

# Architecting Enterprise Cloud Solutions Assignment

## - DEVELOPER IQ

Name : Thirunayan Dinesh Jeeva

Student No: MS20908188

GitHub: <https://github.com/Thirunayan22/DeveloperIQ>

# Contents

Introduction .....	3
Calculating Developer Productivity .....	3
The DeveloperIQ productivity metric.....	3
Microservice Architecture.....	4
Cloud Architecture Pattern .....	7
Database .....	7
AWS Infrastructure .....	8
Containerization.....	9
EKS Cluster and Kubernetes Orchestration .....	11
Deployment Diagram .....	22
Component Diagram.....	22
Fault Scenarios and Fault Tolerance Mechanisms.....	23
Conclusion .....	23

## 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 metrics are then used to calculate an aggregated DeveloperIQ productivity score for the 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 definitions, 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 direct and indirect contributions of a developer like their activeness for responding to issues, issues created along with direct contribution factors like commit frequency, commit addition and deletion.

## The DeveloperIQ productivity metric

Developer Productivity :=

$$\log\left[\frac{\text{commit additions} + \text{commit deletions}}{\text{number of commits}} + (\text{issues created} + \text{issues comments})\right]$$

**Commit addition:** source code addition volume in commits

**Commit deletions:** source code deletion volume in commits

**Number of commits:** total 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 as number of times the developer has interacted with open issues, this includes pull request 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 productivity of the developer is evaluated to be directly proportional to the DeveloperIQ productivity score.

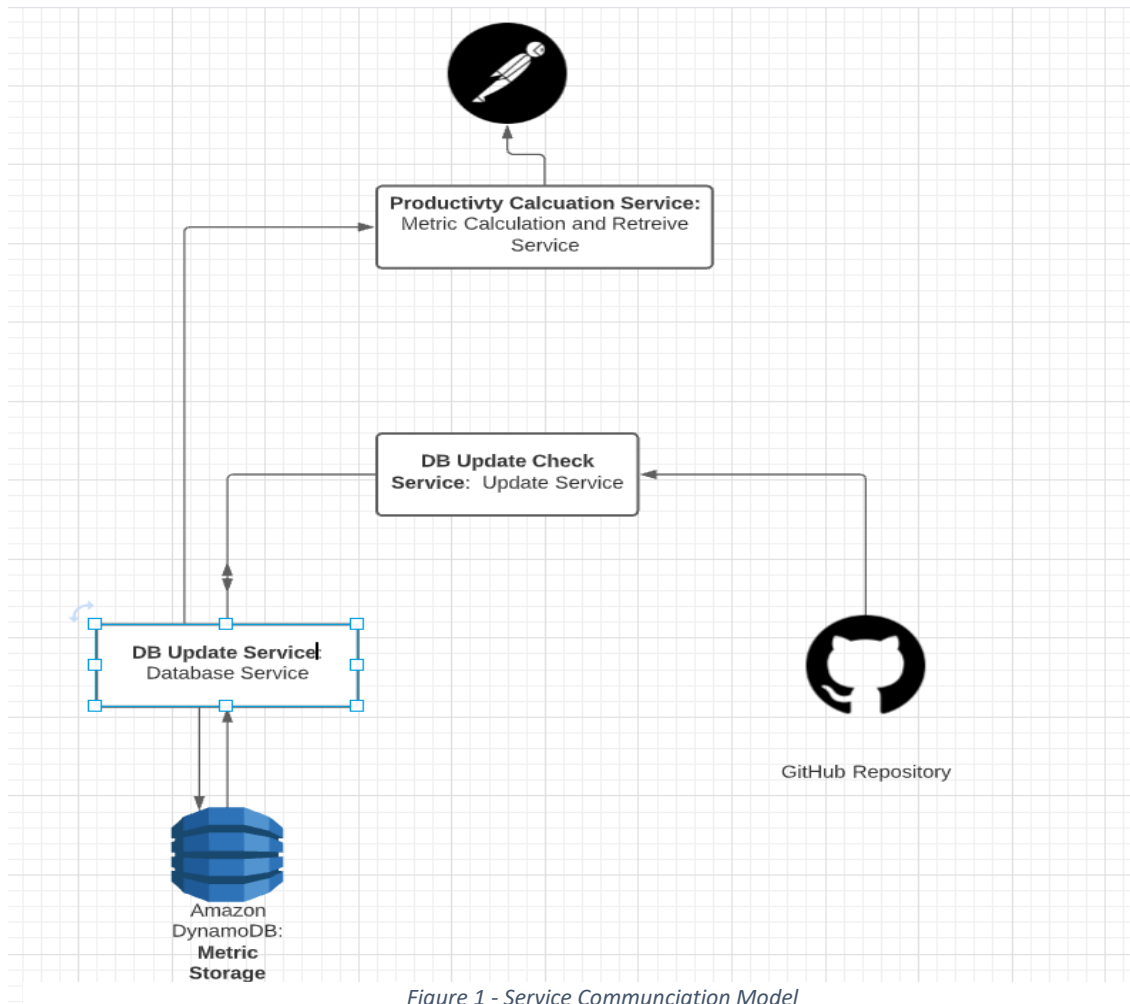
### Microservice Architecture

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

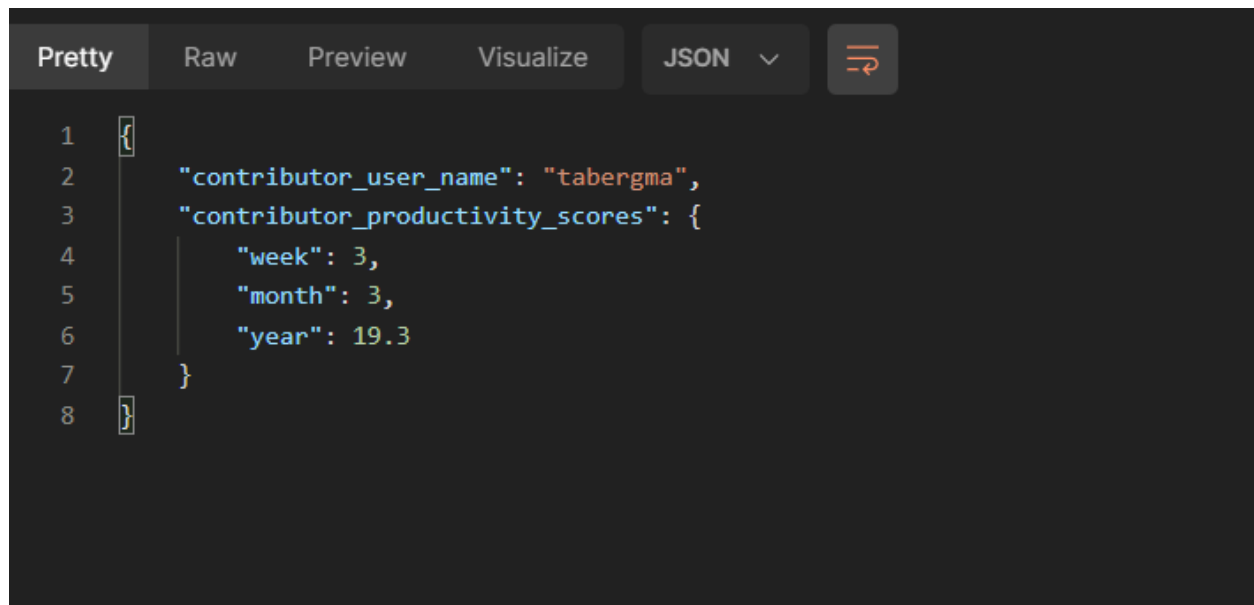
We use 3 core services:

- 1) Productivity Calculation Service
- 2) DB Update Check Service
- 3) DB Update Service

A diagram depicting the inter-service communication is given below:



**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 equation above. This is the service that the frontend will directly interact with to get the developer productivity metrics.

A screenshot of a code editor with a dark theme. At the top, there are tabs labeled 'Pretty', 'Raw', 'Preview', and 'Visualize'. To the right of these tabs is a dropdown menu currently set to 'JSON' with a downward arrow, and a button with a red icon. The main area of the editor displays a JSON object. On the left side of the editor, line numbers 1 through 8 are visible. The JSON object is as follows:

```
1 {  
2   "contributor_user_name": "tabergma",  
3   "contributor_productivity_scores": {  
4     "week": 3,  
5     "month": 3,  
6     "year": 19.3  
7   }  
8 }
```

*Figure 2 - Productivity calculation service output*

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

**DB Update Check Service:** This service calls the GitHub API and gets the real-time productivity metrics for a particular contributor by making multiple requests to the different endpoints which to extract metric values.

**DB Update Service:** This service is responsible for updating the caching database. The DB update service calls the DB update check service in a regular time frame to look for changes in contributor activity in the repository and update the changes to the cache db. The service takes a live snapshot of the cache database and another live snapshot from GitHub to get the contributor’s real-time metrics, and then compares both the snapshots and if it detects any difference, the service updates cache database with the latest snapshot from GitHub.

This routine check is run every 1 hour within an additional “delay” time which can be controlled as a parameter. This type of a batch-based cache update mechanism is used because GitHub imposes a limit of 5000 API requests per hour for an authenticated user, if this limit is exceeded the requests are bounced back and rejected.

### Cloud Architecture Pattern

We used CQRS (Command Query Segregation Pattern) as the base pattern to extract and update the database. The CQRS pattern proposes the idea of separating read and update functions into two different domains.

The core reason for the choice of this cloud design pattern is because this application contains a lot of frequent writes and the volume of reads can vary depending on the number of requests. The CQRS pattern enables us to independently scale the read and write services depending on the traffic, in addition it also provides a better separation of concerns thus 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 key-value store, 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.

```
1 {
2   "contributor_login": "JEM-Mosig",
3   "contributor_stats": {
4     "month": {
5       "commit_additions": 345,
6       "commit_deletions": 92,
7       "issues_comment_interactions": 0,
8       "issues_created": 6,
9       "num_commits": 2
10    },
11    "week": {
12      "commit_additions": 0,
13      "commit_deletions": 0,
14      "issues_comment_interactions": 0,
15      "issues_created": 6,
16      "num_commits": 0
17    },
18    "year": {
19      "commit_additions": 2173,
20      "commit_deletions": 1384,
21      "issues_comment_interactions": 0,
22      "issues_created": 13,
23      "num_commits": 109
24    }
25  }
26 }
```

Figure 3 - A sample user's metrics stored in the dynamo db database.

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

### AWS Infrastructure

In this project we use amazon web services (AWS) 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 the containers.
3. AWS EKS - Used to deploy and manage the Kubernetes cluster and master control plane.
4. AWS DynamoDB – For cache storage and retrieval

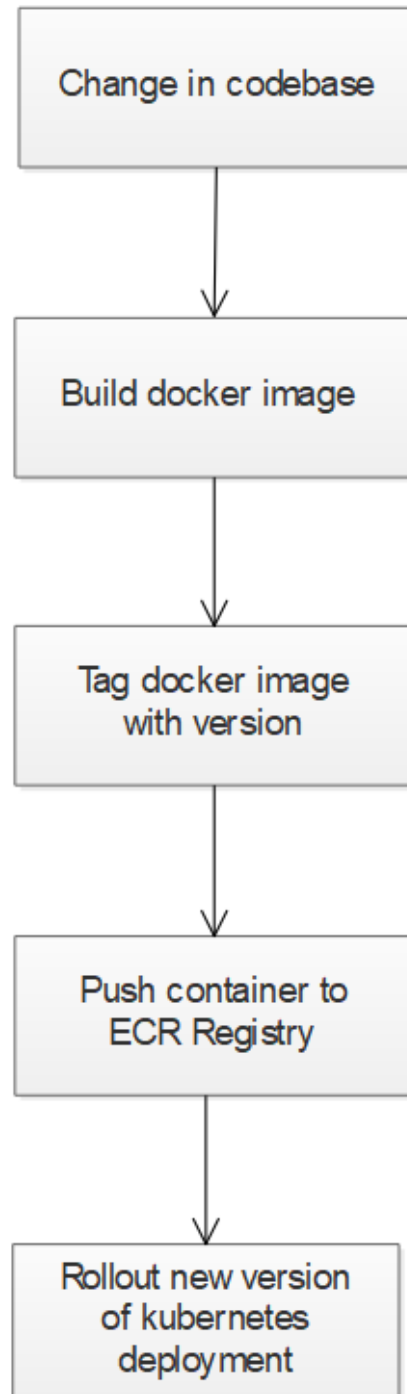


5. AWS IAM – For access management for node-groups within the cluster
6. AWS VPC – For secure private cloud networking within the cluster
7. AWS CloudFormation – For the automation of configuration and deployment of the entire cluster infrastructure

### 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 in the diagram below.

After a container is pushed to ECR registry, a Kubernetes deployment is rolled out and the existing pods 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 4 - Container deployment pipeline*

### EKS Cluster and Kubernetes Orchestration

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 were contained in a single deployment because both the services need to communicate frequently and aggregately serve the same purpose of caching.

In the EKS cluster, a node-group 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 deployment 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.

The complete cluster infrastructure is wrapped with an AWS VPC (Virtual Private Cloud) network to enable secure inter-communication within the services as well as communication with external services.

The deployment.yaml file for the productivity calculation service is given below.

```
apiVersion: v1
kind: Service
metadata:
  name: developeriq-productivity-calc-service
spec:
  selector:
    app: developeriq-productivity-calc
  ports:
  - name: http
    protocol: "TCP"
    port: 7000 #Port which kubernetes service runs on
    targetPort: 8002
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: developeriq-productivity-calc
spec:
  selector:
    matchLabels:
      app: developeriq-productivity-calc
  replicas: 2
  template:
    metadata:
      labels:
        app: developeriq-productivity-calc
    spec:
      imagePullSecrets:
      - name: developeriq-registry-key
      containers:
      - name: developeriq-productivity-calc
        image: 952614855265.dkr.ecr.us-east-1.amazonaws.com/developer-iq:1.3
        command: ["python3", "ProductivityCalculationService/productivity_calculation_service.py"]
        imagePullPolicy: Always
        ports:
        - containerPort: 8002
```

Figure 5 - Deployment file of productivity calculation service

The EKS cluster deployment diagram with VPC configuration is visualized in the cluster deployment architecture diagram below.

### AWS EKS Cluster Configuration With VPC Configured

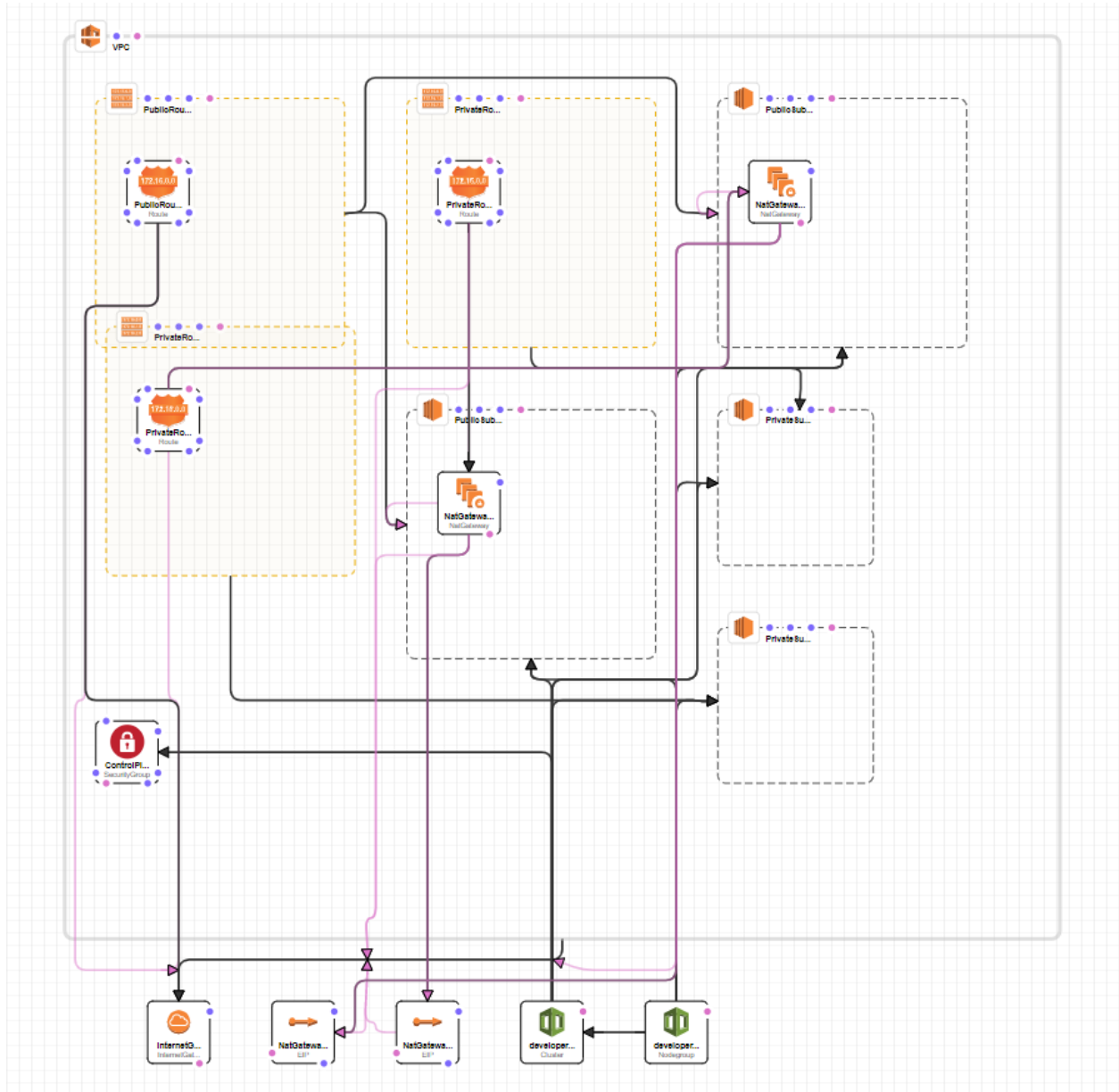
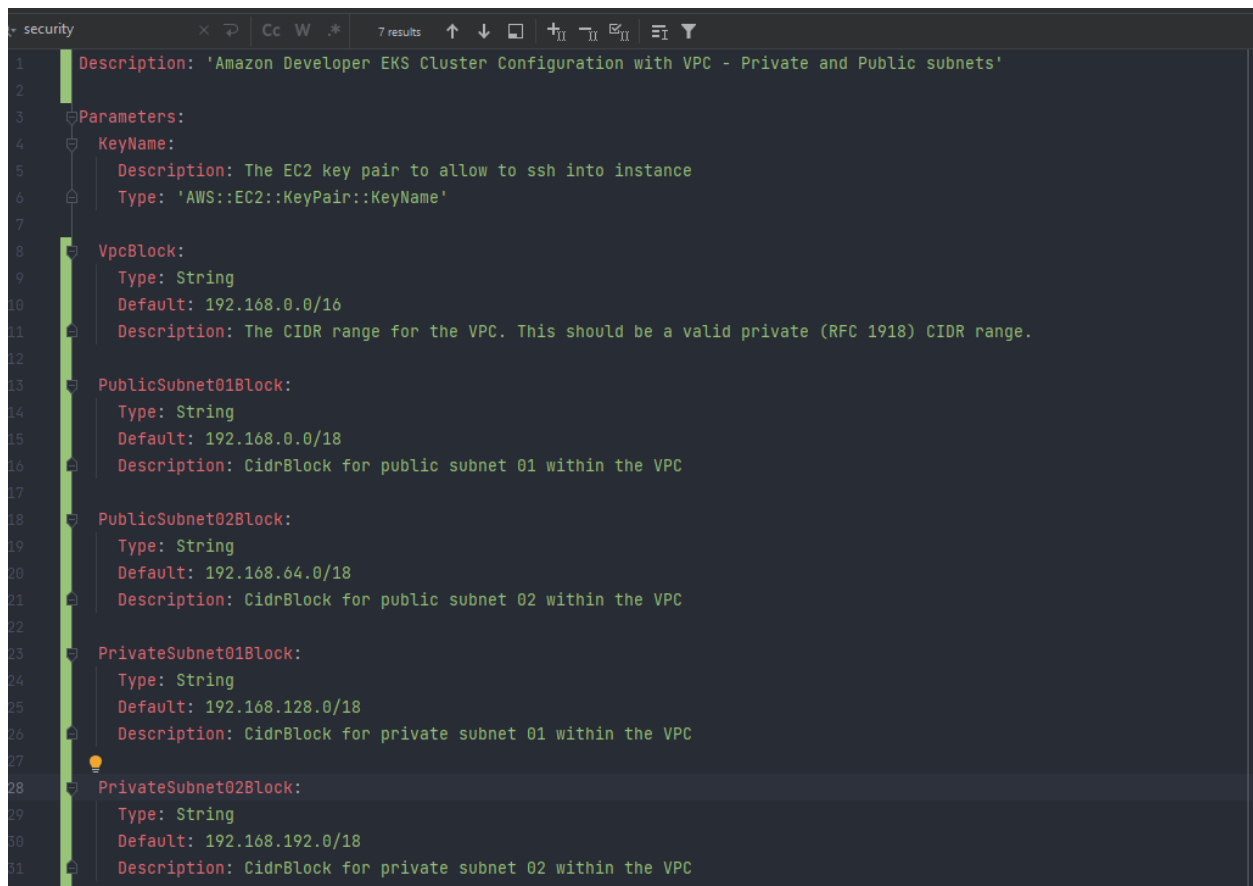


Figure 6 – AWS EKS Cluster with VPC Configuration

## Cloud formation

The deployment of a complete EKS cluster with node-group and VPC configurations can take a lot of time, to avoid this delay and to make infrastructure replication and versioning more seamless, AWS cloud formation is used to script the complete infrastructure configuration and then create all the components. We configure both the VPC and EKS Cluster infrastructure in a single CloudFormation yaml file.

A screenshot of a code editor showing a CloudFormation script in YAML format. The editor has a dark theme and a sidebar on the left with a file explorer. The main area displays the 'Parameters' section of the script. The script is titled 'Amazon Developer EKS Cluster Configuration with VPC - Private and Public subnets'. It defines several parameters: 'KeyName' (EC2 key pair), 'VpcBlock' (VPC CIDR), 'PublicSubnet01Block' (public subnet 01 CIDR), 'PublicSubnet02Block' (public subnet 02 CIDR), 'PrivateSubnet01Block' (private subnet 01 CIDR), and 'PrivateSubnet02Block' (private subnet 02 CIDR). Each parameter includes its type, default value, and a description.

```
1 Description: 'Amazon Developer EKS Cluster Configuration with VPC - Private and Public subnets'
2
3 Parameters:
4   KeyName:
5     Description: The EC2 key pair to allow to ssh into instance
6     Type: 'AWS::EC2::KeyPair::KeyName'
7
8   VpcBlock:
9     Type: String
10    Default: 192.168.0.0/16
11    Description: The CIDR range for the VPC. This should be a valid private (RFC 1918) CIDR range.
12
13   PublicSubnet01Block:
14     Type: String
15     Default: 192.168.0.0/18
16     Description: CidrBlock for public subnet 01 within the VPC
17
18   PublicSubnet02Block:
19     Type: String
20     Default: 192.168.64.0/18
21     Description: CidrBlock for public subnet 02 within the VPC
22
23   PrivateSubnet01Block:
24     Type: String
25     Default: 192.168.128.0/18
26     Description: CidrBlock for private subnet 01 within the VPC
27
28   PrivateSubnet02Block:
29     Type: String
30     Default: 192.168.192.0/18
31     Description: CidrBlock for private subnet 02 within the VPC
```

Figure 7 - CloudFormation Script Yaml File – Parameters Section

```

Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      -
        Label:
          default: "Worker Network Configuration"
        Parameters:
          - VpcBlock
          - PublicSubnet01Block
          - PublicSubnet02Block
          - PrivateSubnet01Block
          - PrivateSubnet02Block

```

Figure 8 - CloudFormation Script -Metadata Section

```

Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcBlock
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-VPC'

  InternetGateway:
    Type: "AWS::EC2::InternetGateway"

  VPCGatewayAttachment:
    Type: "AWS::EC2::VPCGatewayAttachment"
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: Public Subnets
        - Key: Network
          Value: Public

  PrivateRouteTable01:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: Private Subnet AZ1
        - Key: Network
          Value: Private01

```

Figure 9 - CloudFormationScript - Resource Section 1

```
PrivateRouteTable02:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: Private Subnet AZ2
      - Key: Network
        Value: Private02

PublicRoute:
  DependsOn: VPCGatewayAttachment
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway

PrivateRoute01:
  DependsOn:
    - VPCGatewayAttachment
    - NatGateway01
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PrivateRouteTable01
    DestinationCidrBlock: 0.0.0.0/0
    NatGatewayId: !Ref NatGateway01

PrivateRoute02:
  DependsOn:
    - VPCGatewayAttachment
    - NatGateway02
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PrivateRouteTable02
    DestinationCidrBlock: 0.0.0.0/0
    NatGatewayId: !Ref NatGateway02
```

Figure 10 - CloudFormation – Resources Section - 2



```

NatGateway01:
  DependsOn:
    - NatGatewayEIP1
    - PublicSubnet01
    - VPCGatewayAttachment
  Type: AWS::EC2::NatGateway
  Properties:
    AllocationId: !GetAtt 'NatGatewayEIP1.AllocationId'
    SubnetId: !Ref PublicSubnet01
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-NatGatewayAZ1'

NatGateway02:
  DependsOn:
    - NatGatewayEIP2
    - PublicSubnet02
    - VPCGatewayAttachment
  Type: AWS::EC2::NatGateway
  Properties:
    AllocationId: !GetAtt 'NatGatewayEIP2.AllocationId'
    SubnetId: !Ref PublicSubnet02
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-NatGatewayAZ2'

NatGatewayEIP1:
  DependsOn:
    - VPCGatewayAttachment
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: vpc

NatGatewayEIP2:
  DependsOn:
    - VPCGatewayAttachment
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: vpc

```

Figure 11 - CloudFormation - Resource Section - 3

```

PublicSubnet01:
  Type: AWS::EC2::Subnet
  Metadata:
    Comment: Subnet 01
  Properties:
    MapPublicIpOnLaunch: true
    AvailabilityZone:
      Fn::Select:
        - '0'
        - Fn::GetAZs:
            Ref: AWS::Region
    CidrBlock:
      Ref: PublicSubnet01Block
    VpcId:
      Ref: VPC
    Tags:
      - Key: Name
        Value: !Sub "${AWS::StackName}-PublicSubnet01"
      - Key: kubernetes.io/role/elb
        Value: 1

PublicSubnet02:
  Type: AWS::EC2::Subnet
  Metadata:
    Comment: Subnet 02
  Properties:
    MapPublicIpOnLaunch: true
    AvailabilityZone:
      Fn::Select:
        - '1'
        - Fn::GetAZs:
            Ref: AWS::Region
    CidrBlock:
      Ref: PublicSubnet02Block
    VpcId:
      Ref: VPC
    Tags:
      - Key: Name
        Value: !Sub "${AWS::StackName}-PublicSubnet02"
      - Key: kubernetes.io/role/elb
        Value: 1

```

Figure 12 – CloudFormation – Resource Section 4

```
206
207 PrivateSubnet01:
208   Type: AWS::EC2::Subnet
209   Metadata:
210     Comment: Subnet 03
211   Properties:
212     AvailabilityZone:
213       Fn::Select:
214         - '0'
215       - Fn::GetAZs:
216         Ref: AWS::Region
217     CidrBlock:
218       Ref: PrivateSubnet01Block
219     VpcId:
220       Ref: VPC
221     Tags:
222       - Key: Name
223         Value: !Sub "${AWS::StackName}-PrivateSubnet01"
224       - Key: kubernetes.io/role/internal-elb
225         Value: 1
226
227 PrivateSubnet02:
228   Type: AWS::EC2::Subnet
229   Metadata:
230     Comment: Private Subnet 02
231   Properties:
232     AvailabilityZone:
233       Fn::Select:
234         - '1'
235       - Fn::GetAZs:
236         Ref: AWS::Region
237     CidrBlock:
238       Ref: PrivateSubnet02Block
239     VpcId:
240       Ref: VPC
241     Tags:
242       - Key: Name
243         Value: !Sub "${AWS::StackName}-PrivateSubnet02"
244       - Key: kubernetes.io/role/internal-elb
245         Value: 1
246
```

Figure 13 - CloudFormation - Resource Section - 5

```

PublicSubnet01RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet01
    RouteTableId: !Ref PublicRouteTable

PublicSubnet02RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet02
    RouteTableId: !Ref PublicRouteTable

PrivateSubnet01RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PrivateSubnet01
    RouteTableId: !Ref PrivateRouteTable01

PrivateSubnet02RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PrivateSubnet02
    RouteTableId: !Ref PrivateRouteTable02

ControlPlaneSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Cluster communication with worker nodes
    VpcId: !Ref VPC

```

Figure 14 - CloudFormation - Resource Section – 6

```

Outputs:

SubnetIds:
  Description: Subnets IDs in the VPC
  Value: !Join [ ",", [ !Ref PublicSubnet01, !Ref PublicSubnet02, !Ref PrivateSubnet01, !Ref PrivateSubnet02 ] ]

SecurityGroups:
  Description: Security group for the cluster control plane communication with worker nodes
  Value: !Join [ ",", [ !Ref ControlPlaneSecurityGroup ] ]

VpcId:
  Description: The VPC Id
  Value: !Ref VPC

```

Figure 15 - CloudFormation Script - Outputs Section

```

developerIqCluster:
  Type: AWS::EKS::Cluster
  Properties:
    Name: developerIqCluster
    RoleArn: arn:aws:iam::952614855265:role/developer-iq-cluster-role
    ResourcesVpcConfig:
      SecurityGroupIds:
        - !Ref ControlPlaneSecurityGroup
      SubnetIds:
        - !Ref PublicSubnet01
        - !Ref PublicSubnet02
        - !Ref PrivateSubnet01
        - !Ref PrivateSubnet02

developerIqNodeGroup:
  Type: AWS::EKS::Nodegroup
  Properties:
    NodegroupName: developer-iq-node-group
    ClusterName: !Ref developerIqCluster
    NodeRole: arn:aws:iam::952614855265:role/developer-iq-eks-worker-node-role
    Subnets:
      - !Ref PublicSubnet01
      - !Ref PublicSubnet02
      - !Ref PrivateSubnet01
      - !Ref PrivateSubnet02

    InstanceTypes:
      - t3.small
    RemoteAccess:
      Ec2SshKey: !Ref KeyName
    ScalingConfig:
      MinSize: 2
      DesiredSize: 2
      MaxSize: 2

```

Figure 16 - CloudFormation Script - Cluster Section

## Deployment Diagram

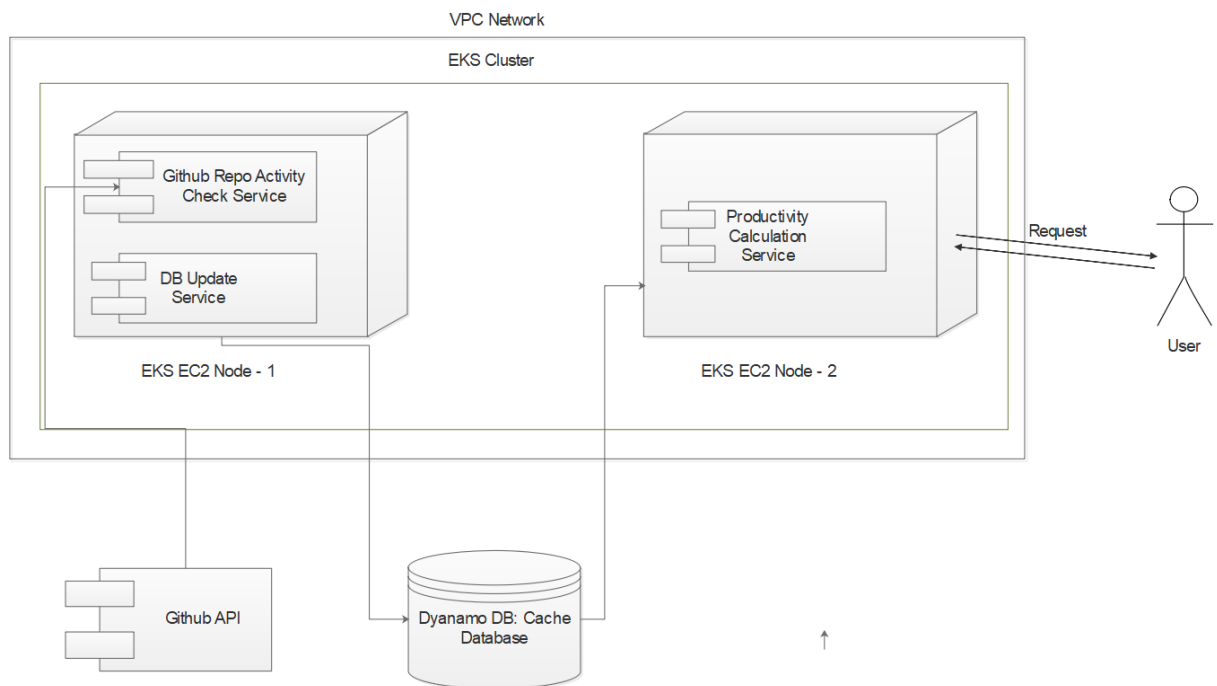


Figure 17 - Deployment Diagram

## Component Diagram

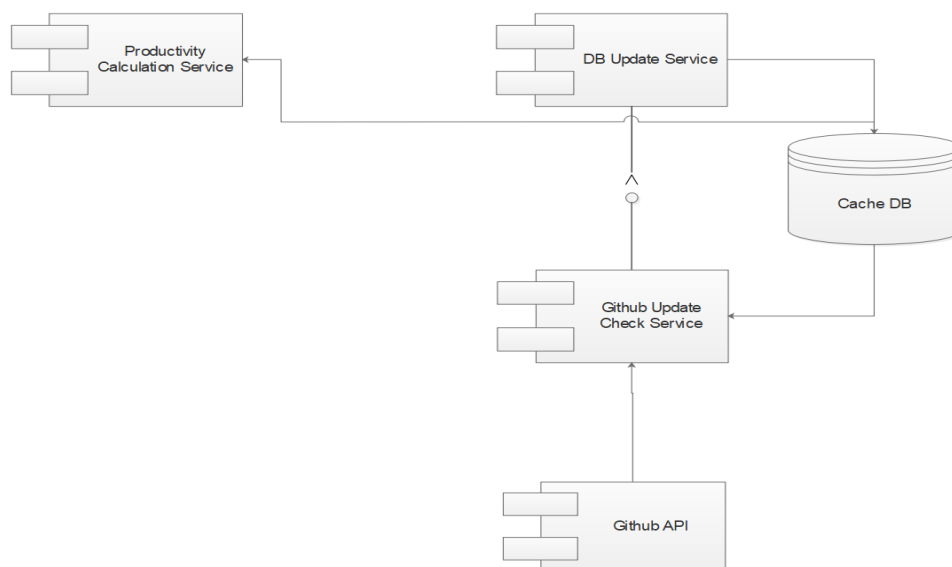


Figure 18 - Component Diagram

## Fault Scenarios and Fault Tolerance Mechanisms

### ***Scenario 1: Node Failure***

**Fault tolerance approach:** The Kubernetes master control plane is configured to be responsible to solve the issue of node failure by spawning new nodes to replace failed nodes.

### ***Scenario 2 : Service Failure***

**Fault tolerance approach :** The concept of Kubernetes replicas is utilized in the deployments so that in the case of a service failure, the Kubernetes control plane realizes that the number of active pods are lower than the number of replicas, thus restarts the pods containing the services

### ***Scenario 3: Unauthorized Access***

**Fault tolerance approach:** IAM roles are created for the node-groups to make sure that the nodes in the cluster only have limited and required control over the AWS services.

### ***Scenario 4 : Interservice communication intercepting***

**Fault tolerance approach :** To make interservice communication efficient and secure we use a hybrid VPC network for the EKS Cluster where the services can communicate with each other using a private network. This VPC configuration prevents against attacks to intercept data in transit between the services.

## Conclusion

In conclusion DeveloperIQ presents a new logical and mathematical method for calculating developer productivity by considering multiple dimensions of their GitHub contribution activity along with a highly scalable and fault tolerant infrastructure. One area of improvement in this application is the automation of the manual workflow of container deployment and ECR key validation by using a workflow automation tool such as Jenkins or GitHub actions.