Architecting Enterprise Cloud Solutions Assignment

* **DEVELOPER IQ**

Name : Thirunayan Dinesh Jeeva

Student No: MS20908188

### COMPLETE BEFORE 7/5/2021 ###

Contents

1. Introduction
   1. What is DeveloperIQ
   2. Definition of developer productivity
2. Productivity Calculation Equation
   1. Explanation of productivity equation
3. Micro service architecture
   1. Architecture diagram of microservices
   2. Usage of CQRS Pattern
   3. Architecture diagrams of microservices and functionality of each service.
      1. Productivity Calculation Services
      2. Database upload service
      3. Database update check service
4. Database
   1. Reason for choosing DynamoDB (A no-sql database)
   2. DB Structure
5. AWS deployment structure diagram. Along with control EC2 instance
6. Containerization
   1. Reason to containerize
   2. Cloud container registry and why choose AWS ECR
7. Cluster Orchestration
   1. Kubernetes deployments
   2. Kubernetes services
   3. EKS Cluster Configuration
   4. Node group configuration
   5. EKS Cluster Architecture
8. Using AWS CloudFormation
   1. Purpose of using cloud formation
   2. Cloud formation script
9. Conclusion

Introduction

Developer IQ is a developer productivity calculation application which can quantitatively calculate the productivity of a developer on a particular project by tracking the developer’s activity on GitHub such as commits, pull requests, issues created and issue comment interactions. These activity metrices are then used to calculate an aggregated DeveloperIQ productivity score for the particular developer, for a “weekly”, “monthly” and “yearly” timeframe.

Calculating Developer Productivity

Calculating an aggregated developer productivity score quantitatively is a complex task since, productivity itself is an abstract concept with multiple definition, in the case of DeveloperIQ we consider developer productivity as the total contribution of the developer to the project repository. Thus, our mathematical formulation for calculating productivity also considers the indirect contributions of a developer like there activeness for responding to issues, issues created along with direct contribution like commit frequency ,commit addition and deletion.

The DeveloperIQ productivity metric

Developer Productivity :=

commit addition : source code addition volume in commits

commit deletions : source code deletion volume in commits

number of commits : number of commits the developer has made for a particular time frame.

Issues created : Issues created by the developer this includes the pull requests and common bug issues as well

Issues comments: Number of times the developer has interacted with open issues, this includes code reviews and answers to open bug issues.

The total commit addition and commit deletion is added to get an aggregate score of the total code contribution which is then divided by the number of commits. The developers are penalized for a high number of commits with low contributions, because the standard best practice is to push meaningful commits with sufficient contributions and functionalities instead smaller commits without any significant additions or deletions. The calculated score is called the **commit score.** The commit score is then added to the sum of the number of issues created by the developer and issue interactions by the developer, this is called the **interaction score**. The **commit score** and **interaction score** are added and are passed into the *log* function to normalize the outputs within a much shorter range.

The output will be the productivity score of the developer, and higher DeveloperIQ scores denote higher developer productivity.

Microservice Architecture

A completely cloud based microservices architecture is used for the implementation of the project.

We use 3 core services :

1. Productivity Calculation Service
2. DB Update Check Service
3. DB Update Service

Productivity Calculation Service : This service fetches data from the AWS DynamoDB cache database consisting of the developer productivity scores for each contributor of a repository and then calculates the DeveloperIQ productivity score for each contributor using the above equation. This is the service that the frontend will directly interact with to get the developer productivity metrics.

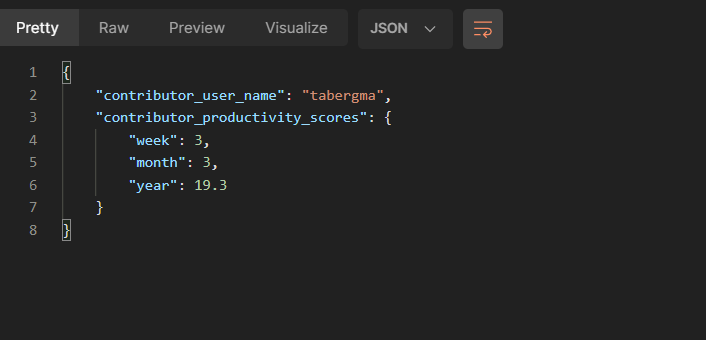


Figure - Productivity calculation service output

The service provides the contributor metrics in 3 time frames ,”week”, “month” and “year”.

**DB Update Check Service :** This service calls the github api in a consistent time loop, to look for changes in contributor activity by comparing the extracted productivity metrics with metrics stored in cache DB, and if there are any changes this service sends a request to the DB Update Service which updates the caching database

**DB Update Service :** This service updates the database whenever a change is reported by the update check service.

We used the CQRS (Command Query Segregation Pattern) when developing these microservices so that the read service and write service can be independently scaled there is a good separations of concerns enabling simpler queries to the database.

Database

We utilize a database for caching so that for every request the api does not have to directly call the GitHub API. The database was implemented using AWS DynamoDB which is a no-sql database. A no-sql database was chosen because it enabled high speed read-write operations along with un-structured schemas which allowed flexibility in the structure of the contributor data storage format.

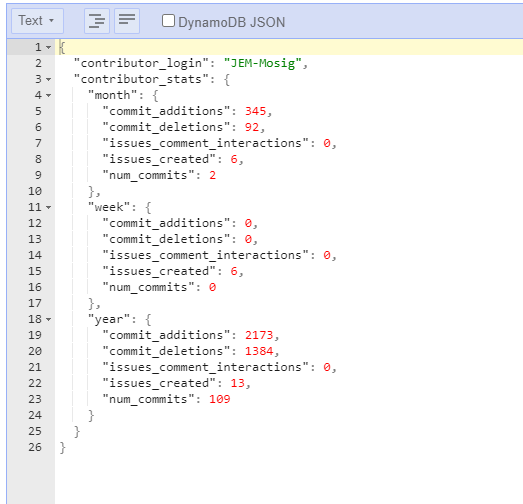


Figure 2 - A sample user's metrics stored in the dynamo db database

The database update is executed by the DBUpdate Service which routinely updates the database if there are any changes in contributor activity.

AWS

In this project we use amazon web services (AWS) for completely to develop the backend infrastructure. The services in AWS utilized for this in this project are :

1. AWS EC2 - For cluster deployment and administration
2. AWS ECR - Used as a registry for the storing and versioning of containers
3. AWS EKS - Used to deploy and manage the Kubernetes cluster and master control plane.
4. AWS DynamoDB – For cache storage and retrieval
5. AWS IAM – For access management for nodegroups within the cluster
6. AWS VPC – For secure private cloud networking within the cluster
7. AWS Cloudformation – For the automation of configuration and deployment of the entire cluster infrastructure

## Insert AWS Architecture Diagram ##

Containerization

All the 3 services use the Dockerfile configuration since the packages and environment needed to run the 3 services are identical. After a code change or environment change is made to the codebase, the container would be built again and then tagged with an incremental version number and then pushed to the repository. The complete container workflow is depicted below.

After a container is pushed to ECR registry, a Kubernetes deployment is rolled out and the existing services and deployments are restarted to work with the updated code base and containers, this is done without a down time since all the Kubernetes services use a load balancer to expose the deployments. The complete container deployment pipeline is presented in the diagram below.

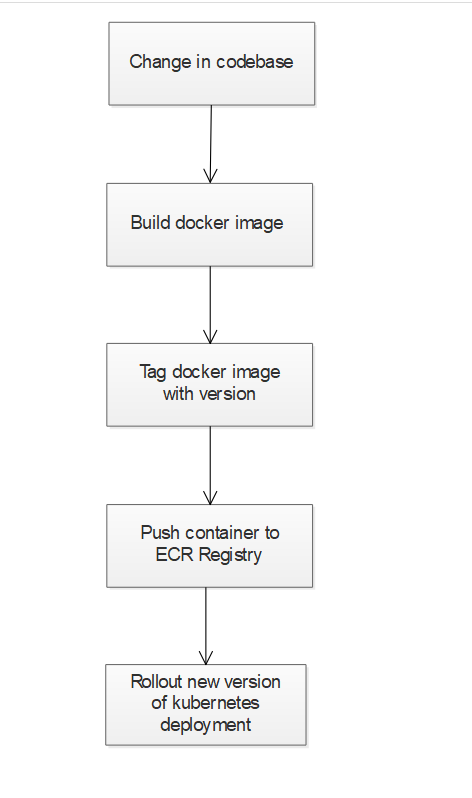


Figure 3 - Container deployment pipeline

EKS Cluster and Kubernetes Orchestration

## Insert details about number of pods and cluster configuration yaml files, services and deployments ##

Kubernetes was used for the orchestration of the deployed containers in this project and AWS EKS was used to manage the Kubernetes master control plane.

There were 2 deployments created for the 3 services. The productivity calculation service was deployed in a separate deployment whereas the github repository monitoring service and database update service was contained in a single deployment because both the services needs to communicate frequently and aggregately serve the same purpose of caching.

In the EKS cluster, a nodegroup consisting of two t3 medium instances were used, t3 medium was chosen as the node compute category since running Kubernetes requires a moderate level of memory. All the nodes were made SSH accessible so that each of the nodes can be accessed by the user and each deployments were configured to have 2 replicas running on each of the nodes, so that even in the case that a single node fails the services can continue to run.

The number of nodes in the cluster were configured to be 2 because of the cost factors associated with creating nodes.

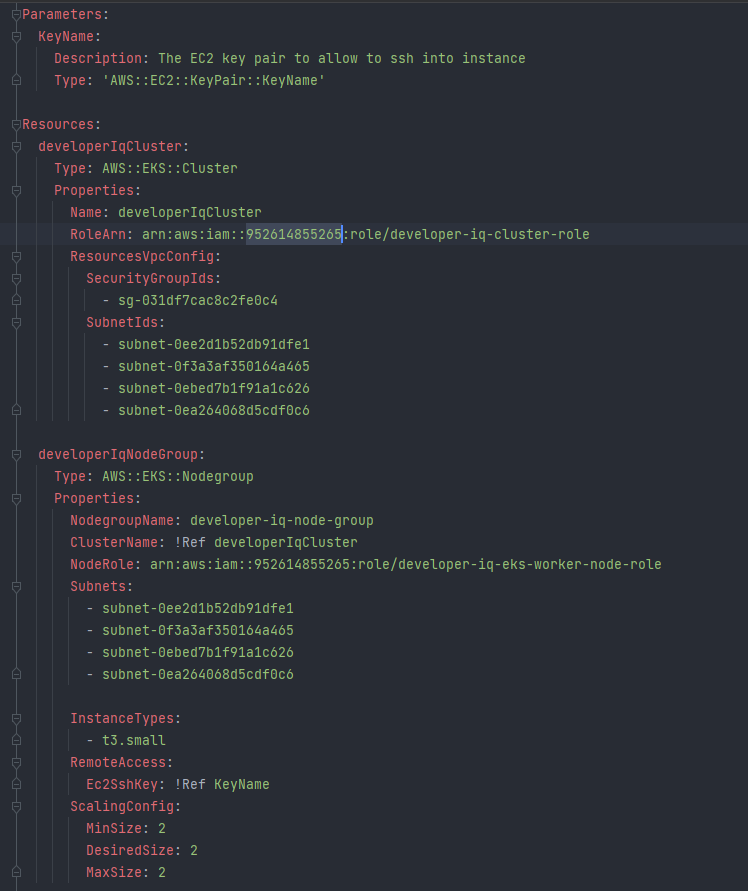
## Show nodegroup diagram.

## Show EKS Cluster Setup##

Cloud formation

The deployment of a complete EKS cluster with nodegroup and VPC configurations can take a lot of time, to avoid this delay and to make cluster configuration and infrastructure replication and versioning more seamless, AWS cloud formation is used to script the complete infrastructure configuration and then create all the components. We use two CloudFormation files, one for VPC configuration and another for cluster configuration. The reason for using two files is so that the VPC configuration is already set up and running even before the cluster build starts. The content of both configuration yaml files are provided below.

Figure - EKS Cluster Formation Configuration

**AWS EKS Cluster Configuration**

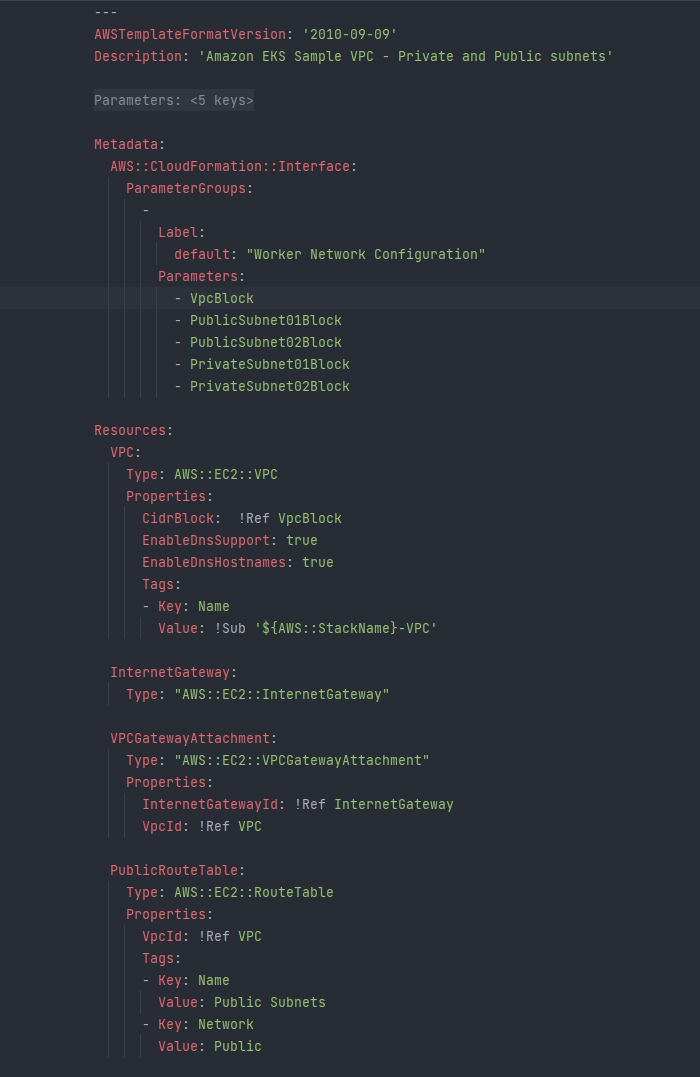
**AWS VPC Configuration**

Figure - AWS VPC Configuration

Fault Scenarios and Fault Tolerance Mechanisms

## Insert scenarios where the application can fail and how fault tolerancy is implemented

**Scenario 1: Node Failure**

**Fault tolerance approach:** Kubernetes Pod Management

**Scenario 2 : Service Failure**

**Fault tolerance approach :**  Kubernetes Replicas

**Scenario 3: Unauthorized Access**

**Fault tolerance approach:** IAM roles

**Scenario 4 : Interservice communication intercepting**

**Fault tolerance approach :** VPC Communication

Conclusion

## Insert conclusion about what aspects can be improved in developer IQ.

Improvement points : Can automate container deployment and ECR key validation using Jenkins and Github actions.