DESCRIPTION OF PROJECT AND TASKS

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 'python' 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 need to configure the services that the server depends on. When you prepare your website and submit the URL, the request will be sent to your website.

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.

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, 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

Requests will begin after the start of the competition.

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.

© 2024 Amazon Connect Technology Services (Beijing) Co., Ltd. or its affiliates. All Rights Reserved.

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 on Cloud Raiser.

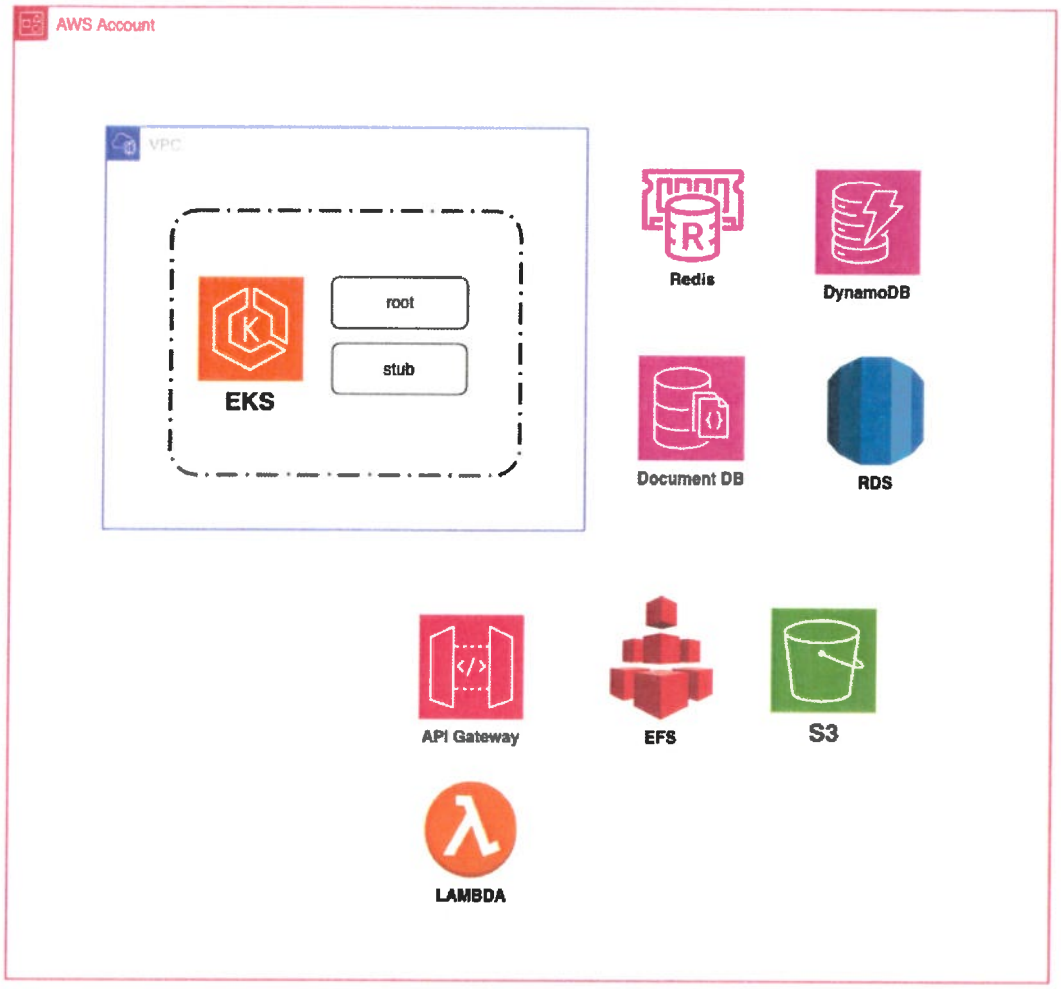
TASKS

1. Log into Cloud Raiser.
2. Read the documentation thoroughly (Outlined below)
3. Log into the Amazon Web Services console (link provided from the Dashboard)
4. Check your own permission before choosing tools to accomplish your tasks
5. Examine existing configurations in VPC (Virtual Private Cloud, Network Segment)
6. Create a EKS Cluster through AWS Console
7. Deploy application into EKS to handle increasing load
8. Configure any server dependencies as outlined in the technical details
9. Monitor performance of the application servers in the "Score Events and Scoreboard" and through the AWS Console with CloudWatch
10. Serve client requests to gain points, reference the "Score Events and Scoreboard" to ensure you are scoring positively by serving the requests
11. Please do not trust any external accounts/resources for the IAM users/roles in your account

TECHNICAL DETAILS

1. The server application is deployed as a Python 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. Be careful about overloading and watch for HTTP 503 responses from the server when the queue fills.
3. The server application is x86 statically linked, unstripped ELF executable binary.
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 access to a running DynamoDB.
    c. It must have access to a running DocumentDB.
    d. It must have access to a running Redis (ElasticCache)server.
    e. It must have access to a running RDS(MySQL) server with the table structure specified.
    f. It must have access to an area to store cached files. (EFS)
5. 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 Linux2** (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.Architecture

© 2024 Amazon Connect Technology Services (Beijing) Co., Ltd. or its affiliates. All Rights Reserved.



The above example illustrates one possible architectural design for the deployment of the application. It is for reference only.

SERVICE DETAILS

Overview

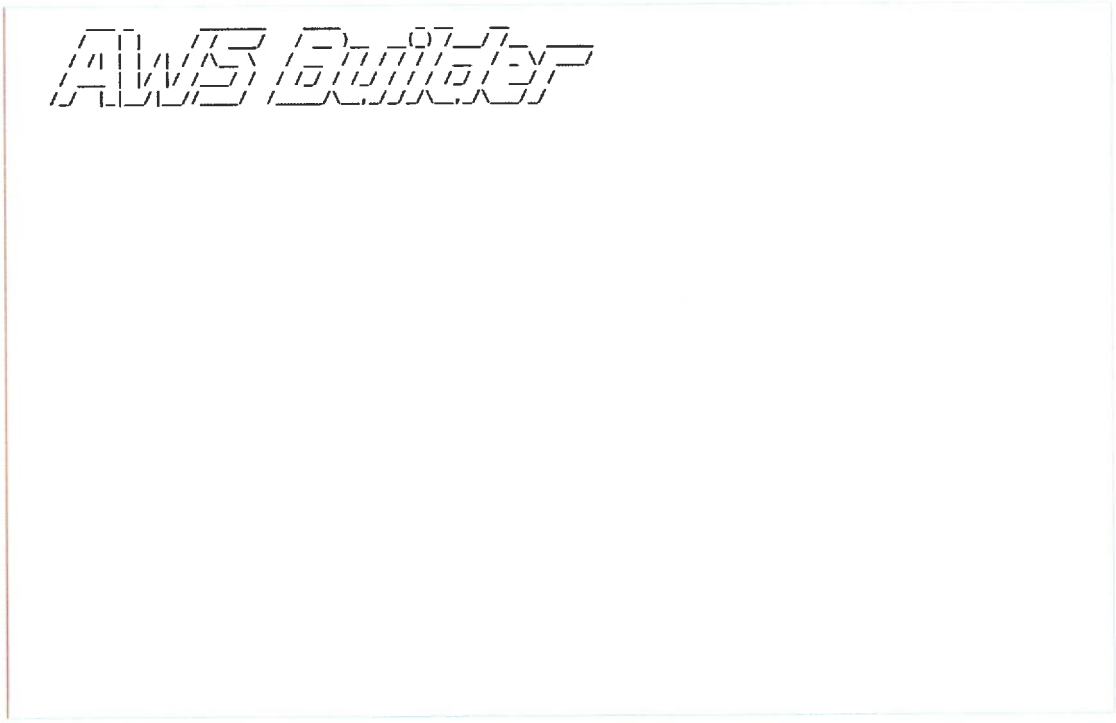
Every time a request is sent to a server that you have deployed, the process must generate a response to send out (very slow). However, once you have answered a response, the data necessary to respond is stored in a combination of DB and file storage. If you received a request and have already answered 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 format of configuration can be found at **Example Configuration** part of this section.

URL

You will need to provide a single URL as the answer, which will be used to access your two applications. To facilitate this operation, you need to perform URL routing. Therefore, if deployed properly, when you access the public URL of the application, you should see the following screen:

© 2024 Amazon Connect Technology Services (Beijing) Co., Ltd. or its affiliates. All Rights Reserved.



Application address description (if your URL is http://server.example.com):

- Home page address: http://server.example.com/
- The request address to be sent to the Stub application: http://server.example.com/root
- The request address to be sent to the Stub application: http://server.example.com/stub

Root&Stub Application - EFS Path

You need to provide an EFS path for the application.

Root application - Redis database

You need to provide a Redis database solution for the application.

Root application - DynamoDB database

You need to provide a DynamoDB database solution for applications.

KEY of table: id (String)

Stub application - DocumentDB database

You need to provide a DocumentDB database solution for applications.

Root&Stub application - S3 bucket

You need to provide an S3 storage solution for the application.

Stub application - RDS (MySQL engine)

© 2024 Amazon Connect Technology Services (Beijing) Co., Ltd. or its affiliates. All Rights Reserved.

You need to provide an RDS database solution for applications. This is not an I/O intensive workload and should be deployed in the most cost-effective way possible. After deploying the RDS service, you need to create the necessary tables to provide the service for the request. A table must be created to store information for service requests. You can find table definition examples through the database.sql file.

Root&Stub Application - API Gateway + Lambda

You only need to set your Serverless service to return HTTP code 200, no additional operations are required

Example Configuration

```
{
  "efs_path": "/mnt/efs",
  "db_config": {
    "host": "cloudraiser.cs2h4ra5vwuj.us-east-1.rds.amazonaws.com",
    "user": "cloudraiser",
    "password": "cloudraiser",
    "database": "cloudraiser"
  },
  "documentdb_config": {
    "host": "cloudraiser1.cluster-cs2h4ra5vwuj.us-east-1.docdb.amazonaws.com",
    "name": "cloudraiser",
    "password": "cloudraiser",
    "port": 27017
  },
  "dynamodb_config": {
     "table": "cloudraiser"
   },
  "redis_config": {
    "host": "cloudraiser.jndheo.ng.0001.use1.cache.amazonaws.com",
    "port": 6379
  },
  "s3_config": {
    "bucket_name": "cloudraiser-657805226609"
  },
  "api_gateway_config": {
    "api_gateway_url": "cloudraiser",
    "api_key":"cloudraiser"
  }
}
```


© 2024 Amazon Connect Technology Services (Beijing) Co., Ltd. or its affiliates. All Rights Reserved.

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.

© 2024 Amazon Connect Technology Services (Beijing) Co., Ltd. or its affiliates. All Rights Reserved.