Test Project Day two

Cloud Computing

Submitted by: Taze Miller US

Contents

This Test Project consists of the following documentation/files:

- WS2019_Kazan_Cloud_Computing_Main
- WSC2019_TP53_DayOne_actual
- **WSC2019_TP53_DayTwo_actual**
- WSC2019_TP53_DayThree_actual
- WSC2019_TP53_DayFour_actual

Description of project and tasks

This module is six hours - **Day 2: Reservations API**.

The goal this test project is to deploy and scale a public client api web application to support a reservations API and web lookup service.

Deploy a highly available, scalable, and efficient web application using the two 'go' (Golang) server binaries provided. These binaries have different service dependencies as outlined in the Technical Details section below. These applications, respond well to caching as well as horizontal scaling.
These applications will not work "out of the box". Instead, you will have 30 minutes to get an initial infrastructure rolled out. 30 minutes from the start time of the event, your applications will start receiving requests on the root application. If you do not have a solution started that functions, you will lose points. 15 minutes after starting to receive requests on the root application, you will then start to receive data on the lookup application (45 minutes from start of event).

Details on the service requirements are below.

Throughout the day it is your responsibility to respond to any challenges that may present themselves. Once the day has concluded, additional load testing will be conducted to determine your architecture's resiliency and ability to self-repair.

Four hours into the competition we will introduce two new challenges for you to complete that are unrelated to your previous architecture. As a solutions architect, you may be required to perform many different types of tasks. These challenges will gauge your ability to change tasks and respond to an ambiguity.

Initial state

The API is not currently functional but there is also no traffic being generated. Requests will begin 30 minutes after the start of the competition.

Tasks

1. Log into Gameday with your assigned hash (Provided on the day)
2. Set your team/competitor name on the Dashboard – (Format: NAME SURNAME)
3. Read the documentation thoroughly (Outlined below)
4. Log into the AWS console (link provided from the Dashboard)
5. Examine existing configurations in EC2 (Elastic Cloud Computer server)
6. Examine existing configurations in VPC (Virtual Private Cloud, Network Segment)
7. Configure application to auto scale to handle increasing load (Auto Scaling Groups, with Launch Configuration)
8. Configure any server dependencies as outlined in the technical details
9. Configure necessary application monitoring, metrics and alarms in CloudWatch
10. Monitor performance of the application servers in the "Score Events and Scoreboard" and through the AWS Console with CloudWatch
11. Serve client requests to gain points, reference the "Score Events and Scoreboard" to ensure you are scoring positively by serving the requests
12. Monitor costs and do not scale up the infrastructure excessively to minimize penalties
13. Process exceptions when they are received, reference the "Request Exception Handling" below
14. This section of the test project will continue for 4 hours. At this point, you will lose access to your architecture and will need to begin working on 2 additional tasks.
15. You will be given the option to view 'clues' in solving your task. Clicking to reveal a clue will result in a penalty.

Technical Details

1. The root server application is deployed as a Go binary compiled from source. Do not alter the binary in any way as that will be grounds for disqualification.
2. The lookup server application is deployed as a Go binary compiled from source. Do not alter the binary in any way as that will be grounds for disqualification.
3. For this day of competition focus on EC2 as your compute resource. Do not use ECS or Fargate.
4. The root serve application can handle about 5 connections before starting to get slow. Be careful about overloading and watch for HTTP 5XX responses from the server when the queue fills.
5. The server applications are x86 statically linked, unstripped ELF executables found via an S3 URL provided during the test project in the player README file in the player dashboard.
6. The ROOT server in this version has many more requirements in order to run. In addition to the baseline requirements listed below there are additional service requirements outlined below.

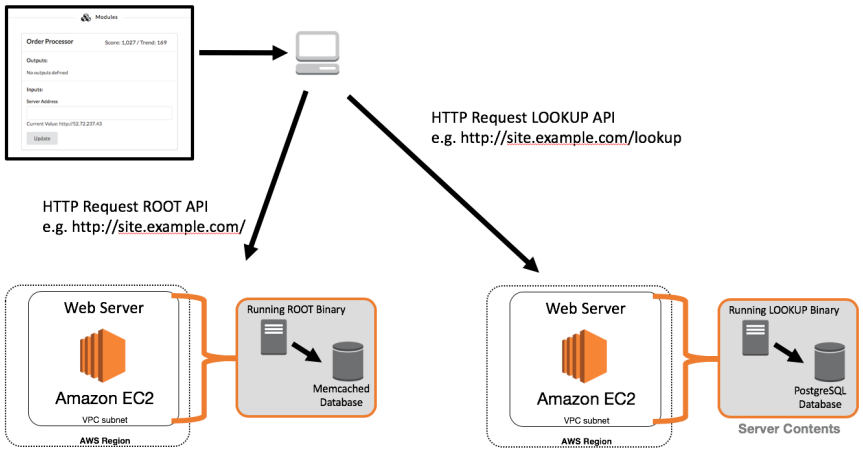
   (a) It must have permissions to listen on the TCP port defined (default port 80)
   (b) It must have a configuration file supplied as "server.ini" (an example is provided below)
   (c) It must have access to a running Memcached server.
   (d) The server must be able to write to a log file. The file is named "app_log" and located in the directory supplied in the server configuration file.

7. The LOOKUP server in this version has many more requirements in order to run. In addition to the baseline requirements listed below there are additional service requirements outlined below.

   (a) It must have permissions to listen on the TCP port defined (default port 80)
   (b) It must have a configuration file supplied as "server.ini" (an example is provided below)
   (c) It must have access to a running PostgreSQL server.
   (d) The server must be able to write to a log file. The file is named "app_log" and located in the directory supplied in the server configuration file.

8. These servers require a connection to a Memcached database and PostgreSQL database to store data. The more efficient you run the infrastructure, the faster your servers will respond to requests and the more points you will earn.
9. The base OS that has been chosen is **Amazon Linux Version 1** (https://aws.amazon.com/amazon-linux-ami/). This distribution was selected for its broad industry support, stability, availability of support and excellent integration with AWS.
10. Use of the AWS CLI (Command Line Interface) can be very helpful. Information on installing this tool locally can be found at http://docs.aws.amazon.com/cli/latest/userguide/installing.html. The AWS CLI is already installed on Amazon Linux EC2 instances.

Architecture



The above example illustrates one possible architectural design for the deployment of the application. This shows all required segments needed to server requests.

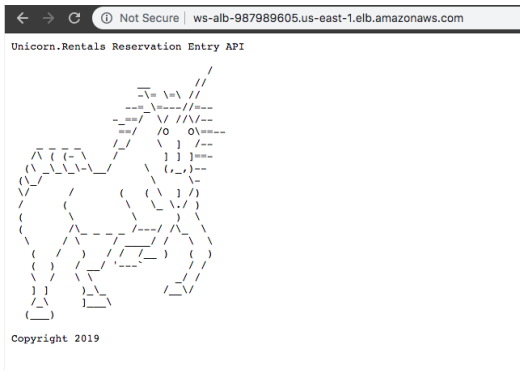
Service Details

Overview

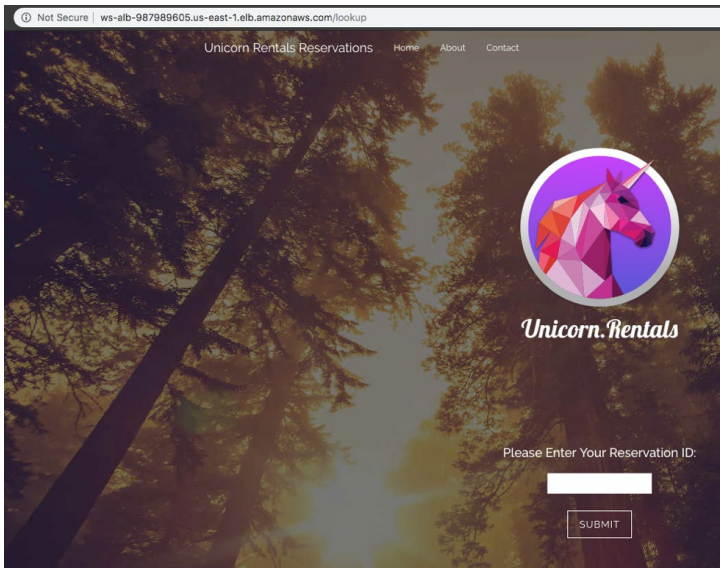
Every time a request is sent to a server that you have deployed, the process must generate a response to send out (very slow). However, once you have answered a response, the data necessary to respond is stored in a database. If you received a request and have already answered it previously, you will not have to create a new answer, you will already have the answer that your server will respond with. If your server is using a centralized solution for each of these technologies, then it does not matter which instance responded to the request, all of the instances will have access to the answer. If however, you have a database server deployed on each instance, the answers will not be shared and each server will end up having to create a response for each request.

URL Routing

You will provide a single URL that the application will use to access both of your applications. In order to facilitate this, you will need to perform URL routing. The application identified as "root" should run on the root of your URL. Thus, if deployed properly, when you visit the public URL of your application, you should see the screen below:



You will need to deploy the second application named, "lookup" to the URL of "/lookup" off of your main URL. If your main URL is http://server.example.com, then hitting that main site in your browser should return the image above. Going to http://server.example.com/lookup should however show the page below:



You can accomplish this by also deploying the application named "lookup" to the "/lookup" suffix to your URL. The URL that you enter will receive requests on the root of your URL as well as the "/lookup" endpoint.

Once you have your application deployed, there are default reservations loaded on your database server. Examples can be found below. If your installation is correct, you should be able to search for these reservations inputting one of the reservation ID values listed below:

```
5000001
5000002
5000003
```

Root Application – Memcached Database

You will need to provide the root application a Memcached database solution. This is not an I/O intensive workload and should be deployed in the most cost-effective way possible. While the application will function with a Memcached server deployed on each instance running the application, you will not benefit from previous instance processes.

Once complete, you will need to give each server the instructions on how to use this service via the "**server.ini**" file deployed to each instance. The section relevant to Memcached is shown below:

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

MemcacheHost is the hostname used to connect to the Memcached service.

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

Lookup Application Structure

Throughout the day, the reservation lookup system that you deploy will receive new reservation data from an external system. You must ensure the availability of your application as these reservations are only received once. If your application misses the receipt of this information, it will not be sent again. Periodically, Unicorn Rentals customers will look for their reservation details using the system that you deploy. If their reservation is not listed in the system, you will be deducted points. Every time a Unicorn Rentals customer rents a unicorn they must whisper their Unicorn Access Code into the ear of their unicorn to activate their rental. Without this access code, customers are unable to use their rental.

Lookup Application – PostgreSQL

You will need to provide the application a PostgreSQL database solution. This is not an I/O intensive workload and should be deployed in the most cost-effective way possible. While the application will function with a PostgreSQL server deployed on each instance running the application, you will not benefit from previous instance processes. Once you have a PostgreSQL service deployed, you will need to create the tables necessary to serve the requests. There are two tables which must be created, one table for the website content management system and one table to store the unicorn rentals reservation information. Example table definitions as well as default content for these tables can be found via the database.sql file linked in the player README file in the player dashboard.

Once complete, you will need to give each server the instructions on how to use this service via the "**server.ini**" files deployed to each instance. The section relevant 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. It is NOT a best practice to use the root username.

PgsqlPass is the password assigned to the user (defined above) for access to this service.

PgsqlDb is the name of the database that houses the new table you have created. You will most likely need to create this database name as well.

Lookup Application – Security

Unicorn Rentals, LLC has recently found out that their legacy reservation lookup system is vulnerable to SQL injection attacks. One of our competitors, Unicorns.Direct has found out about this vulnerability and is using this exploit to corrupt our data. They have been changing the Unicorn Access Code for our customer rentals to "H@XX0RR333DDD" in order to drive business to Unicorns.Direct. The developers that created the application are no longer available and thus we are hoping that you will be able to find a way to ensure the availability of the application while also emphasizing security.

AWS Jam – S3 Access Control

As part of the technical due diligence team, you need to audit (and possibly fix) some of the S3 setup. Our data is super important; but are they adequately protected? You are in charge of finding out!

You are in charge of fixing some bucket permissions after your new boss has seen that some objects can be read without company credentials. Unfortunately, the S3 buckets policy or IAM policy has some permissions that allowed external entities to browse and get those sensitive files.

You can review the past logs and try to find from where the breach happened or apply a new configuration that don't disrupt the current employees' access while preventing any external user from reading the company data.

Buckets permissions and Object permissions are your scope

Here's some stuff you need to know:

- There is a URL presented within output parameters that will provide you an update on your status in completing this challenge. Check this regularly to monitor your progress.
- Some buckets host web content and must remain open to the public
- **Don't touch buckets with names like *-challenge-* and *-status-* or risk penalty**
- Internal users must be able to continue using S3 buckets, but can't expose objects to the internet

Good starting points

- Search S3 buckets from **command-line or using scripts** to speed up the process
- Enable S3 logging if necessary to watch what is happening

AWS Jam - From Monoliths to Containers

Your organization has decided to embrace containerization as a migration path for REST API applications. As a starting point, you have been asked to migrate an existing REST API application from on-premises to Amazon ECS in a cost effective way with minimal infrastructure management. In this challenge you will be working with Amazon EC2 Container Service (Amazon ECS), Amazon EC2 Container Registry (ECR), Amazon EC2 and building Docker images.

You have a python REST API application hosted on an Amazon EC2 instance (to simulate the on-premises environment), which also have Docker pre-installed. The python code **app.py** and Dockerfile for the REST API resides in a folder called **app**. The objective of this challenge is to migrate this application to ECS and have the REST API application running inside a container.

Check the output properties in the Jam console for the for the **checkChallengeStatusURL**. Use this URL to check the progress of your migration.

When the challenge has been successfully completed, the response from the **checkChallengeStatusURL** will provide the answer.

Please refer to the **WS2019_Kazan_Cloud_Computing_Main** document for more details