Test Project Day One

Cloud Computing

Submitted by: Taze Miller US

WSC2019_T53_DayOne_actual

Contents

This Test Project consists of the following documentation/files:

- WS2019_TP53_Main_document
- **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 1: Billing API**.

The goal this Test Project is to deploy and scale a public client api web application to support a billing API.

Today you will deploy a highly available, scalable, and efficient web application using the 'go' server binary provided. This binary has service dependencies as outlined in the Technical Details section below. This application responds well to caching as well as horizontal scaling.

This application will not work "out of the box". Instead, you will have one hour to get an initial infrastructure rolled out. One hour from the start time of the event, your application will start receiving requests. If you do not have a solution started that functions, you will lose points. 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.

Five hours into the competition we will introduce a new challenge for you to complete that is unrelated to your previous architecture. As a solutions architect, you may be required to perform many different types of tasks. This challenge will gauge your ability to change tasks and respond to ambiguity.

Initial state

Upon starting the competition, there is a running AWS EC2 instance in the account. This instance is not currently serving the billing API but does contain a sample script in the user data segment illustrating how the server could be started. The billing API is not currently functional but there is also no traffic being generated. Requests will begin one hour after the start of the competition.

Tasks

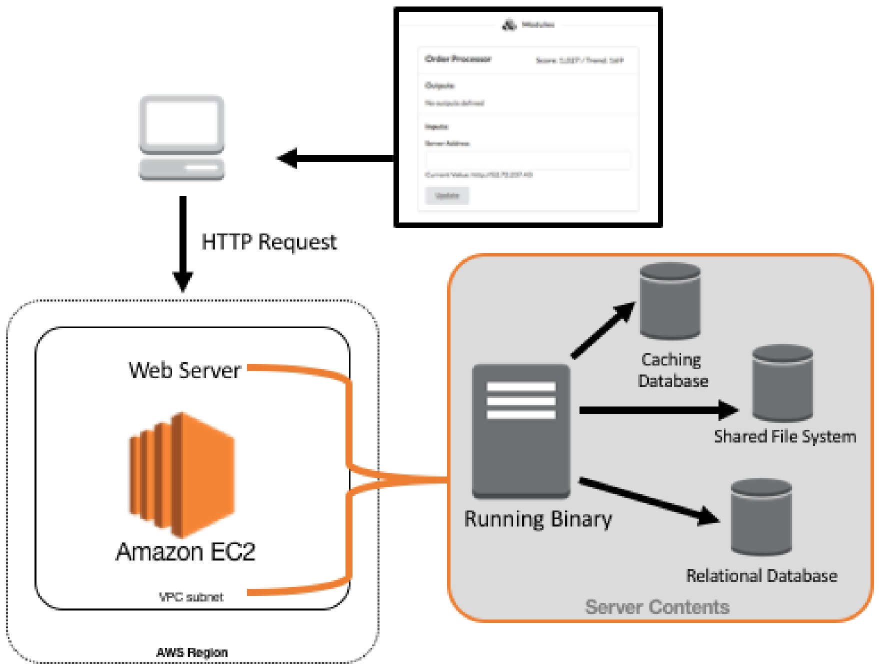
1. Log into Gameday with your assigned hash (Provided on the day)
2. Set your team/competitor name on the Dashboard – (Format: NAME SURNAME)
3. Read the documentation thoroughly (Outlined below)
4. Log into the AWS console (link provided from the Dashboard)
5. Examine existing configurations in EC2 (Elastic 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)
   (a) Create and/or update the user data configuration correctly, including download locations of server binary and server configuration file
   (b) Create and/or update the user data configuration to include any required local dependencies
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 five hours. At this point, a new challenge will appear.
15. Follow the accompanying instructions to complete the module and earn points.
16. 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 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 server application can handle about five connections before starting to get really slow. Be careful about overloading and watch for HTTP 503 responses from the server when the queue fills.
3. The server application is an x86 statically linked, unstripped ELF executable found via an S3 URL provided during the test project.
4. The 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 Redis server.
   (d) It must have access to a running MySQL server with the table structure specified.
   (e) It must have access to an area to store cached files.
   (f) 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.

5. This server requires a connection to a Redis database, a MySQL database, and a location to store data cache files. As with the previous deployment, you can install a Redis server, a MySQL server, and a file storage directory on each instance that has the application deployed but that will be far less efficient than creating a centralized solution. The more efficient you run the infrastructure, the faster your servers will respond to requests and the more points you will earn.
6. The base OS that has been chosen is **Amazon Linux** (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.
7. 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 combination of Redis, MySQL, and file storage. 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.

MySQ

You will need to deploy a MySQL server that is highly available and centralized. This is not an I/O intensive workload and should be deployed on a T2 series instance profile. While the application will function with a MySQL server deployed on each instance running the application, you will not benefit from previous instance processes. Once you have a MySQL service deployed, you will need to create the table necessary to serve the requests. You can use the table definition below.

```
CREATE TABLE unicorns (
unicornid varchar(256),
unicornlocation varchar(256)
);
```

This example creates a table called unicorns. 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 MySQL is shown below:

```
"MysqlHost" = "127.0.0.1"
"MysqlPort" = "3306"
"MysqlUser" = "username"
"MysqlPass" = "password"
"MysqlDb" = "database"
```

MysqlHost is the hostname used to connect to the MySQL service.

MysqlPort is the TCP port number used to connect to the service (default is 3306).

MysqlUser is the username used to connect to the service. It is NOT a best practice to use the root username.

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

Note: The table name is set to "unicorns" in the application and cannot be changed. You will need to create the table as defined in the example.

Redis

You will want to deploy a centralized Redis service for the same reasons as you want to create a centralized MySQL service, for efficiency. Just as with MySQL, you can create a per-instance deployment of Redis but that will operate slower (fewer responses to requests) than using a centralized solution. Again, this workload is not intensive and thus should be deployed on **T2 Micro series instances**. It should take no more than two Redis instances to meet the needs of our application. Using more than two would indicate an inefficient deployment. To configure your application to use the Redis service that you have deployed, the relevant portions of the "**server.ini**" file are below:

```
"RedisHost" = "127.0.0.1"
"RedisPort" = "6379"
```

RedisHost is the hostname of the Redis service provider.

Red*isPort* is the TCP port number for the service.

File Storage

A central file storage location IS NOT a requirement for the application to operate. The application will serve some requests faster with a centralized file storage solution but competitors should focus on initial application functionality rather than fine tuning application performance. As with the previous service examples, you could look to create a centralized file storage solution to use with your application. You can use a local directory to save the cache files but a shared storage solution or the ability to share files to each instance will allow for faster responses. The relevant line in the "**server.ini**" configuration file is below:

```
"FsPath" = "/path/to/files/"
```

FsPath is the local directory (followed by a "/") on the instance where the server application is able to both read and write to the cache files. This can be a local directory on the server or a remote directory mounted as a local directory on the server.

AWS Jam – Feline Takeover

We host a static website (not on EC2). However, it has been changed to a website about CATS! Someone who likes cats must have done this terrible thing! Can you change it back to the Security Jam website?!? The answer will be in the fixed page.

1. Login to the Jam website
2. Select the Feline Takeover Challenge
3. Input the 'answer' to the challenge in the Jam console

Please refer to the **WS2019_TP53_Main_**document for more details