Overtime Lifecycle Environment



**Prepared By:**

Abdelrahman Mamdouh

Abdelrahman Hesham

Mirna Azmi

Nourelhoda Ashraf

Omar Hesham

Randa Osama

Reham Ashraf

Table of Contents

[Objective 2](#_Toc15306751)

[System Requirements 2](#_Toc15306752)

[Cloud Architecture 3](#_Toc15306753)

[Infrastructure Components 3](#_Toc15306754)

[VPC 3](#_Toc15306755)

[Bastion Server 3](#_Toc15306756)

[Orchestration Machine 3](#_Toc15306757)

[Routing Tables 3](#_Toc15306758)

[API Gateway 3](#_Toc15306759)

[Subnets 3](#_Toc15306760)

[NACLS 3](#_Toc15306761)

[Security Groups 3](#_Toc15306762)

[NAT 3](#_Toc15306763)

[Infrastructure as a code 4](#_Toc15306764)

[Cloud infrastructure diagram 5](#_Toc15306765)

[User Automated Creation 6](#_Toc15306766)

[Cost 7](#_Toc15306767)

[Cloud Development 8](#_Toc15306768)

[Used Services/Tools 8](#_Toc15306769)

[AWS Lambda Service 8](#_Toc15306770)

[Relation Database Services (RDS) 8](#_Toc15306771)

[Simple Email Service (SES) 9](#_Toc15306772)

[API Gateway (Postman) 9](#_Toc15306773)

[Service Diagram 10](#_Toc15306774)

# Objective

We are required to develop a -Cloud Native- system that manages the overtime lifecycle according to the role of each system user.

The current system depends on two sub-systems “CATW” and “EES”, the users should update and insert the data in the two sub-systems, so we are required to create a single point data entry to decrease effort, time also minimize data error.

The new system is cloud native applied on AWS amazon web services trying to gain the max benefit from the AWS services such as RDS, Lambda………etc.

# System Requirements

* Orchestration Machine to manage RDS.
* Complete and secured cloud architecture.
* Service to handle single point data entry.
* Service to handle email approvals.
* Service for storage and archiving.

# Cloud Architecture

The infrastructure is built according to system requirements -security and networking- also to be easily using tunnels to access secured RDS.

## Infrastructure Components

### VPC

The virtual private cloud is created with CIDR range for IPV4 10.0.0.0/24.

All the infrastructure and the services are on the same region “us-east-1” USA Virginia.

### Bastion Server

The bastion server is the only component that exits in a public subnet and can be reached directly from the IGW.

A tunnel is created from the bastion to access RDS for development and testing.

### Orchestration Machine

The orchestration machine is the main machine used to handle RDSs also the machine can host other applications if needed.

The machine exits in a private subnet that can be only accessed using bastion server where the private key is installed.

SQL client is installed on the machine to manage the RDS.

### Routing Tables

### API Gateway

### Subnets

### NACLS

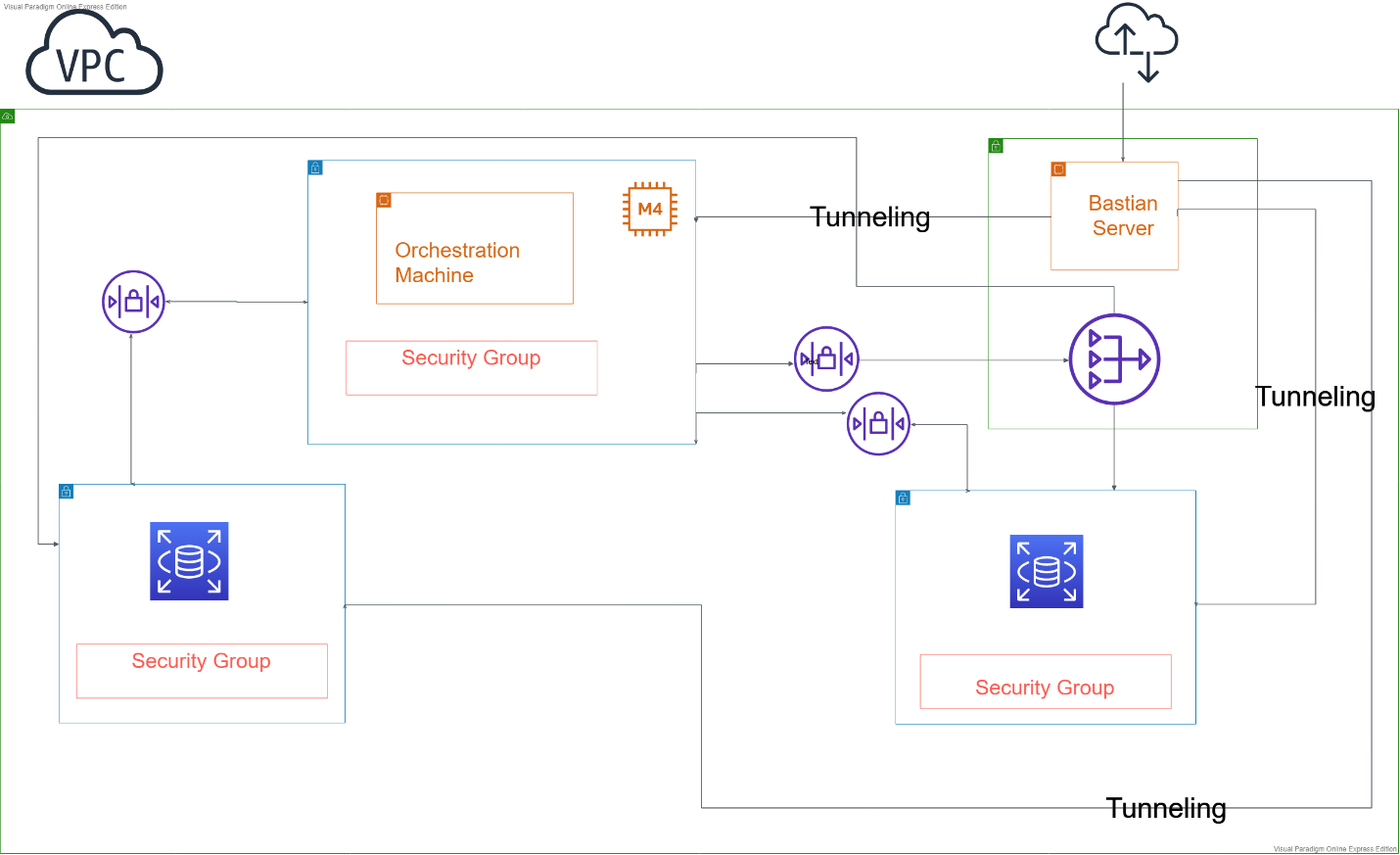
### Security Groups

### NAT

## Infrastructure as a code

Terraform tool is used in creating infrastructure, the tool uses AWS CLI inserted credentials to verify the account and specify the user creating infrastructure

## Cloud infrastructure diagram



## 

## User Automated Creation

## Cost

# Cloud Development

## Used Services/Tools

  -AWS-Lambda Service

  -Relation Database Services (RDS)

  -Simple Email Service (SES)

  -Simple Storage Service (S3)

  -API Gateway (Postman)

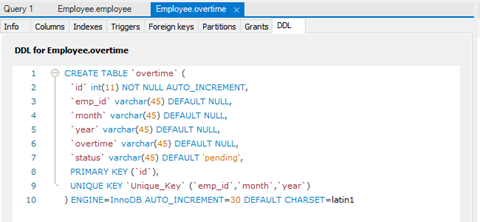
  -Eclipse IDE

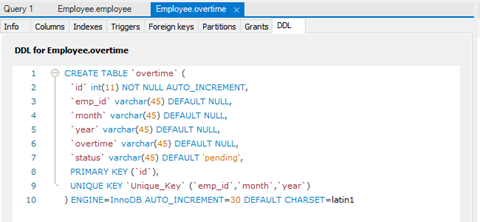
### AWS Lambda Service

Three lambdas functions have been developed to satisfy the core requirements of the project. ***The First Lambda***inserts the overtime information in the overtime table in the database. It also decodes the email content being received as base64 string and upload such on S3 in .msg format with a file name that concatenates the employee ID, the month and the year. ***The Second Lambda***listens to the creation of an object in S3 Bucket(trigger) to get invoked. It uses the employee ID to fetch the data in the database and relates it to their manager’s ID to get the manager email. It also uses ***SES***to send emails with the sender being (assuming) the employee and recipient being the manager with content that contains information about the employee and related overtime information (I.e. Employee A will have X overtime in month Y in Year Z) as well as making the (the employee ID, the month and the year.msg) file attachable to the email sent ***The Third Lambda*** updates the status of approval in the database depending on the response (Approved/Rejected) .

### Relation Database Services (RDS)

MySQL database was used as all the team have knowledge as well as it is included in the free tier. And we made 2 tables, one for employee’s info and the other for the overtime. The employee table contains the (employee ID-name-mail-the manager ID). The overtime table contains (employee ID- month- year- overtime).





### Simple Email Service (SES)

SES was used to send emails because it is a reliable, cost-effective service for businesses of all sizes that use email to keep in contact with their customers. And SNS wasn’t used because Troubleshooting can be very difficult when encountering an issue with SNS and often many AWS services.

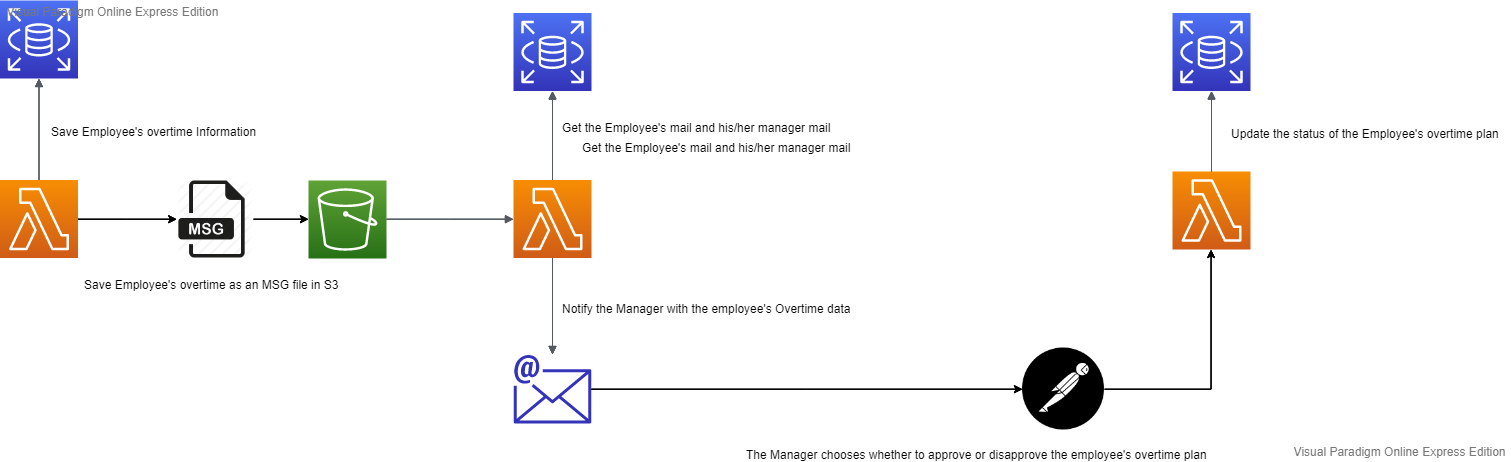
### API Gateway (Postman)

Postman was used to stimulate and test the functions without front-end.

Issues and Difficulties

-The network firewall didn’t allow us to connect to the RDS.

## Service Diagram



We are a team of 7 individuals we’re divided into 2 teams, team 1 developers of 4 and team 2 architecture of 3.  
  
As an architecture team at first we sketched the design of the project architecture then we implemented what is designed on the AWS console using one of the team account.   
  
As developers team they tried their trial codes using eclipse (JAVA programming language) on their personal accounts before integrating all on same account.  
-Region:  
 VPC🡪 N.Virginia

Bucket 🡪N.Virginia

-CIDERS & I/P addresses:  
 1)VPC 🡪10.0.0.0/24  
 2) Subnets

Orch-VM 🡪10.0.0.128/28  
Dev2-RDS🡪10.0.0.16/28  
Networking RDS 🡪 10.0.0.64/26  
Dev1-RDS 🡪10.0.0.0/28

TEST1-RDS 🡪10.0.0.32/28  
TEST2-RDS 🡪10.0.0.48/28  
3) NAT gateway 🡪10.0.0.70

-NACLS:  
 1) DEV-RDS  
inbound:

|  |
| --- |
|  |
| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Allow / Deny** |  |
|  | | | | | |  |
| 1 | MySQL/Aurora (3306) | TCP (6) | 3306 | 10.0.0.133/32 | ALLOW |  |
| 2 | MySQL/Aurora (3306) | TCP (6) | 3306 | 10.0.0.126/32 | ALLOW |  |

Outbound:

|  |
| --- |
|  |
| **Rule #** | **Type** | **Protocol** | **Port Range** | **Destination** | **Allow / Deny** |  |
|  | | | | | |  |
| 1 | ALL TCP | TCP (6) | 0 - 65535 | 0.0.0.0/0 | ALLOW |  |

2) Public-Networking

Inbound:

| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Allow / Deny** |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | |  |
| 1 | ALL TCP | TCP (6) | 0 - 65535 | 0.0.0.0/0 | ALLOW |  |

Outbound:

|  |
| --- |
|  |
| **Rule #** | **Type** | **Protocol** | **Port Range** | **Destination** | **Allow / Deny** |  |
|  | | | | | |  |
| 1 | ALL TCP | TCP (6) | 0 - 65535 | 0.0.0.0/0 | ALLOW |  |

3) TEST-RDS  
Inbound:

| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Allow / Deny** |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | |  |
| 1 | MySQL/Aurora (3306) | TCP (6) | 3306 | 10.0.0.133/32 | ALLOW |  |
| 2 | MySQL/Aurora (3306) | TCP (6) | 3306 | 10.0.0.126/32 | ALLOW |  |

Outbound:

| **Rule #** | **Type** | **Protocol** | **Port Range** | **Destination** | **Allow / Deny** |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | |  |
| 1 | ALL TCP | TCP (6) | 0 - 65535 | 0.0.0.0/0 | ALLOW |  |

4) ORCH-VM  
inbound:

| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Allow / Deny** |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | |  |
| 1 | SSH (22) | TCP (6) | 22 | 10.0.0.126/32 | ALLOW |  |
| 2 | ALL TCP | TCP (6) | 0 - 65535 | 0.0.0.0/0 | ALLOW |  |

Outbound:

| **Rule #** | **Type** | **Protocol** | **Port Range** | **Destination** | **Allow / Deny** |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | |  |
| 1 | ALL TCP | TCP (6) | 0 - 65535 | 0.0.0.0/0 | ALLOW |  |

-We used in the cloud:  
 1)2 EC2  
 2)3 Lambda  
 3)SES  
 4)1 RDS  
 5) 1 S3   
 6) API Gateway  
 7) 1 EBS  
 8)1 NAT  
 9) Internet Gateway  
 10) 2 route tables  
 11)6 subnets (5 private & 1 public)  
 12)Tunneling  
  
Difficulties we encountered:  
 As architecture team:

We had issues to connect to the Bastion server and some security issues we solved some by security groups and NACLs.  
We had issues in creating Bastion Server itself as we realized later it is just an EC2 we special characterization.  
We had issues with connect the VPC to the internet and the solution was NAT.  
We did not know how to make swap memory, so we searched and the issue is solved.  
We were encountered be suitable inbound and outbound rules for the NACls, we discussed and solve the issue.  
  
As developers:  
 Left for the developers team  
Design:  
  
Flow of data and operation:

