# *CLOUD COMPUTING COMPETITON*



Version: 1.4

# PROJECT AND TASK DESCRIPTION

The goal of this project is to deploy and scale a web application to support the Reservations API and Web Lookup services.

You need to use two pre-provided binary files based on the "Go" (Golang) programs to deploy a highly available, scalable, and efficient web application. These binaries have different service dependencies, as described in the "Technical Details" section below. These applications can respond well to caching and horizontal scaling.

Additionally, for security reasons, the download links for the two "Go" program binaries have been placed in the Parameter Store by the operations team under the name "findbin". Please analyze your current initial architecture and permissions information to find a way to obtain the binary files.

⚠ ***Note***: Do not delete the EC2 resources in the initial architecture before obtaining the binary files, as this may prevent you from completing the challenge!

These applications are not "ready to use"; instead, you will have 30 - 50 minutes to launch the initial infrastructure. After the challenge begins, your application will start receiving requests at the root application. If you fail to launch a valid solution, you will start losing points. During the challenge, you may need to address any issues that arise. Challenges may come not only from dynamic traffic changes but also from the initial architecture itself.

# INITIAL STATE

The API is currently not working properly, and it is not generating any traffic either. Requests will start being sent to your API 60 minutes after the challenge begins.

# TASKS:

1. Log in to Cloud Raiser using the assigned hash value (provided on the day)
2. Set your team's name (format: as per the participation requirements)
3. Read the documentation carefully, as it will be very helpful
4. Log in to the AWS Console
5. Check the existing configuration in EC2 (Elastic Compute Cloud service)
6. Check the existing configuration in the VPC (Virtual Private Cloud), Subnets
7. Configure the application to auto-scale to handle the increasing load
8. Configure the relevant server dependencies as described in the “Technical Details”
9. Your most important task is to launch and ensure the availability and scalability of the ROOT application
10. Configure necessary application monitoring, metrics, and alarms in CloudWatch
11. Monitor the performance of the application servers through the "Score Events and

Scoreboard" as well as via the AWS Console and CloudWatch

12. Earn scores by receiving requests; refer to "Score Events and Scoreboard" to ensure you are gaining positive scores for the provided services
13. Monitor costs, and avoid over-scaling the infrastructure to minimize point deductions
14. Traffic volume will change over time and continue to fluctuate

# TECHNICAL DETAILS:

1. The **Root** server application is a Go binary file compiled from source code. **Do not modify the binary file in any way, or you will be disqualified from the competition.**
2. The **Lookup** server application is a Go binary file compiled from source code. **Do not modify the binary file in any way, or you will be disqualified from the competition.**
3. In the competition, use EC2 as your compute resource. **Do not use EKS, ECS, or Fargate, or you will be disqualified.**
4. The root application can handle around 5 connection requests before it starts to slow down. Be careful of overloading and watch for HTTP 5XX responses from the server when the queue is full.
5. The ROOT server in this version has many other requirements to run. In addition to the basic requirements listed above, it must also meet the following service requirements:

    a) It must have permission to listen on the defined TCP port (default port 80)
    b) It must have a configuration file provided in the form of "server.ini"
    c) It must be able to access a running Memcached server
    d) It must be able to access the Api-gateway endpoint
    e) The server must be able to write to a log file named "app_logs" in the directory provided in the server configuration file
    f) It must have permission to upload files to an S3 bucket

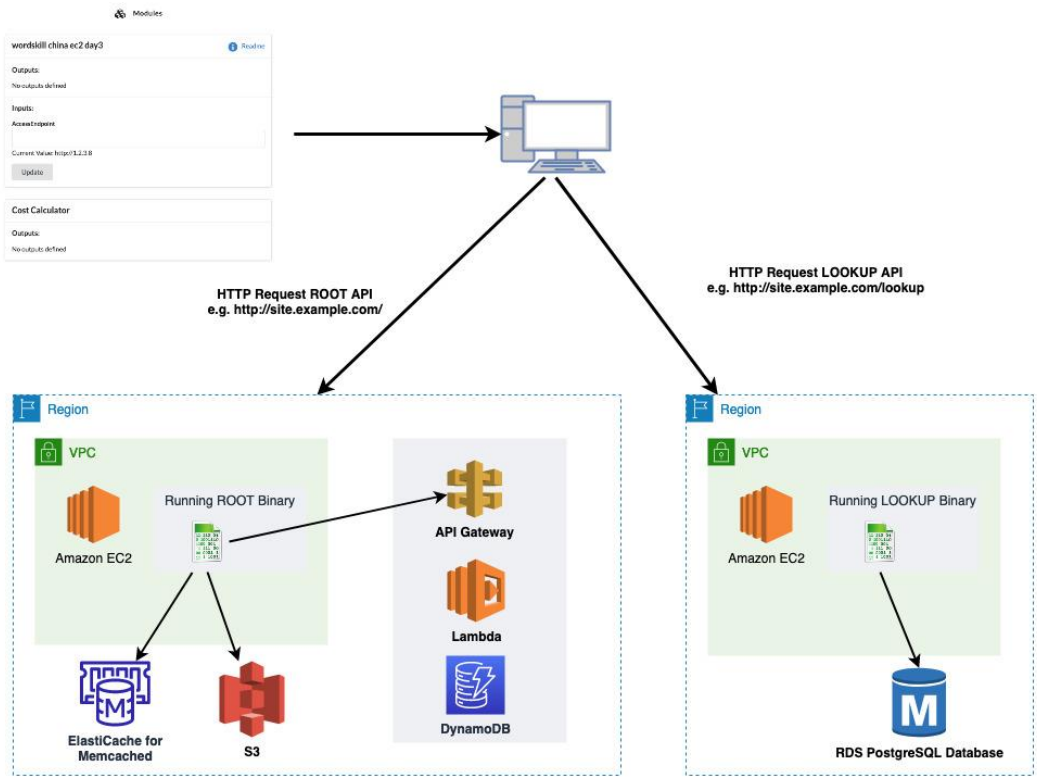
6. The LOOKUP server in this version also has many other requirements to run. In addition to the basic requirements listed above, it must also meet the following service requirements:

    a) It must have permission to listen on the defined TCP port (default port 80)
    b) It must have a configuration file provided in the form of "server.ini"
    c) It must be able to access a running PostgreSQL server
    d) The server must be able to write to a log file named "app_logs" in the directory provided in the server configuration file

7. These servers need to connect to Memcached and PostgreSQL databases to store data. The more efficiently the infrastructure runs, the faster the servers will respond to requests, and the more points you will earn.

8. The chosen base operating system is Amazon Linux Version 2.
9. Using the AWS CLI (Command Line Interface) may be very useful. For information on installing this tool locally, visit http://docs.aws.amazon.com/cli/latest/userguide/installing.html. The AWS CLI is already installed on Amazon Linux EC2 instances.

# REFERENCE ARCHITECTURE




The example above illustrates one possible architectural design for deploying the applications, **provided for reference purposes only.**

# SERVICE DETAILS

### Overview

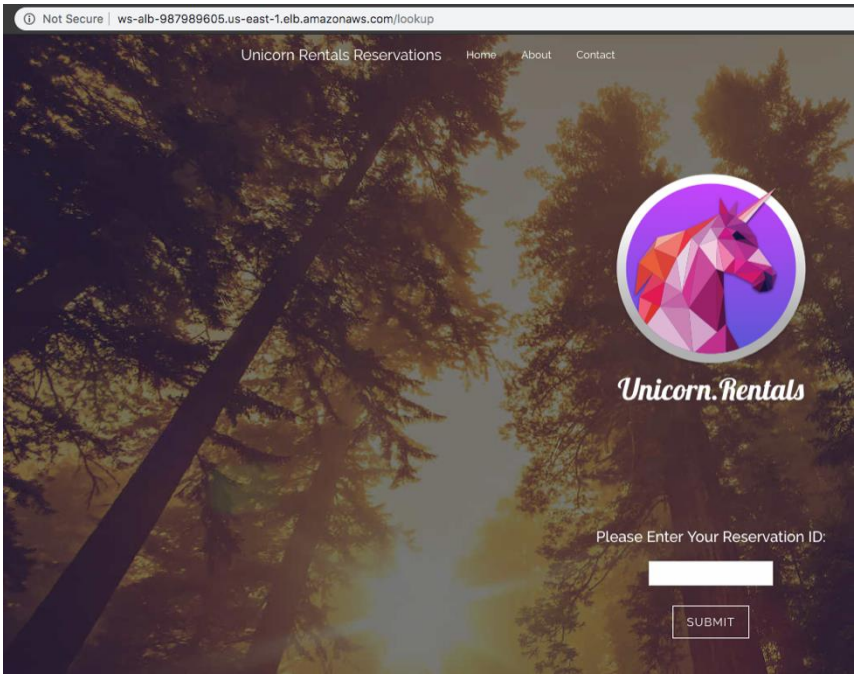
Once the servers you set up receive a request, the data needed to respond the request will be stored in a database. If you receive a request that has been responded to previously, no new data will be created. If your servers use a centralized solution for each technology, then all instances can access the response data regardless of which instance responded to the request. However, if you deploy database servers on each instance, the response answers will not be shared, and each server will eventually need to create a response for every request.

### URL Routing

You will need to provide a single URL address as the answer, which will be used to access

your two applications. To facilitate this, you will need to perform URL routing. The application identified as "root" should run at the root directory of your URL. Therefore, if deployed correctly, when you access the public URL of your application, you should see the following screen:


You need to deploy the second application named "lookup" to the "/lookup" URL instead of the main address. If your main address is http://server.example.com, then clicking on that main site in a browser should return the image shown above. However, going to http://server.example.com/lookup should display the following page:




You can achieve this by deploying the application named "lookup" to the "/lookup" suffix of your URL. The URL you enter will receive requests at the root directory of the URL as well as the "/lookup" endpoint.

After deploying the applications, default reservation information will be loaded on the database server. Examples can be found below. If your installation is correct, you should be able to query these reservation details by entering one of the reservation ID values listed

below:

```
5000001
5000002
5000003
```

**Root Application - Memcached Database**

You need to provide a Memcached database solution for the root application. This is not an I/O-intensive workload and should be deployed in the most economical and efficient way possible.

After completing this, you need to instruct each server on how to use this service through the "**server.ini**" file deployed to each instance. The configuration section related to Memcached is shown below:

```
"MemcacheHost" = "127.0.0.1"
"MemcachePort" = "11211"
```

MemcacheHost is the hostname used to connect to the MemcacheHost service.
MemcachePort is the TCP port number used to connect to the service (default is 11211).

**Root Application - S3 Bucket**

Your application will need access to S3 and permission to upload reservation information (refund requests). An EC2 Profile with a name in the format 'xxx-EC2InstanceProfile-xxx' will exist in your AWS account to use.

You need to instruct each server through the "**server.ini**" file deployed to each instance, which S3 bucket the reservation information (refund requests) will be uploaded to.

```
"AwsS3Bucket" = "aaa-bbb-ccc-ddd-eee-xxxyyyzzz"
```

**Root Application - API Gateway + Lambda + DynamoDB**

Your application will need access to an API GATEWAY endpoint, which will trigger a Lambda function already created in your account (named in the format 'xxx-LambdaFetchDynamoDb-xxx'). This Lambda function will access a DynamoDB table (table name: **ws-account-table**) that you need to create, with the partition key (Partition_key) being **account_id** and the value being your AWS account ID.

You need to instruct each server through the "server.ini" file deployed to each instance, which API Gateway endpoint will be accessed.

```
```

```
"APIGWUrl" = "https://abc123.execute-api.us-east-1.amazonaws.com/v1/"
```

### Lookup Application Structure

Throughout the day, your deployed reservation lookup system will receive new booking data from external systems. You must ensure your application is available, as these bookings will only be received once. If your application misses this information, it will not be resent. Unicorn Rentals' customers will periodically use the system you deployed to look up their booking details. If their reservation is not listed in the system, you will be deducted points.

### Lookup Application – PostgreSQL

You need to provide a PostgreSQL database solution for the application. This is not an I/O-intensive workload and should be deployed in the most economical and efficient way possible. After deploying the PostgreSQL service, you need to create the tables required to serve the requests. Two tables must be created, one for the website content management system, and one for storing unicorn rental booking information. Examples of table definitions, as well as default data for these tables, can be found in the **database.sql** file.

After completing this, you need to instruct each server on how to use this service through the "**server.ini**" file deployed to each instance. The configuration section related to PostgreSQL is shown below:

```
"PgsqlHost" = "localhost"
"PgsqlPort" = "5432"
"PgsqlUser" = "unicorndb"
"PgsqlPass" = "unicorndb"
"PgsqlDb" = "unicorndb"
```

*PgsqlHost* is the hostname used to connect to the PostgreSQL service.
*PgsqlPort* is the TCP port number used to connect to the service (default is 5432).
*PgsqlUser* is the username used to connect to the service. Using the root username is not best practice.
*PgsqlPass* is the password assigned to the user (as mentioned above) for accessing this service.
*PgsqlDb* is the name of the database where you created the new tables. You will likely need to create this database name as well.

### Lookup Application – Security

Our booking lookup system has recently been found to be vulnerable to SQL injection attacks, and we have discovered someone is exploiting this vulnerability to tamper with our data. They have been changing the access codes for the unicorns our customers have rented to "H@XX0RR333DDD". The developers who created the application are no longer available, so we hope you can find a way to ensure the application's availability.

**Requests that will be sent to your application:**
**(assuming your URL is http(https)://server.example.com):**

Requests that will be sent to the ROOT application:
http(https)://server.example.com
http(https)://server.example.com/healthcheck
http(https)://server.example.com/calc

Requests that will be sent to the LOOKUP application:
http(https)://server.example.com/lookup
http(https)://server.example.com/lookup/add
http(https)://server.example.com/add
http(https)://server.example.com/search
http(https)://server.example.com/lookup/search