worldskills

Test Project

Cloud Computing

Day two

Submitted by: Qiang Wang CN

Contents

Contents

This Test Project consists of the following documentation/files:

1. WSC2022SE_TP53_MainDocument_actual_en
2. WSC2022SE_TP53_Day1_actual_en
3. **WSC2022SE_TP53_Day2_actual_en**
4. WSC2022SE_TP53_Day3_actual_en
5. WSC2022SE_TP53_Day4_actual_en

Description of project and tasks

This module is 7.5 hours (until 16:30) - **Day 2: Micro Services On AWS EKS**.
The goal of this game is to architect a public client API web application on EKS.

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 two hours to get an initial infrastructure rolled out. 2 hours from the start time of the event, your application will start receiving requests. You will lose points if the application can't up and running in the given time.

Throughout the day it is your responsibility to respond to any challenges that may present themselves. 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.

Background

Unicorn Service core functions can only run in EKS but in order to make the transition journey smoother, you can build and test it on EC2 or any Linux systems. However, if you run Unicorn Service on EC2 instance and provide its address to customer, it will cause you losing points at some point during the game. Unicorn Service provides root path (/) for health check and won't be affected by the environment where it is running, once you get HTTP 200 response from root path, it attests that your application is running normally.

Unicorn company has strict security policy that doesn't allow SSH access from internet, please use suitable AWS service to access EC2 instances.

The company security team has created several policies allow you to create necessary IAM roles to deploy Unicorn Service but they don't leave any document on this process, you need to figure out how to do it. You can find what role you can create through the policy of Team Role. Security team also requires that all database credential should be automatically rotated every 30 days. The following table describe the IAM policies provided by Security team:

POLICY NAME	PURPOSE
UnicornPolicy	Allow Unicorn Service to run in EKS
AutoScalerPolicy	Allow EKS automatically adjusts the number of nodes in your cluster
ELBControllerPolicy	Required by AWS Load Balancer Controller if you want to use it
EFSPolicy	Allow Unicorn Service to connect EFS

Please check your permission to decide how to create necessary roles.

Security team has a third-party tool which need to access EKS cluster to evaluate its security posture in the future, it's a SaaS product, so you need to design a proper mechanism to protect Kubernetes API endpoint from potential attacks but allow access from internet in the future.

The following software has been used during development; you need to deploy the following application to support Unicorn Service.

- Amazon Web Services Load Balancer Controller

According the feedbacks from UAT team, using two t3.medium instances should be able to serve normal peak traffic when EKS cluster is normal and use instance beyond t3.medium or other instance family may increasing cost significantly. But you need to monitor the traffic closely in case of any unexpected traffic and make sure Unicorn Service is available.

Initial state

Upon starting the competition, there is a running Amazon EC2 instance in the account. The unicorn service is not functional at this stage but there is also no traffic being generated. Requests will begin 2 hours after the start of the competition.

Caution

Please bear in mind that do not terminate the running instance provided by default. Otherwise, you wouldn't be able to create EC2 instance due to unicorn company's security protocol if you deleted it. The default instance act as a bastion server, you should use this instance to accomplish your tasks and make sure that judges can login into this instance and check your system, if you installed any application or tools on this machine, make sure its installation path is added to PATH variables. (There is an EC2 instance profile ready for you to use if you need, the profile would be named like *TeamRoleInstanceProfile*)

Re-platform schedule

Phase I

- Build a high availability, resilient basic infrastructure environment for Unicorn Service, you may need to consider the following AWS Services: VPC, Security Group etc.

Phase II

- Build Unicorn Service, you may need to consider the following AWS Services: ECR, EKS, IAM, RDS etc.

Phase III

- Provide Unicorn Service, you need to provide Access Endpoint to GameDay Dashboard.

Tasks

1. Log into Gameday with your assigned hash (Provided on the day)
2. Set your team/competitor name on the Dashboard
3. Read the documentation thoroughly (Outlined below)
4. Log into the Amazon Web Services console (link provided from the Dashboard)
5. Check your own permission before choosing tools to accomplish your tasks
6. Examine existing configurations in EC2 (Elastic Cloud Computer server)
7. Examine existing configurations in VPC (Virtual Private Cloud, Network Segment)
8. Create a EKS Cluster through AWS Console, two IAM roles named EKSClusterRole and EKSNodeRole have been created for you to run a functional EKS Cluster, please check the permissions of those roles to decide how to use it.
9. Deploy application into EKS to handle increasing load
10. Configure any server dependencies as outlined in the technical details
11. Monitor performance of the application servers in the "Score Events and Scoreboard" and through the AWS Console with CloudWatch
12. Serve client requests to gain points, reference the "Score Events and Scoreboard" to ensure you are scoring positively by serving the requests
13. Process exceptions when they are received, reference the "Request Exception Handling" below
14. This section of the Test Project will continue for 7.5 hours.

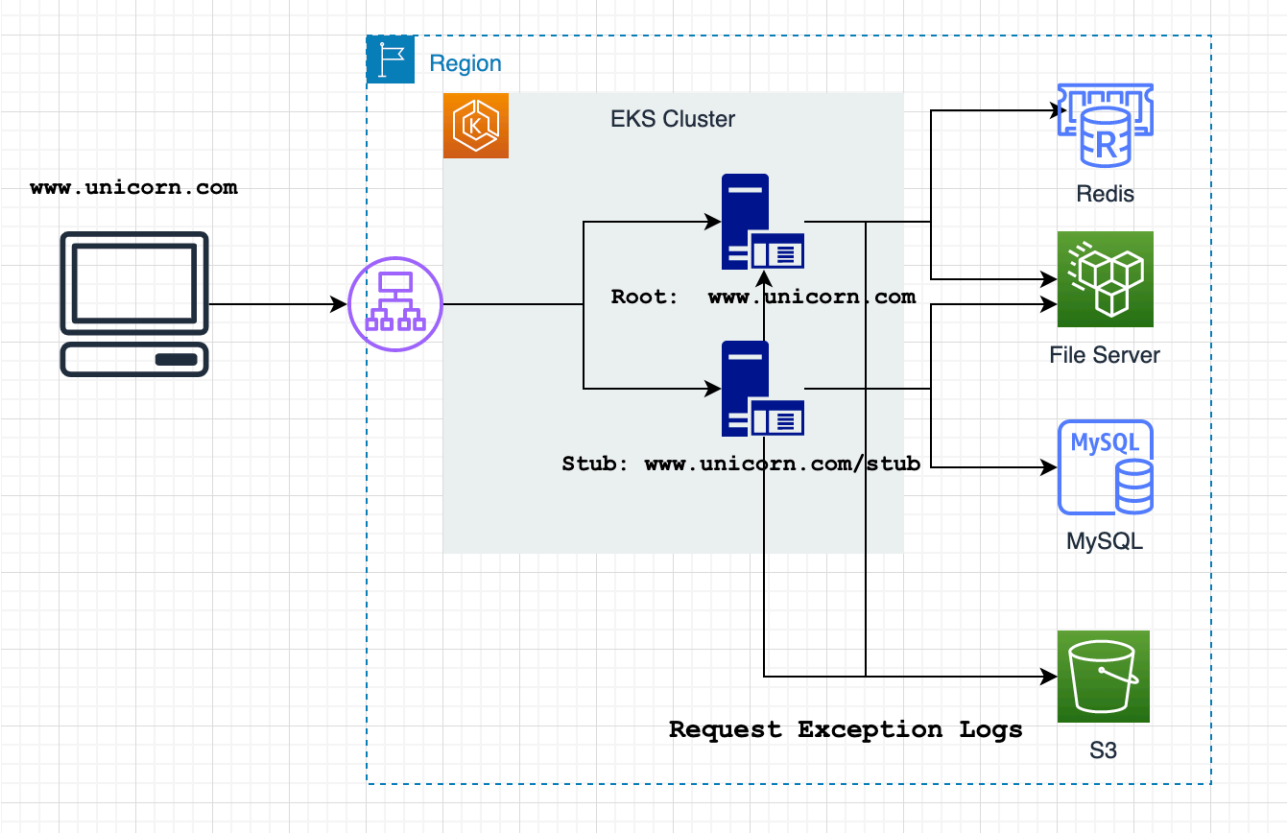
Technical Details

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. For this day of competition focus on EKS as your compute resource. Do not use ECS or Fargate.
3. The server application can handle about 5 connections before starting to get really slow. Be careful about overloading and watch for HTTP 503 responses from the server when the queue fills.
4. The server application is x86 statically linked, unstripped ELF executable binary, both root and stub application binaries can be found in Readme file of this game event.
5. 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 from AWS AppConfig service.
   (c) It must have access to a running Redis (ElasticCache)server.
   (d) It must have access to a running MySQL(RDS PostgreSQL) server with the table structure specified.
   (e) It must have access to an area to store cached files. (EFS)

6. This server requires a connection to a Redis database, a MySQL database, and EFS. 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.
7. 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.
8. 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.

The development team has adopted AWS AppConfig Service as a configuration center, all unicorn applications need to communicate to AWS AppConfig Service in order to get configuration. The format of configuration can be found at **Example Configuration** part of this section.

As to how can those application be started, you can run the binary with the --help parameter.

MySQL (RDS)

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 t3.medium 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 a database called **unicorndb**

and 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 above creates a table called unicorns, your table name must be unicorns 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 (ElasticCache)

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 t3.medium 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.

Please note that the application currently is not work with Redis Cluster.

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.

```
"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.

Example Configuration

Stub Server Configuration is shown as follow:

```
{
"DBSecretArn": "arn:aws:secretsmanager:region:accountid:secret:prod-7aKIJz",
"DBName": "unicorn",
"FsPath": "./",
"Bucket": "unicorn-us-es-ws-1903",
"Port": 80
}
```

DBSecretArn is the ARN of secret used to maintain MySQL password rotation.

DBName 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.
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.

Bucket is the S3 bucket used to store refund records.

Port is the TCP port number on which Stub server is listening

Root Server Configuration is shown as follow:

```
{
  "RedisHost": "wschina.hqvb4d.ng.0001.use1.cache.amazonaws.com",
  "RedisPort": "6379",
  "FsPath": "./",
  "Port": 80,
  "Bucket": "unicorn-us-es-ws-1903"
}
```

RedisHost is the hostname of the Redis service provider.

RedisPort is the TCP port number for the service.

Bucket is the S3 bucket used to store refund records.
Port is the TCP port number on which Root server is listening

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.

REQUEST EXCEPTION HANDLING

Throughout the time your web applications are running, you will occasionally receive an exceptional request indicating that you need to cancel the relevant order and make a refund. The development team has decided to keep all refund requests in S3 bucket, an example content of this object is shown below:

```
c4a5010e-6734-41e8-974b-59af1bd55ca0
```

Throughout the day at random intervals, you will receive a request similar to the one identified above. Each request will have a string associated in a UUID format. Each of these requests identifies a customer request for a refund on a Unicorn Rental purchase. Each of these requests must be sent to the endpoint provided. Every time one of these requests is received, you should send it to the audit system using an HTTP POST call and the tool of your choice. The example below demonstrates using the utility "curl" on the Linux command line:

```
curl -i -H "Accept: application/json" -X POST -d '{"game": "GAME_ID", "team":"TEAM_ID", "order":"REQUEST_UUID"}' https://stats.aws.dev-null.link/proc/refund
```

There is also a script that can be used, please refer to download link provided in Readme file providing to you. GAME_ID is alias of Event ID which you can find on your team dashboard.

Reference

1. Kubernetes Deployment: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
2. Kubernetes Service: https://kubernetes.io/docs/concepts/services-networking/service/
3. Kubernetes Ingress: https://kubernetes.io/docs/concepts/services-networking/ingress/
4. Amazon Web Services Load Balancer Controller: https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html
5. Amazon Web Services Load Balancer Controller Guide: https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.3/ , Please note that this guide may contains outdated information, all information related to Kubernetes should align with Kubernetes official documents.