the instance has been added successfully, this API returns 200. If your web service is not working as expected, this API returns 400.

16 - **http://[load-generator-dns]/test/kill** This endpoint is used to terminate the test before 24 minutes.

AWS Autoscaling

Information

Show Submission Password

Getting Ready

In this project, you have a load generator and several web service instances trying to ingest data from the load generator. When adjusting the number of Web Service instances, every time you add an instance, you need to inform the load generator, which must perform some computations to begin to distribute the traffic evenly among the web service instances. Here, the load generator distributes the traffic equally among all target hosts. However, what would happen if you wanted to scale down? Or if an instance failed? How would you monitor if an instance failed? These are serious issues that need to be addressed.

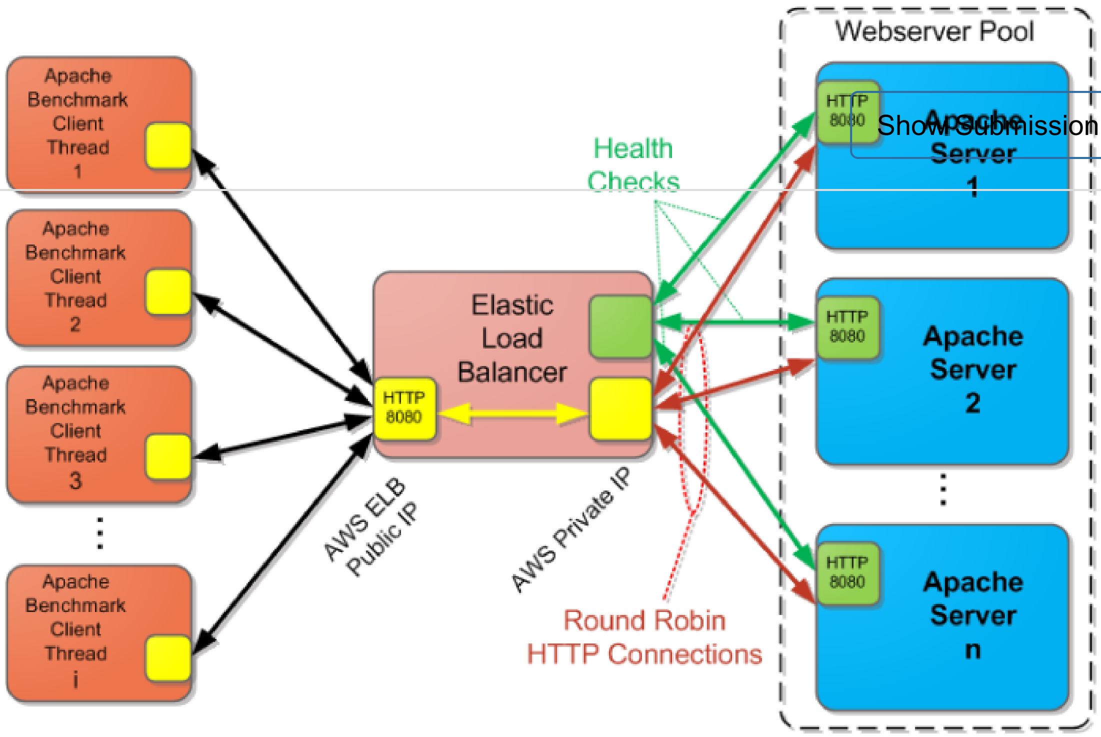
To be able to address these shortcomings of effectively distributing traffic, AWS has a service known as **Elastic Load Balancing** that automatically distributes traffic to connected instances and also handles instance failures. Elastic Load Balancing can also tie in an **Autoscaling Group** which can dynamically shrink or grow a set of instances based on some scaling policies that you can define. We will learn more about these services in this section.

Introduction

Quality of service (QoS) and cost are two very important factors when deploying a web service. One of the sure ways to lose users to the competition is to provide a web service that does not meet acceptable QoS guarantees. Performance, availability, reliability and security are some dimensions of QoS. One can always provide high QoS by over-provisioning resources in order to account for most variability in the load or failure scenarios, however the ability to compete on the cost of the service will become an issue. The focus of this task is to provide you with hands-on experience with balancing QoS and cost.

Elastic Load Balancer

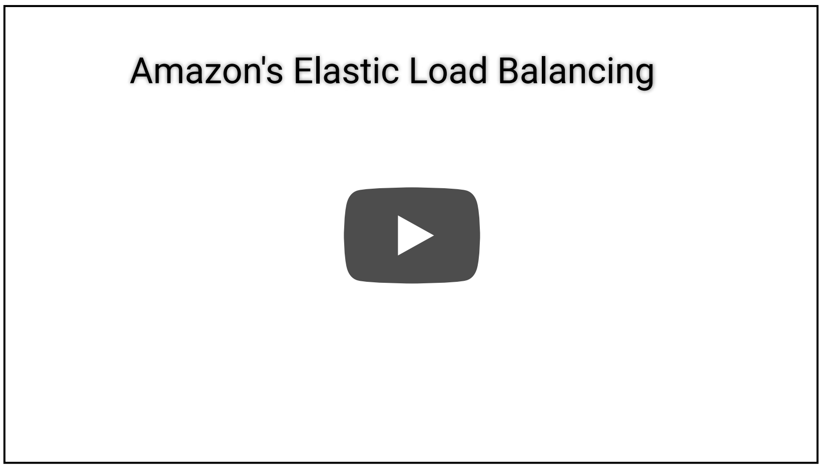
The Elastic Load Balancer (ELB) acts as a network router that will forward incoming requests to multiple resources (e.g., EC2 Instances as in this project) in a round-robin fashion. In this project, we will use an **Application Load Balancer**. The group of instances serving requests behind a given Application Load Balancer is called a **target group**. Instances can be added/removed to/from a target group manually through the web console, programmatically through an API, or dynamically with an Auto Scaling Group. ELB can also perform a Health Check to check if the instance is responsive (if not, it will stop sending requests to it). An ELB basically acts as a front door of your web service or system. When a new instance is added to the target group; a minimum number of health checks at a predefined intervals are performed before the ELB starts sending requests to the newly created instance. You can read more about the health check configuration here (https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-healthchecks.html#health-check-configuration)


Apache Benchmark Client Thread 1
Apache Benchmark Client Thread 2
Apache Benchmark Client Thread 3
Apache Benchmark Client Thread i
Elastic Load Balancer
HTTP 8080
AWS ELB Public IP
AWS Private IP
Health Checks
Round Robin HTTP Connections
Webserver Pool
HTTP 8080
Apache Server 1
Apache Server 2
Apache Server n

Show Submission Password

Figure 3: Elastic Load Balancer

The following video covers the basics of ELB:



Video 1: Basics of ELB

You can programmatically interact with an Elastic Load Balancer in many ways, including:

1 - The AWS Command Line Interface (AWS CLI) (https://aws.amazon.com/cli/)

2 - The AWS SDK for Java (https://aws.amazon.com/sdk-for-java/)

3 - boto3 for Python (https://boto3.readthedocs.io/en/latest/)

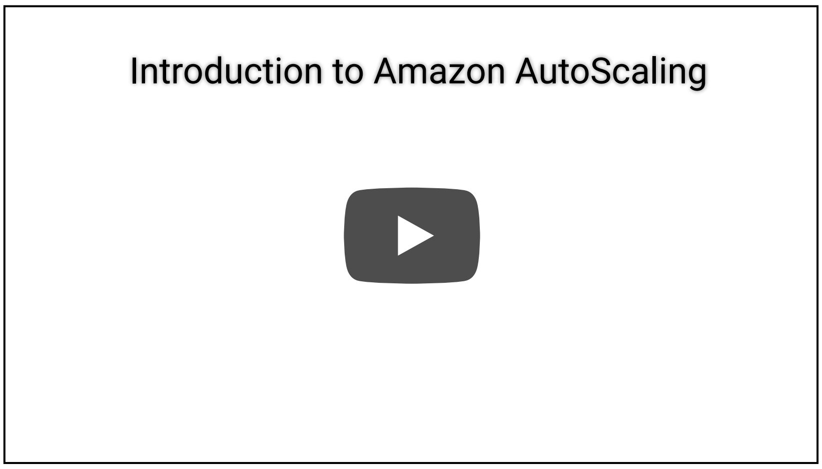
4 - Terraform (https://www.terraform.io/)

We will give you more information about working with these APIs after introducing the next section on Auto Scaling.

Show Submission Password

Amazon Auto Scaling

AWS's Auto Scaling service automatically adds or removes computing resources allocated to an application, by responding to changes in demand. AWS Auto Scaling refers to horizontal scaling, which is the act of increasing or decreasing the compute capacity of an application by changing the number of identical servers assigned to it. This is in contrast to vertical scaling, which is achieved by changing the size of individual servers in response to demand (such as changing m5.medium to m5.large). Amazon Auto Scaling can be accessed through command-line tools or through APIs in the SDK. The following video explains AutoScaling:



Video 2: Auto Scaling

Amazon's Auto Scaling provides the following deployment modes on a group of EC2 instances:

- Maintain a fixed number of running EC2 instances at all times, by performing periodic health checks on instances and automatically launching new instances when one or more are unhealthy;
- Scale instances manually, which will increase or decrease the number of running instances on your behalf;
- Predictable scaling based on a developer-specified schedule (for example, a condition you could specify to increase the servers every Friday at 5 p.m. and to decrease them every Monday at 8 a.m);
- Dynamically scale based on conditions specified by the developer (for example, CPU utilization of your EC2 instances).

Using Elastic Load Balancing and Auto Scaling

An important use case for Auto Scaling is dynamic infrastructure management. In the case of MSB, it should be clear how Auto Scaling can help deal with variable traffic patterns. As the traffic increases and the load on the servers increases, the number of servers can be increased dynamically. Similarly, as the traffic decreases and the load on the servers decreases, the number of servers can be decreased dynamically. In that sense, we can program Auto Scaling to respond to changes in CPU load (or other metrics, such as response time) of provisioned servers.

Show Submission Password

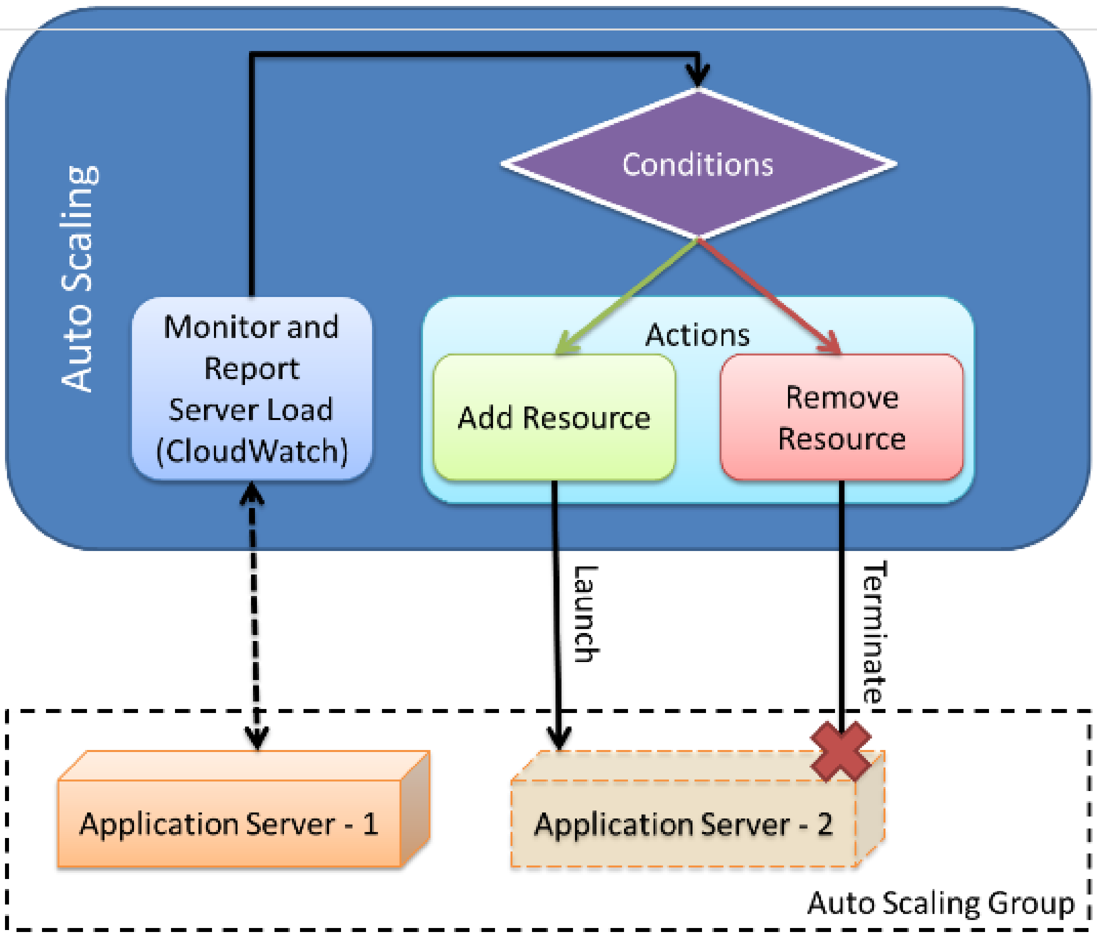


Figure 4: Auto Scaling Architecture in Amazon EC2

AWS Auto Scaling Internals

Using Amazon's Auto Scaling in a dynamic fashion requires, at minimum, the following:

- An **Auto Scaling Group (ASG)** should be defined with a minimum, maximum and desired number of instances defined during the group creation. ASG is also associated with an Elastic Load Balancer and a corresponding Target Group. More information can be found here (http://docs.aws.amazon.com/autoscaling/latest/userguide/GettingStartedTutorial.html) on creating auto-scaling groups.
- A **Launch Configuration** template should be defined, which includes the AMI ID, instance types, key pairs and security group information, among others. Auto Scaling is meant to scale automatically based on application demands; i.e. the instance should be configured to automatically start the required application services and work seamlessly on launch.
- An **Auto Scaling Policy** should be created, which defines the set of actions to perform when an event, such as a monitoring service (e.g., CloudWatch in this project) alarm is triggered.

The following video will demonstrate how you can use Auto Scaling (keep in mind that the latest UI is slightly different):

Show Submission Password

Video 3: Auto Scaling Demo

CloudWatch

You will be using CloudWatch extensively in this project to come up with near-optimal auto scaling policies.

Amazon CloudWatch

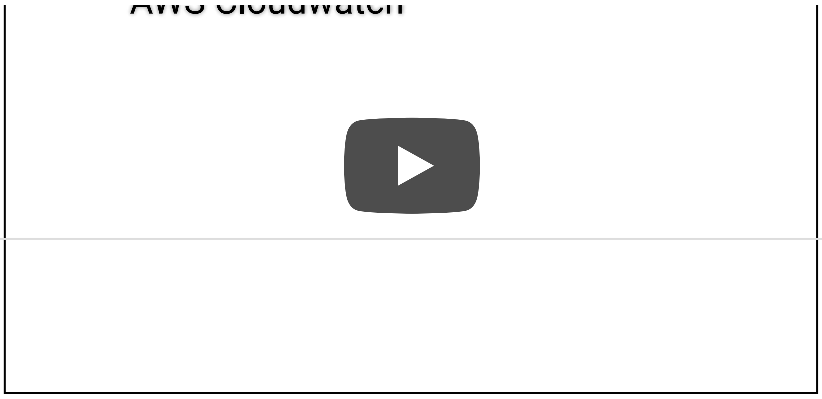
Amazon CloudWatch enables developers to monitor various facets of their AWS resources. Developers can use it to collect and track metrics from various AWS resources that are running on the AWS Cloud. CloudWatch also allows you to programmatically retrieve monitoring data, which can be used to track resources, spot trends and take automated action based on the state of your cloud resources on AWS. CloudWatch also allows you to set alarms, which constitute a set of instructions to be followed in case of an event that is tracked by CloudWatch is triggered.

CloudWatch can be used to monitor various types of AWS resources. In this project, you will monitor the following resources:

- EC2 instances
- Application Load Balancer
- Auto Scaling Group

For EC2 instances, CloudWatch allows you to monitor CPU, memory and disk utilization. For Application Load Balancer, CloudWatch allows monitoring of such metrics as request/response counts, response time, failures, etc.

For more information on CloudWatch please refer to the Amazon CloudWatch documentation (http://aws.amazon.com/documentation/cloudwatch/).



Show Submission Password

Video 4: CloudWatch

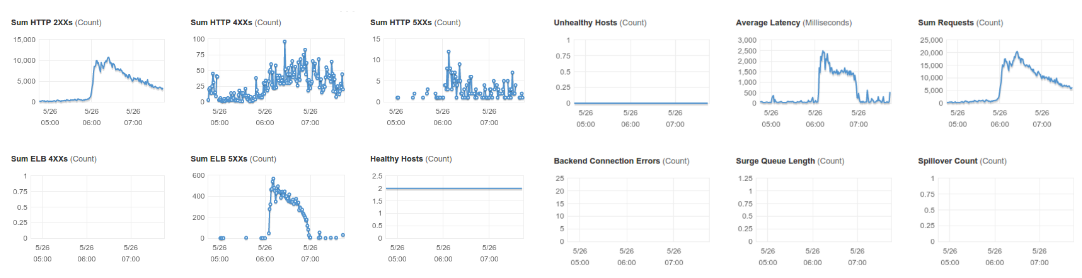


Figure 5: Example of CloudWatch Metrics

The above image shows some sample metrics captured and displayed by the CloudWatch service. These metrics can be useful when devising a policy for auto-scaling:

- **HTTPCode_ELB_2XX:** This is the count of HTTP 2XX status codes which ELB receives from the client when the request is sucessfully processed. A large number of 200 requests in the graph means that the policy is working fairly well.
- **HTTPCode_ELB_5XX:** This is the count of HTTP 5XX (which indicates a server error) that the ELB responds to the client. There are few scenarios where the ELB can also generate 5XX error e.g., if no instance is registered, the instance is unhealthy, or the request rate is higher than the ELB instance's current capacity.
- **HTTPCode_ELB_4XX:** This is the count of HTTP 4XX (which indicates a client error) status codes which the ELB responds to the client. One scenario where the ELB can respond with this status code is when the instances behind the load balancer have not fully started.
- **Latency:** TThis tells you about the performance of your web server instances. It measures the amount of time the webserver is taking to process a request.
- **Request Count:** This is the total number of requests which are received by the load balancer and routed back to instances. This CloudWatch metric can help the user understand the load pattern which can be used to make changes to the policy alarms.
- **Backend Connection Errors:** This is the number of failed conncetions to the instance behind the load balancer.
- **UnHealthyHost Count:** The UnHealthyHost count instances that have failed more health checks than the given threshold. The instance can be unhealthy if the instance health check is giving non-200 response or time-out when performing the health check.

- **Healthy Count:** The HealthyHost count instances are ready to receive requests from the Load Balancer.

Show Submission Password

Advanced Auto Scaling in AWS

MSB Senior System Architect Test

This is the final test before you pass the first round of interviews of the MSB System Architect. You will need to use the AWS APIs to create or launch everything you need for the final test. In the final test, the LG will send a changing load to your ELB, and your autoscale policy will adjust the number of instances running to achieve a required request per second (RPS) rate within reasonable instance hours (defined later).

Information

A Note on Time Warping

Unfortunately, we do not have the budget (and most of you do not have the time) to conduct 24-hour long tests. For the rest of this module you should assume that we will simulate 1 hour of MSB time in 1 minute of real time. Thus, we will simulate a full **24 hour test** in only **24 real world minutes**.

The system architect candidate gets interview points relative to a day's performance. However, to get those points you must meet the minimum requirement of creating a system with a mean RPS of over 7 for the entire 24 hours of traffic. (i.e., in our simulation that signifies a mean RPS >= 7 during the 24 minute test to get a non zero score).

Ideally, an architect should maintain a mean RPS of 12 (you will need to reach a target mean RPS >= 12 to get a full score).

During a typical day, there are several load phases reflecting activity in different parts of the world (where the application has users). During the peak phases, MSB receives a large number of requests to lookup access codes (requests to the /lookup/random page of the Web Service Instance). You are required to achieve the highest average RPS possible, with at least 35 RPS during a peak (not reaching a max RPS of 35 during the 24 minute period will reduce your score). Additionally, system architects at MSB do not have an unlimited budget. MSB keeps track of their resources using the instance-hour metric. As the name suggests, 1 instance-hour means that one instance is running for one hour. An m5.large running for 1 hour counts for 4 instance-hours. At the MSB, system architects are limited to 280 instance-hours per day. However, they are encouraged to use fewer instance-hours, to save budget. Hence, they need to balance between RPS and resource consumption. See more details in the Scoring section below.

An architect should use fewer than 220 instance hours (you will need to use instance hours <= 220 to get a full score).

Information

A Note on Instance-Hours

Instance-hours are meant to keep track of the total resources used to complete a task. As such, different instances have a different multiplier to account for their costs. Furthermore, since we are simulating MBS time in this project (remember 1 minute of test-time equals one hour of simulated MBS time) an instance-hour in simulated time is 1 minute in test-time. So, if you are limited to 280 instance-hours in the simulation your real test can only use ~3 m5.large instances if you run them for the entire 24-minute test.

Show Submission Password

Average RPS

The system architect candidate gets interview points relative to a day's performance. However, to get those points you must meet the minimum requirement of creating a system with a mean RPS of over 7 for the entire 24 hours of traffic. (i.e., in our simulation that signifies a mean RPS >= 7 during the 24-minute test to get a non zero score). Ideally, an architect should maintain a mean RPS of 12 (i.e., you will need to reach a target mean RPS >= 12 to get a full score).

Peak RPS

During a typical day, there are several load phases reflecting activity in different parts of the world (where the application has users). During the peak phases, MSB receives a large number of requests to lookup access codes (requests to the /lookup/random page of the Web Service Instance). You are required to achieve the highest average RPS possible, with at least 35 RPS during a peak (not reaching a max RPS of 35 during the 24 minute period will reduce your score).

Cost

Additionally, system architects at MSB do not have an unlimited budget. MSB keeps track of their resources using the instance-hour metric. As the name suggests, 1 instance-hour means that one instance is running for one hour. An m5.large running for 1 hour counts for 4 instance-hours. At the MSB, system architects are limited to 280 instance-hours per day. However, they are encouraged to use fewer instance-hours, to save budget. Hence, they need to balance between RPS and resource consumption. See more details in the Scoring section below. An architect should use fewer than 220 instance hours (you will need to use instance hours <= 220 to get a full score).

In order to optimize the tradeoff between increased RPS and decreased capacity instance-hours, your auto-scaling solution should use the bare minimum number of instances required for most of the day, quickly scale up to handle the spikes as they occur, and then scale down as soon as the spikes is fading. You can visualize your instance-hours using the "Healthy Hosts" graph on the ELB page. You should use an ELB to distribute traffic evenly between your instances.

You should use an ELB to divide traffic evenly between your instances. Using Auto Scaling and CloudWatch, you can scale your system to handle the traffic. In the section "Putting it all together," we give you an example of the Auto Scaling rules, but you should feel free to devise your own rules to achieve the best score.

Fault Tolerance

As the scale of an infrastructure increases, the chances for failure also grow. In order to simulate such a situation in this project, **one of the instances will be killed during the runtime of the application**. The Load Balancer will continue to send traffic to this server until they are no longer marked as unhealthy. You need to ensure that most of these requests are not lost by setting up the ELB and/or the Auto Scaling Group to detect and respond to failures using health checks.

Show Submission Password

Putting It All Together

To become a qualified MSB System architect, you are required to write a program that performs the following steps. You may also choose to use the AWS Web Console to do this for dry runs. But in order to report the test result to MSB, you need to upload code for at least one test.

Note: the AMIs provided in this project are available only in **US-EAST-1** region. Hence, you are required to work in this region only.

Initially, for testing, you may ignore setting up CloudWatch alarms and an Auto Scaling Group (Steps 2-6 below could be replaced by attaching a fixed number of instances in the Auto Scaling Group to the Elastic Load Balancer) to have a sense of the traffic pattern. Moreover, you can consider using Spot instances during initial experimentation, to save your project budget for further fine-tuning and optimization. Once you have analyzed the traffic pattern using a fixed set of instances (e.g., 5), you could continue with the full set of steps below:

For this task, you can find the starter code under `/home/clouduser/VM_Scaling_Project/<java/python>/task2/`

1 - Create two security groups to allow **incoming traffic on port 80 and all outgoing traffic on all ports**. One of them should be associated with the Load Generator and one with your Elastic Load Balancer and Auto Scaling Group.

2 - Create a Load Generator instance of size **m5.large** using `ami-0120179ee1facd28b` with one of the security groups you created in step 1.

3 - Create a Launch Configuration for the instances that will become part of the Auto Scaling Group, with the following parameters:
- AMI ID: `ami-0496215388838dae0`
- Instance Type: `m5.large`
- Detailed Monitoring: enabled

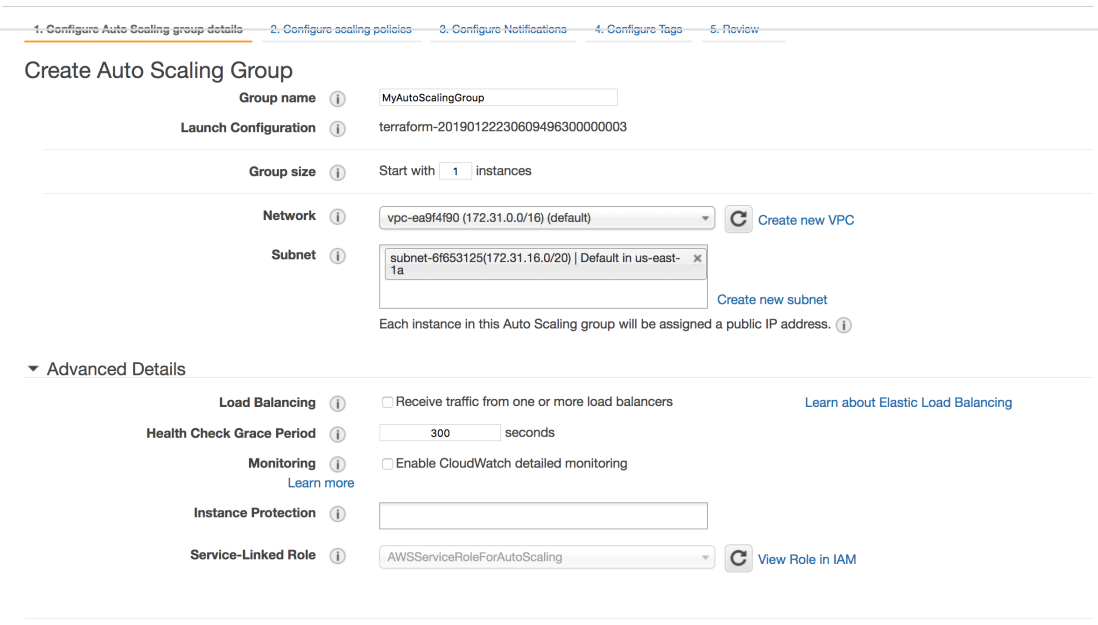
4 - Create an Application Load Balancer that forwards the HTTP:80 requests from the load balancer to HTTP:80 on the instance. You will have to use the target group from step 4 to configure the listener.

5 - Create a Target Group for instances using HTTP:80. Use / (which is the heartbeat endpoint of the web service in this project) as the Health Check path and use HTTP. Note that ELB health checks may become queued behind other requests. On a congested server, this may cause health checks to fail.

6 - After analyzing the traffic pattern from the Load Generator, figure out a good rule for Scale Out and Scale In operations. Below we define the following default rules to give you an idea of the various parameters. Part of the configuration will be covered in the next steps. - Group Size: Start with 1 instance
- Subnet: Recommended to choose the same availability zone(s) corresponding to your ELB

- Load Balancing: Receive traffic from the created Target Group
- Health Check Type: EC2
- Detailed Monitoring: enabled

Show Submission Password



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

Create Auto Scaling Group

Group name	MyAutoScalingGroup
Launch Configuration	terraform-20190122230609496300000003
Group size	Start with 1 instances
Network	vpc-ea9f4f90 (172.31.0.0/16) (default)
Subnet	subnet-6f653125(172.31.16.0/20) | Default in us-east-1a

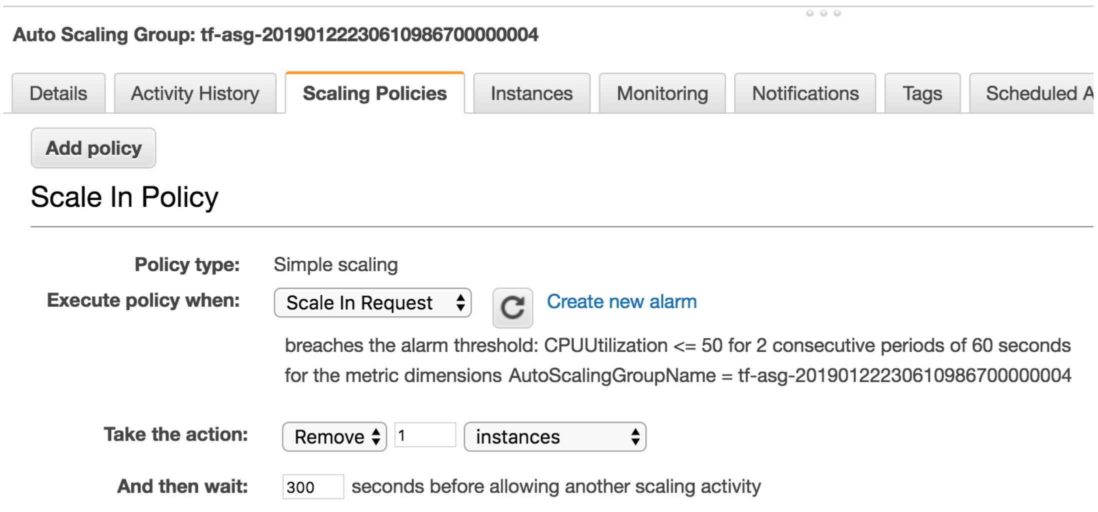
Each instance in this Auto Scaling group will be assigned a public IP address.

Advanced Details

Load Balancing	☐ Receive traffic from one or more load balancers
Health Check Grace Period	300 seconds
Monitoring	☐ Enable CloudWatch detailed monitoring
Instance Protection	
Service-Linked Role	AWSServiceRoleForAutoScaling

Figure 6: An example of Auto Scaling Group configuration

7 - Create the following scaling policies as the starting point: Please note that you may find it **difficult** to achieve a passing grade using an Auto Scaling Group with the following parameters.
- Minimum Instance Size: 1
- Maximum Instance Size: 2
- Create a Scale Out policy that automatically adds 1 instance to the Auto Scaling Group.
- Create a Scale In policy that automatically removes 1 instance from the Auto Scaling Group.



Auto Scaling Group: tf-asg-20190122230610986700000004

Scale In Policy

Policy type:	Simple scaling
Execute policy when:	Scale In Request
	breaches the alarm threshold: CPUUtilization <= 50 for 2 consecutive periods of 60 seconds for the metric dimensions AutoScalingGroupName = tf-asg-20190122230610986700000004
Take the action:	Remove 1 instances
And then wait:	300 seconds before allowing another scaling activity

Figure 7: An example of Auto Scaling Group configuration

8 - Create CloudWatch Alarms that invoke the appropriate policy for the following scenarios: Please note that you may find it **difficult** to achieve a passing grade using an Auto Scaling Group with the following parameters.
- Scale Out when the group's CPU load exceeds 80% on average over a 5-minute interval.
- Scale In when the group's CPU load is below 20% on average over a 5-minute interval.

Show Submission Password

9 - Link the CloudWatch Alarms to the Scale Out and Scale In rules of your Auto Scaling Group. You will be able to add an Auto Scaling policy with the alarms and scaling actions by editing the Auto Scaling Group.

10 - Configure correct tags for your Auto Scaling Group using your code. Java and Python with Boto3 have support for ELB tagging.

11 - Before you start, you must authenticate with the load generator. Your code must submit your submission password and username(your full andrew email address) to the load generator. Your submission password can be retrieved by clicking the "Show Submission Password" button at the top of this page. Again, the username is the full email.

Welcome to MSB Load Generator & Test Center!

Step 0. Enter your submission password
Step 1. Horizontal Scaling Test   Add Instance to Horizontal Scaling Test
Step 2. Warm Up Test
Step 3. Auto Scaling Test
Step 4. Upload Code

Test logs

12 - Your code should request the following HTTP URL to start the Autoscaling test:

`http://[load-generator-dns]/autoscaling?dns=[elb-dns]` .

You can view the progress of this test using the page:

`http://[load-generator-dns]/log?name=test.[testId].log` .

13 - You may want to the following structure to run a test:

```
//Start Test
while(!isTestComplete){ //Parse and check the INI file to see if the test is com
plete
    Thread.sleep(<=1 second) //Very small sleep time
}
//Terminate Resources
```

14 - At the end of the 24 minutes, your average RPS for the entire test and total instance hours consumed will be displayed.

```
[Minute 21]
rps=18.01

[Minute 22]
rps=16.00

[Minute 23]
rps=2.00

[Minute 24]
rps=2.00

[Load Generator]
username=student@andrew.cmu.edu
platform=AWS
instanceId=i-w9r0d7sgs87gds0s7
instanceType=m5.large
hostname=ec2-10-10-10-01.compute-1.amazonaws.com

[Elastic Load Balancer]
dns=project-2-elb-028705270.us-east-1.elb.amazonaws.com

[Test End]
time=2019-01-09 02:41:42
averageRps=13.91
maxRps=44.48
pattern=194
ih=201.20
; Instance-Hour Usage
; i-002686bf0d487g9er   m5.large    54.87   2019-01-09 02:13:54 2019-01-09 02:2
7:37
; i-025f0e87r8tb7e0be   m5.large    41.00   2019-01-09 02:31:10 2019-01-09 02:4
1:25
; i-03663fc98r7g69ec9   m5.large    23.60   2019-01-09 02:22:14 2019-01-09 02:2
8:08
; i-05460wr7bet987673   m5.large    19.27   2019-01-09 02:27:13 2019-01-09 02:3
2:02
; i-07cb8re97t9857074   m5.large    10.47   2019-01-09 02:25:32 2019-01-09 02:2
8:09
; i-0863wbr807f4c2084   m5.large    10.60   2019-01-09 02:39:00 2019-01-09 02:4
1:39
; i-08dvwr7gs962a37f5   m5.large    16.87   2019-01-09 02:37:08 2019-01-09 02:4
1:21
; i-0f8w8b7ret8721179   m5.large    24.53   2019-01-09 02:11:00 2019-01-09 02:1
7:08

; MSB is validating...
```

Show Submission Password

Figure 8: A sample test result

15 - You should terminate all the resources when you are done through your code.

Code Portability Expectations

Please make your code as portable as possible. Do not have any code that depends on your personal identifiable information such as your email address. Do not hard code details specific to your AWS account, use environment variables or configuration files instead. Resource IDs should be obtained by querying for the resources. For example, it's fine to assume a default VPC is available, but you should programmatically obtain the resource ID for it. The same holds for other resources, such as default subnets.

Show Submission Password

Performance Targets and Scoring

Instance Hours:

Your instance hour score will be between 0 and 100, proportional to the instance-hours consumed during the run in the range between 220 and 280 instance-hours. 220 or fewer instance-hours will result in a score of 100. Above 280 instance-hours will result in a score of 0.

Average Requests-Per-Second (RPS):

Your RPS Score will be between 0 and 100, proportional to the Average RPS achieved during the run in the range between 7 and 12 RPS. RPS of 12 or more will result in a score of 100. RPS of 9 will result in a score of 40. RPS below 7 will result in a score of 0.

The following figure shows how the two scores are calculated.

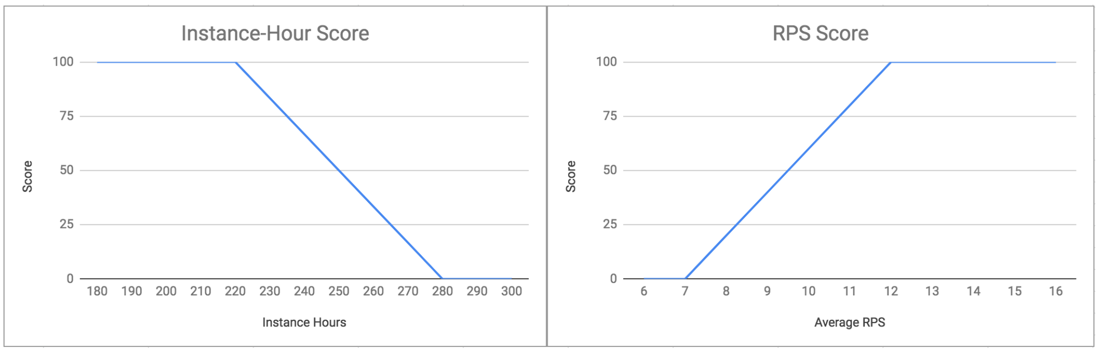


Figure 9: Calculation of Instance-Hour and RPS Scores

This section is worth 40 points.**Your Autograded Score will out of 100 will be scaled to 40 points**.**Your total autograded score is the average of the Instance-hour score and the RPS score**.

Max Requests-Per-Second (RPS)

Furthermore, if your **Max RPS is below 35, 5 points will be deducted** from your total score.

Apart from the RPS performance and instance hour score, your code will also be manually graded. Manual grading will consist of **15** points. Those points will go to coding style, proper use of cloud APIs (such as polling the status of resources to check if they are ready rather than waiting for a fixed amount of time), your description of the traffic patterns you've observed and how/why you chose your Auto Scaling policy parameters (see Step 6 of "Tasks to Complete" below).

Information

A Note on ALB Warmup

Application Load Balancer(ALB) is designed to handle large loads of network traffic provided the traffic increases gradually over a long period of time (several hours). However, if the traffic load increases over a short period of time, it becomes difficult for the ALB to handle this load.

Show Submission Password

AWS considers that if the traffic increases more than 50% in less than 5 minutes, then it means that the traffic is sent to the load balancer at a rate that increases faster than the ELB can scale up to meet it. In order to deal with these issues, the clusters need to be scaled up even before the actual load spikes through a process called warming up. Warming up an ALB using an expected pattern allows the ALB to be vertically scaled before it can handle the real test load.

Tasks to Complete

1 - Write your program in Python or Java using the AWS APIs.
2 - Your program should launch one Load Generator instance with the correct AMI and instance type. **3** - Your program should create two Security Groups, launch an Elastic Load Balancer and initiate an Auto Scaling Group along with a Launch Configuration, Auto Scale Policies and CloudWatch Alarms. **Your program should wait for all resources to be ready before it starts the warmup and/or test.**
4 - Your program should warm up the ELB before starting the system architect test. The load balancer provides an endpoint to run a warmup test for 15 minutes.

```
http://[load-generator-dns]/warmup?dns=[load-balancer-dns]
```

This warm-up test makes a large number of requests to your load balancer. You can terminate the warmup before 15 minutes by using the following endpoint:

```
http://[load-generator-dns]/test/kill
```

This endpoint can be used to terminate any running horizontal-scaling/autoscaling/warmup test.

You may want to use logic like this to run the warmup:

```
//Start Warmup API call
end_warmup_time = 15 minutes // You can use less than 15 minutes but you will ne
ed to manually kill the warmup via the API
start_time = get_current_time()

while (current_time - start_time < end_warmup_time){
    Thread.sleep(<=1 second) //Very small sleep time to save clock cycles
    current_time = get_current_time()
}
//Kill Warmup API call if ending before 15 minutes.
```

5 - Start the test by making a request to the endpoint. Your program should also record the **testId** in the response.

```
http://[load-generator-dns]/autoscaling?dns=[load-balancer-dns]
```

6 - Your program should wait for the test to complete, and upon the test completion, your program must terminate all resources (Security Group for ELB and ASG, Elastic Load Balancer, Auto Scaling Group, Launch Configuration, Auto Scale Policies, CloudWatch Alarms)

7 - Create `patterns.pdf` which answers the following questions:

Show Submission Password

```
    a) What traffic patterns did you see in the load sent to the ELB by the load generator? How did you actually figure out the load pattern? (Please provide appropriate screenshots from the AWS dashboard wherever necessary) (3 pts)

    b) How did you model the policies for the AutoScaling Group in response to the insights gained in the above question? (2 pts)
```

8 - Repeat steps 2-5 if you have budget left and hope to try with different parameters/policies. But remember, only your latest submission will be graded for every task.

9 - Do not forget to terminate all of your instances at the end using your code.

What to Submit

In addition to the above submission, you also need to upload your source code to TPZ using submitter. Your test will be graded only if you have uploaded code for that test. The submitter will upload your source code to our autograder for check-style grading. Note that your submission must contain a file called **references**, which should contain a list of references you used to complete the project. A sample reference file can be found here (https://s3.amazonaws.com/15619public/webcontent/references). Please do not include any binaries (.jar) or cloud credentials (.pem,access_key.csv) files.

How to Submit

```
    export TPZ_USERNAME=your_tpz_username
    export TPZ_PASSWORD=your_tpz_pwd
    ./task2_code_submitter
```

Danger

Warning

Timezone awareness when getting the test start time - In Project 1, we have discussed implicit environment reliance, and the timezone is another example. "2019/9/20 3:00 PM" in Pittsburgh is not the same as "2019/9/20 3:00 PM" in SV, but "2019/9/20 3:00 PM ET" always means the same time for anyone in any timezone. When you set up a global meeting, you don't want to omit the timezone information. Similarly, please make sure your program is timezone aware, and you should never use any timezone-naive local time to compare with other times. For example, when you're parsing the time from the logs, the default time value in the logs will not reflect the time value on your local system.

Information

Hints

Show Submission Password

- One full test takes `24` minutes. Plan your time accordingly. **Start early**.
- You should feel comfortable with setting up AWS resources in the AWS console (the web GUI) before you start programming; you'll likely find the console more intuitive. It's not much overhead, since you'll be able to reuse the parameters set in the console for the programming portion. The documentation for Boto, Terraform, etc. assume you know how AWS works.
- You may wish to perform several runs manually - one without Auto Scaling, the next with a set of Auto Scaling rules based on your observations of the static test, followed by a run with modified rules to perform well.
- In order to understand the characteristics of the load pattern, you may want to run the test with a fixed amount of resources a few times. Note that the pattern is not the same between runs, meanwhile the patterns are designed so that a good scaling policy can pass the test deterministically.
- Note that CloudWatch provides very interesting metrics that you can use to understand the behaviour of the system. Make sure to read the **Example CloudWatch Metrics** carefully.
- In order to fine-tune your Auto-Scaling policy without waiting for 24 minutes in each run, you can use a synthetic benchmarking tool such as loadtest (https://www.npmjs.com/package/loadtest). An example command to send requests to your load balancer with 25 RPS rate for 1 minute: `loadtest -c 8 --rps 25 -t 60 -r http://YOUR_LB_DNS_NAME/lookup/random` . Our "real" Load Generator uses a similar approach (i.e., the generated load does not change within minute boundaries).
- Scaling out quickly is tricky. Find a way to mitigate the impact of adding/removing new instances. Because of the 24-minute simulation of 24 hours, the overhead of instance launching/termination time is effectively significantly amplified.
- Pay close attention to the peak RPS to determine the maximum number of instances required.
- The rules to scale out and scale in are important, but you should also consider configurations such as the scale out/in cool-down period, auto-scaling group cool-down period, draining, health checks, etc.
- The availability zone you choose may have an impact on your test result. Creating all instances in the same availability zone would potentially lead to better results.
- A good scaling policy can and should pass the test deterministically. Your final score will be decided by your most recent submission, not the highest!
- Do not store your Andrew ID/Submission password in the code. Read it from a file or environment variables. For submission, we will expect TPZ credentials to be passed through environment variables; AWS credentials will be passed through ~/.aws/credentials. You may find it helpful to use the AWS CLI to generate a credentials file by using **aws configure**.
- If you think your API calls are correct, but they fail to get resources within a reasonable amount of time (5-10 minutes), it's possible that your request is asking for resources of an availability zone in high demand. Spot instances may be hard to acquire in certain availability zones. Auto Scaling groups with spot instances may fail to acquire instances if you restrict instances to one availability zone. We suggest that you use a single availability zone first, if you experience high delay of resource provisioning, then consider switching availability zone or using multiple availability zones.
- Be sure to delete Auto Scaling groups before terminating the instances created by Auto Scaling. Note that, terminating instances without deleting the Auto Scaling

Group first may result in Auto Scaling Group trying to boot more instances.

- Endpoint: /warmup?dns=[] can be used to start a warmup test, which
  - Returns 200 if the warmup has been started successfully.
  - Returns 400 if your ELB is not responding or there is another test running. Please check the Troubleshooting section for more information.
  - Returns 500 if your web service is not responding. Please check the Troubleshooting section for more information.
  - Ignore the RPS reported during the warmup at the beginning, as it takes time for your ELB to scale and start accepting requests at a large scale.
- API: /autoscaling?dns=[] can be used to start a new Auto Scaling test
  - Returns 200 if a test has been started successfully.
  - Returns 400 if your ELB DNS is not working as expected or there is another test running. Please check the Troubleshooting section for more information.
  - Returns 500 if your web service is not responding. Please check the Troubleshooting section for more information.
- You should be able to access your web service via http://[ELB Public DNS]/lookup/random before you can submit.
- Your RPS should eventually grow during the warmup.

Show Submission Password

Warning

Cloudwatch Monitoring Hints

During previous semesters, we observed that students who used the trial and error approach to tune the policy ended up spending too much time and some ran out of budget and time. It is advisable that you learn and follow best practices to create a policy that can work with the test patterns. The load pattern is designed so that a structured approach to monitor the pattern and tune the policy will be more likely to succeed. Observe and analyze the pattern, experiment with a policy, collect data to verify why it achieved a certain performance, and iterate until you achieve your goal.

Under the Autoscaling tab under the AWS dashboard, you can `Monitoring > Enable Autoscaling Group Metrics` . These metrics describe the group rather than any single instance in the target group.

We suggest that you monitor the pattern by the metrics included but not limited to:

- ELB
  - Total Request Count
  - Number of 200 requests
  - Number of 400 requests
  - Number of 500 requests
- EC2
  - CPU Utilization
- ASG
  - Number of healthy/unhealthy instances

Information

Troubleshooting

- If you observe unhealthy hosts for more than one short period of time during the run, check your health check configuration, timeouts, etc. Note that sometimes ALB health checks are less reliable when instances are under a heavy load because health check requests are queued together with regular requests. Think of what happens when instances are incorrectly identified as unhealthy when they are under a heavy load.
- If your CloudWatch alarm is stuck in `INSUFFICIENT_DATA` state, make sure you set all parameters correctly for the alarm (especially `Unit` and `Namespace`).
- If your RPS is nearly zero (0 <= RPS <= 1), please
  - Check if there is at least one healthy instance connected to the ELB. Note that your solution should be tolerant to instance failures (i.e., when an instance is stopped or terminated - e.g., due to a hardware failure).
  - Check if appropriate parameters are used when creating Scaling policies and CloudWatch alarms.
- If you are receiving 400 status code requests, make sure that the web service instances behind the load balancer are healthy/have completely started
- A large number of 500 requests on the graph shows that you're scaling slowly such that the web server on your instance is getting overloaded

Show Submission Password

Additional Resources and References

Autoscaling Developer Guide (http://awsdocs.s3.amazonaws.com/AutoScaling/latest/as-dg.pdf)

Autoscaling API Guide (http://awsdocs.s3.amazonaws.com/AutoScaling/latest/as-api.pdf)

CloudWatch Documentation (http://aws.amazon.com/documentation/cloudwatch/)

Elastic Load Balancing Documentation (http://aws.amazon.com/documentation/elastic-load-balancing/)

AWS Autoscaling with Terraform

AWS Autoscaling with Terraform

Before completing this task, review the Infrastructure as Code primer to gain a better understanding of infrastructure automation and the relevant tools.

Information

Task Notes

1. You are not required to run the 24 minute test for this task.
2. You are only required to complete the terraform template.

3. We will grade this task manually to verify if the required resources with the defined settings have been added.
4. You should assign the values to all the variables you define so that Terraform will not ask the user for manual input of unassigned variables when running `terraform plan`.

Show Submission Password

Information

Tasks to Complete

In the previous task you used the AWS APIs to orchestrate various AWS resources and launch an autoscaling web service. From the examples provided in the Infrastructure as Code primer, you may expect that infrastructure automation tools could be used to deploy similar or even more complex architectures.

Starting from the provided Terraform configuration template file - `task3-terraform.tf` under `task3` folder, you will complete the configuration so that it achieves the autoscaling architecture defined in the previous task. Your task is to implement the Terraform template to:

1. Create two security groups to allow **incoming traffic on port 80 and all outgoing traffic on all ports**. One of them should be associated with the Load Generator and one with your Elastic Load Balancer and Auto Scaling Group.

2. Create a Load Generator instance of size `m5.large` using `ami-0120179ee1facd28b` with one of the security groups you created in step 1.

3. Create a Launch Configuration for the web-service instances that will become part of the Auto Scaling Group, with the following parameters:
   - AMI ID: `ami-0496215388838dae0`
   - Instance Type: `m5.large`
   - Detailed Monitoring: enabled
4. Create an Application Load Balancer that forwards the HTTP:80 requests from the load balancer to HTTP:80 on the instance. You will have to use the target group from step 4 to configure the listener.

5. Create a Target Group for instances using HTTP:80. Use / (which is the heartbeat endpoint of the web service in this project) as the Health Check path and use HTTP. Note that ELB health checks may become queued behind other requests. On a congested server, this may cause health checks to fail.

6. After analyzing the traffic pattern from the Load Generator, figure out a good rule for Scale Out and Scale In operations. Below we define the following default rules to give you an idea of the various parameters. Part of the configuration will be covered in the next steps.
   - Group Size: Start with 1 instance
   - Subnet: Recommended to choose the same availability zone(s) corresponding to your ELB
   - Load Balancing: Receive traffic from the created Target Group
   - Health Check Type: EC2
   - Detailed Monitoring: enabled

7. Create the following scaling policies as the starting point: Please note that you may find it **difficult** to achieve a passing grade using an Auto Scaling Group with the following parameters.
    - Minimum Instance Size: 1
    - Maximum Instance Size: 2
    - Create a Scale Out policy that automatically adds 1 instance to the Auto Scaling Group.
    - Create a Scale In policy that automatically removes 1 instance from the Auto Scaling Group.

Show Submission Password

8. Create CloudWatch Alarms that invoke the appropriate policy for the following scenarios: Please note that you may find it **difficult** to achieve a passing grade using an Auto Scaling Group with the following parameters.
    - Scale Out when the group's CPU load exceeds 80% on average over a 5-minute interval.
    - Scale In when the group's CPU load is below 20% on average over a 5-minute interval.
9. Link the CloudWatch Alarms to the Scale Out and Scale In rules of your Auto Scaling Group. You will be able to add an Auto Scaling policy with the alarms and scaling actions by editing the Auto Scaling Group.
10. Configure correct tags for your Auto Scaling Group using your code.

What to Submit

In addition to the above submission, you also need to upload your source code to TPZ using submitter. Your test will be graded only if you have uploaded code for that test. The submitter will upload your source code to our autograder for check-style grading. Note that your submission must contain a file called **references**, which should contain a list of references you used to complete the project. A sample reference file can be found here (https://s3.amazonaws.com/15619public/webcontent/references). Please do not include any binaries (.jar) or cloud credentials (.pem,access_key.csv) files.

How to Submit

```
export TPZ_USERNAME=your_tpz_username
export TPZ_PASSWORD=your_tpz_pwd
./task3_code_submitter
```

Project Reflection Task (Mandatory, graded)

Project Reflection Task (Mandatory, graded)

Upon completing this project you will make one (1) post in the [Forum] Horizontal Scaling and Advanced Resource Scaling (https://theproject.zone/cloud-forum/topic/publish/132/) to reflect on your experience before the project deadline. **Before you publish the post, please double**

check that the category is the project category "Horizontal Scaling and Advanced Resource Scaling", NOT the course category "Cloud Computing" or any other project category.

Consider the following topics when creating your post, however, you should never share any code snippets in your reflection:

Show Submission Password

- Describe your approach to solving each task in this project. Explain alternative approaches that you decided not to take and why.
- Describe any interesting problems that you had overcome while completing this project.
- If you were going to do the project over again, how would you do it differently, and why?

After completing this task, confirm that your ***Reflection Score*** has been updated on the scoreboard before the project deadline.

Success

If you have successfully completed all the tasks and obtained a score, congratulations and welcome to the MSB.

Now you have already got some experience working with virtual machines and adding scalability, performance and fault-tolerance to your applications. In the next project, we will show you how to play with docker and kubernetes, which are very popular in industry. Apart from their scalability, performance and fallout-tolerance, they are also very easy to use and benefit the continuous integration and continuous deployment.

Please leave us feedback for this project here. This will help us strengthen this project for future offerings.

Feedback

Show Submission Password

Please leave us feedback so that we can improve this project in future offerings!

* Required

How many total hours did you invest in this project? *

Your answer

Which parts of this project were frustrating and why? *

Your answer

How could you use what you learned in this project in the real world? *

Your answer

What learning topics did we omit in this project and what new ideas should we consider? *

Your answer

Rate the Cloud Service Provider you used this week *

	Very Poor	Poor	Fair	Good	Very Good	Did not use
AWS: Familiarity before project	○	○	○	○	○	○

Project Discussion

Show Submission Password

Project Discussion (Mandatory, graded)

After this project has completed (after the project deadline), all reflection posts by all students will become visible for review.

Your task is to reply and provide feedback to **3** posts in the [Forum] Horizontal Scaling and Advanced Resource Scaling (https://theproject.zone/cloud-forum/category/132/), within **7** days after the project deadline.

After completing this task, confirm that your Discussion Score has been updated on the scoreboard within 7 days after the project deadline.