SOFTWARE PRODUCTION ENGINEERING

Final Project Report

Matching Memory Game

G.Shanthan Reddy(IMT2018522) P.S.V. Sai Madhav (IMT2018524)

Introduction to Devops

What is Devops?

Devops is a software engineering methodology which practices continuous integration and continuous Delivery/Deployment. Devops benefits both the business and the users. Here we aim at faster reach of end features to the users and users also aim that bugs are fixed soon. Devops aims to better assistance and cooperation between the Dev and Ops teams. As a result Devops smoothens transition of different stages of the SDLC pipeline. Devops also makes the transition more reliable and stable. Devops is a culture of shared accountability between the Dev and Ops teams. Both teams work with a collective mind of upbringing the organization rather than individual achievements.

The common goal of devops is - continually delivering value.One way agile techniques achieve this is by bringing people from all groups together in a team so that they begin to care about the same things. Devops is largely about making it easier for people from these backgrounds to care about the same things by removing roadblocks and streamlining delivery. This includes things like automated build, test, deployment pipelines (continuous integration and continuous delivery), writing code in a way that will handle production level load and can be monitored, and having for example devs and testers on the pager so that everyone in the team has responsibility for how well the system runs in production.

Why Devops?

Main reason for companies to use devops are:

- Faster time to market for software
- Lower DownTime
- Rapid improvement based on feedback
- Less Menial work Thanks to Automation
- less complexity to manage
- Quick defect fixing
- Reduced operational cost, Faster Return on investment

Application Introduction

A fun matching game where you can test how good your memory is and hone your cognitive functions like attention span, concentration and focus. A grid of tiles will be displayed initially. When the game starts, all tiles are turned face down. The player then flips over two cards, selecting them by clicking on them. If the two tiles have the same image, they remain face up, else they will be flipped face down again after a short delay. Flip all the cards as fast as you can to win the game. We provide three levels for the same.

System Configuration

Operating System

- MacOS BigSur

Development Technologies

- Front end: HTML5, CSS, JS
- Backend: Node js, Express framework, Sequelize
- Database: MySQL

Tools used

- Source control management: Git, GitHub
- Testing: Jest, super test
- IDE: Vscode
- Build: npm
- Continuous Integration: Github Actions
- Containerize: Docker, Docker-compose, Volumes
- DockerHub
- Configuration management and deployment: Ansible
- Cloud : AWS
- Monitoring: ELK Stack

Software Development Life Cycle

Development Environment Installations

Node.js

Introduction

Javascript is a major language used in full stack applications. This was created only to be used in browsers. But with the Introduction Node the scenario has changed. Now we can run Anything in using js. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js is perfect for data-intensive real-time applications that run across distributed devices. Simply Node.js runs JavaScript.

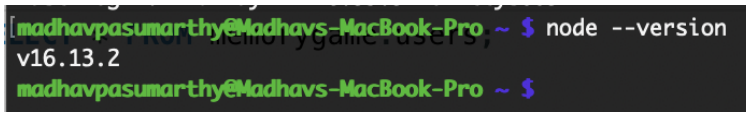
Configuration

Installation of Node:

> Node.js (nodejs.org) go to this website and download a stable version. Follow the instructions provided by setup.

Verification :

> $ node --version



MySql

Introduction

MySQL is an open-source relational database management system (RDBMS). SQL is a language used to interact with SQL databases. MySQL is one type of SQL database system,

as is oracle Sql. These are programs which specialize in handling a lot of data. An application uses these database programs to store its data.

Configuration

Installation of Mysql:

```
$ brew install mysql
```

MySql workbench helps to visualize Mysql instead of using terminal Mysql. MySQL :: Download MySQL Workbench download appropriate version for your laptop.

Verification : If proper process is followed in installation of workbench we find it in the applications section.

Version Control System

Git , GitHub

Introduction

Git is a Distributed version control system. This approach follows the peer to peer approach. This is used to maintain and manage the source code. GitHub is an online web hosting service that allows the developers who use git to share/showcase their code to their peers.

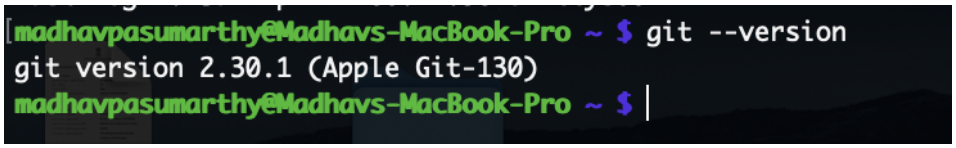
Configuration

Installation of git in My MacBook:

```
$brew install git
```

Verification:

```
$ git —version
```



Configuration of Github:

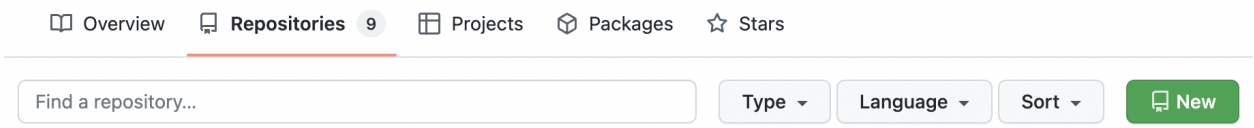
$ git config -- global user.name “myusername”

$ git config -- global user.email “myemail@gmail.com”

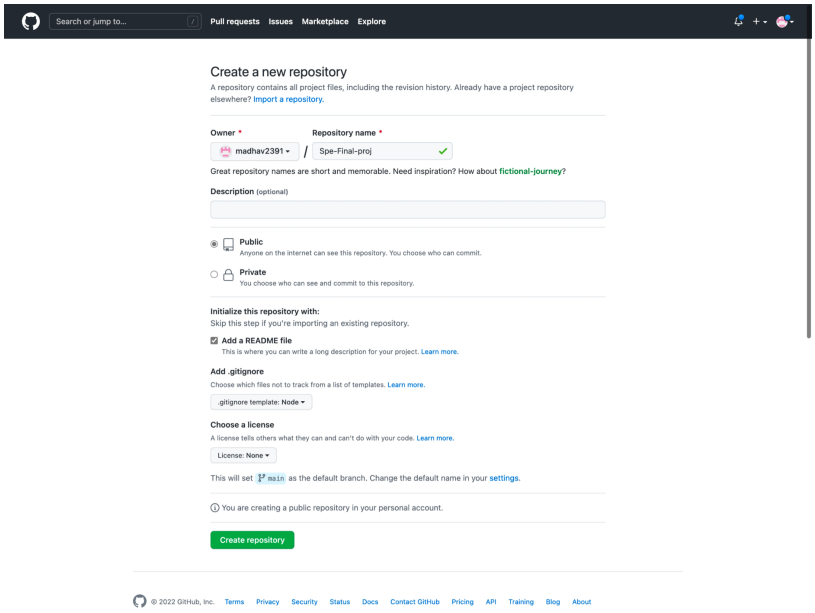
Now we have to create a Repository in GitHub.

Steps :

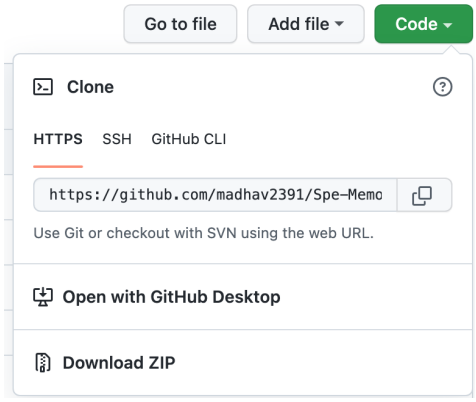
- Go to Your github profile. Navigate to the repositories section. On right we can find a New button click on it.



- Enter the required details of repo creation as below. On completion click on create.



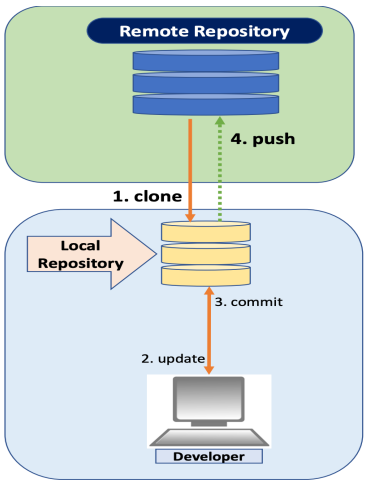
- On success we will be redirected to the repo home page. Next step is to clone the repo into our system. We can see a Green code button which expands as below on click.



- Copy the https url. We will be using this to clone our local system.

  $ git clone <copied url>

- Checkout to a branch, Then we use the git add command to add the files to have the changes in the development history. Next step is to commit the changes to the local repo with a proper commit message. Finally we push the code to our gitHub so our peers can access our code. The image displays the steps explained above. Lookout for Merge conflicts.



Steps to update the remote repo:

$ git pull origin main # do this before adding any feature so that repo becomes uptodate

$ git checkout -b <branch_name>

$ git add *

$ git commit -m “commit message”

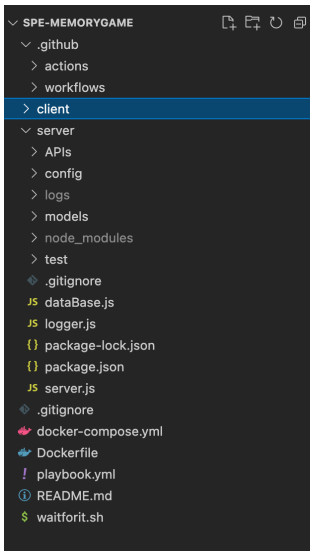
$ git push origin branch_name

In gitHub merge the PR so that main gets up to date.

Note: In git configuration password of github profile might not work so you need to generate a personal access token from [Personal access tokens (github.com)]

Now we have cloned our repo we can start working on the code. Run the below command to get initialized with package.json in the backend folder.

```
$ npm init -y

$ npm start    # we can start the server with this command
```



This is the folder structure followed in our application.

Client folder contains all the front end code, server contains all the backend code. Docker file, Docker compose file, playbook have been added outside of these folders.

Server folder contains package.json files, test folder, apis folder and the required server.js, dataBase.js files.

ExpressJs

Introduction

ExpressJS is a server environment that waits for a browser to connect to it and then the server can send content back to the browser in the form of HTML files, JavaScript files, JSON files, etc.

To do its job, ExpressJS uses APIs made available by Node.js. So ExpressJS is more of a "thin" layer on top of Node.js technologies. ExpressJS also includes some APIs of its own that are more specific to a web server and that are not found in Node.js APIs by default.

Configuration

Run the below command in the server folder.

```
$ npm install express
```

We can verify by checking our package.json file.

Building

Introduction

Npm is the build tool we will be using for our nodejs application.

npm install -> installs all the dependencies mentioned in the package.json

npm test -> runs all the test cases

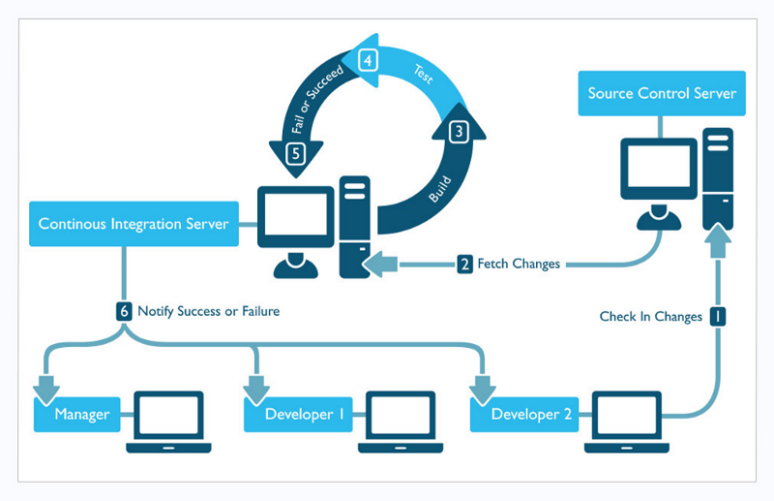
npm start -> start the server application

We are running our server application of 7000 port.

Continuous Integration

Introduction

It is a process of automating the build and testing of code every time when a change is committed to the version control system. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.



Without Continuous Integration we cannot automate from one stage to another stage. In each stage we can automate with the help of an automation tool but if we want to trigger from one stage from another stage then we definitely need continuous integration.

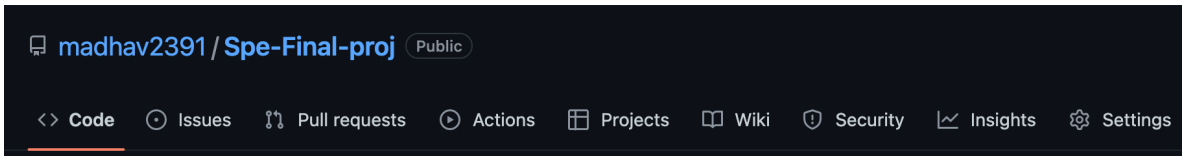
Along with running the unit and integration checks, you can check the quality of your code and profile performance and format the documentation from the source code, thereby facilitating QA processes.

GitHub Actions

Github Actions is an uprising tool that helps the CI/CD pipelines. This tool is offered by GitHub. This helps in automating the process of build,test,deploy.

Configuration

Navigate to the actions sections section of your github repo.



You can find a search bar. Search for node. Github provides a default template. We will be using this and go on updating the file for next stages.

Testing with Jest and superTest

Introduction

Jest is a Javascript Testing framework built by Facebook. It is primarily designed for React based apps but could be used to write automation scenarios for any Javascript-based apps also. There are a lot of advantages or features for using Jest such as:

1. Less configuration
2. **Fast:** Jest tests run in parallel which reduces test execution time.
3. **Built-in code coverage:** Jest supports built-in code coverage which is a useful metric for all CI-based delivery pipelines and overall testing.
4. **Isolated and sandboxed tests:** Each Jest test runs in its own sandbox, which ensures no two tests can impact each other.

5. **Powerful Mocking support:** Jest tests support all types of mocking – be it functional mocking, timer mocking, or mocking individual API calls.

Configuration

In order to install jest:

```
$ npm install --save-dev jest
```

In order to install jest globally:

```
$ npm install -g jest
```

Now, in order to run the Jest tests whenever we enter '*npm test*' we need to configure the package.json file. Go to package.json and add the below script in the scripts section.

```
"scripts":
{
    "test": " jest --testTimeout=10000 --forceExit"
}
```

Now create a test folder in the working directory and write your tests with the name *name.test.js*

```
process.env.NODE_ENV = 'test';
// const { expect } = require('chai');
const request=require('supertest');
const Server = require('http');      You, 5 hours ago • testing working …


const app = require('../server.js').app;
const connectDB=require('../dataBase.js').Users;
```

```
describe("login Testing", () => {
    test("Login successful", async () => {
    const response = await request(app)
      .post("/APIs/login/")
      .send({
        email: "abcd@gmail.com",
        password: "qwerty"
      });
      expect(response.headers['location']).toBe('./../../main.html');
    });

    test("Login unsuccessful", async () => {
        const response = await request(app)
          .post("/APIs/login/")
          .send({
            email: "abcd@gmail.com",
            password: "qwert"
          });
          expect(response.statusCode).toBe(501);
        });
```

server > test > JS api.test.js > ...

```
test("Sign up unsuccessful", async () => {
    const response = await request(app)
      .post("/APIs/signup/")
      .send({
        signemail: "abcd@gmail.com",
        firstname: "madhav",
        lastname: "reddy",
        signpass: "qwerty"
      });
      expect(response.statusCode).toBe(400);
    });

test("Forget password otp sent", async () => {
    const response = await request(app)
      .post("/APIs/forget/")
      .send({
        forgetemail: "abcd@gmail.com",
      });
      expect(response.headers['location']).toBe('../../passwordModifications/conformation.html');
    });

test("Forget password email invalid", async () => {
    const response = await request(app)
      .post("/APIs/forget/")
      .send({
        forgetemail: "ab@gmail.com",
      });
      expect(response.statusCode).toBe(404);
    });

});
```

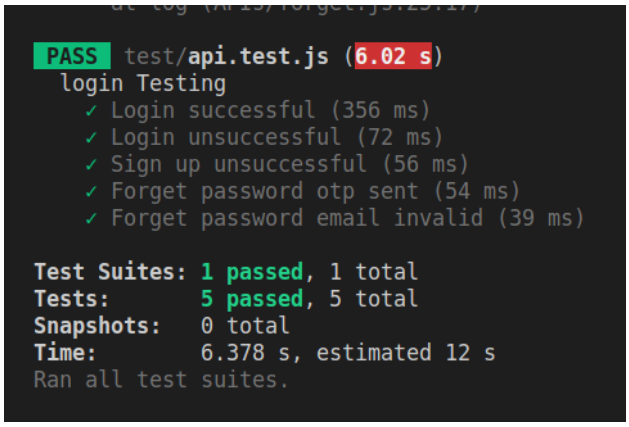
We have written a test suite for testing APIs.

- In the first test we are checking if the login api is successful or not. This is a True positive case. We first listen to the server port and send valid credentials. Since the credentials are valid we must get a valid response which is redirecting to main.html.

We check the *response.headers* and expect it to be redirected to main.html using *expect()* and *toBe()*.

- In the second test, we check the false positive case. So we send an invalid username or password and expect it to not login but fail which it does. Here we check the response code to be 501 statusCode which is the *Not Implemented* error, which we are generating from the server side.
- In the 3rd test, we again check the false positive case for Sign up by sending an already existing email for signing up.
- In the 4th and 5th test cases, we check the true positive and false positive cases for Forgot Password i.e, providing a valid email id for sending OTP and providing a wrong one for error detection.

When we enter *npm test* in the terminal, all the files that are ending with a *test.js* in the test folder will execute and give this:



Test Output from GitHub Actions:

```
995
996  PASS test/api.test.js
997    login Testing
998      ✓ Sign up successful (224 ms)
999      ✓ Login successful (52 ms)
1000     ✓ Login unsuccessful (81 ms)
1001     ✓ Sign up unsuccessful (55 ms)
1002     ✓ Forget password otp sent (60 ms)
1003     ✓ Forget password email invalid (70 ms)
1004
1005 Test Suites: 1 passed, 1 total
1006 Tests:       6 passed, 6 total
1007 Snapshots:   0 total
1008 Time:        2.098 s
1009 Ran all test suites.
```

There are 4 cases we can test a function for:

1. True Positive: This case is when the code is perfect and the test cases pass.
2. True Negative: This case is when the code is perfect but still some cases don't pass.
3. False Positive: This case is when there are errors which the test cases can't find and they pass.
4. False Negative: This case is When the code is broken and the tests don't pass.

As this is a simple project we mainly focused on one true positive and one false positive for each API. The file is written in the test folder of our project with an appropriate class name.

Continuous Delivery

Docker

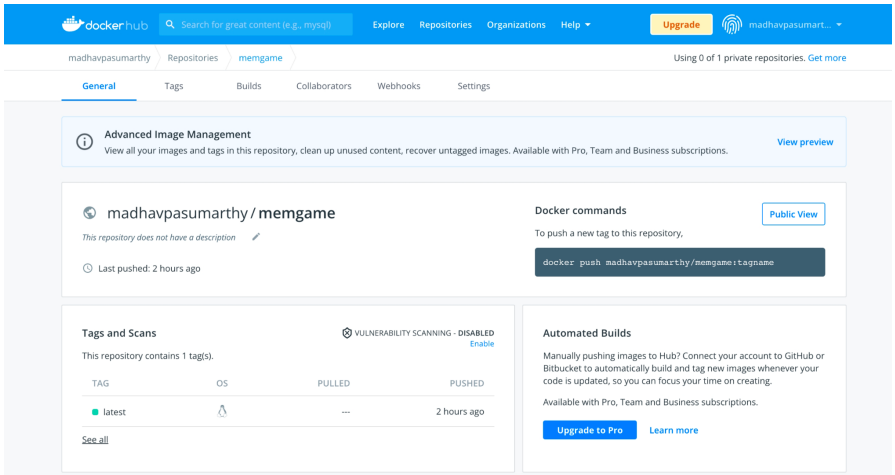
Introduction

Docker is based on the concept of "containers," which are dedicated environments in which a Docker image's services can execute. VM requires a hypervisor, custom OS extensions

and sophisticated management of resources which are unnecessary if you want to deploy small applications like a local web server. One of the main uses of docker is that your code becomes mostly platform independent which helps in your deployment.

VM's use Virtual Domains (space VM uses) while docker creates "Containers," which are essentially sandboxes for whatever image is running inside of them. As a result Containers are very light weight. These containers are going to be loosely isolated, So Rapid deployment and horizontal scaling of the product is going to come in handy.

DockerHub is a way to manage your team's images. This is similar to the Github here we can upload to share our image to our peers. We can also find many official images released by the companies or custom modified of those images of individuals.



Configuration

Installation in macOS:

1. Go to the official docker site (Docker Desktop for Apple silicon | Docker Documentation)
2. Download and open the image
3. After successful installation click on the signin on the top right corner which redirects you to Docker hub. Create an account in the Docker Hub.

Installation in VM Ubuntu server:

1. $ sudo apt update

2. $ sudo apt install apt-transport-https ca-certificates curl software-properties-common
3. $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
4. $ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
5. $ sudo apt install docker-ce
6. $ sudo service docker status
7. Execute docker without sudo (mandatory)
    a. $ sudo groupadd docker
    b. $ sudo usermod -aG docker ${USER}
    c. $ newgrp docker
    d. Restart the OS
    e. Check if docker runs without sudo
        i. $ docker run hello-world

Docker Commands:

- docker pull: docker pull <image name>
- show all containers: docker ps -a
- Docker run: docker run -i <imageID>
- Remove a image: docker rmi <imageID>
- Remove a container: docker rm <imageID>
- docker-compose up
- docker-compose up –build
- docker-compose down

Docker file:

Dockerfile helps to convert our js code application to Docker images so that the application is prepared to be deployed. Dockerfile acts like a blueprint to create our image.

```
1   FROM node:latest      madhavpasumarthy, 5 days a
2
3   WORKDIR ./
4
5   COPY ./server/package*.json ./
6   COPY waitforit.sh ./
7   RUN chmod +x waitforit.sh
8   #RUN npm install1
9   # If you are building your code for production
10  # RUN npm ci
11  # RUN npm install
12
13  COPY . .
14
15  # EXPOSE 7000
16  RUN cd server
17  RUN npm install
18  RUN cd ..
19  # CMD [ "node", "server.js" ]
20  # RUN ./waitforit.sh
21  CMD ./waitforit.sh
22
```

Explanation : We are copying the package.json files, and installing those dependencies. Also we are copying my *waitForIt.sh* file which contains the npm start command. We need to give executing permissions for the shell script so it can run.

Docker Compose

Introduction

Imagine you want a few containers to talk to each other. We have to create a network beforehand and make sure we enter the fields of the network for all the containers and make them run parallelly and enter run commands for all the containers manually. This might be easy for two containers but when the project grows and more people add this process is error prone. To help with this Docker has implemented a feature docker-compose. Docker-Compose takes care of creating a network and making sure the containers are run parallelly.

Docker Compose also provides an option called depends_on. Generally all the containers will start parallelly but depend on creating a dependency and make sure to start the container only after the start of upstream container.

Note that depends_on is only going to stop the downstream container till the upstream container passes the entrypoint, not till it's completely up and running. This was the challenge we have faced, Nodejs container was up and running fast but the Mysql server is taking To counter this we have created a waitForIt.sh file. It makes sure that Nodejs server starts late. Below(on left) is the implementation of *waitForIt.sh* file.

```
#!/bin/bash
echo "going to sleep for 30s"
sleep 30
echo "returning from sleep"
cd server
npm start
```

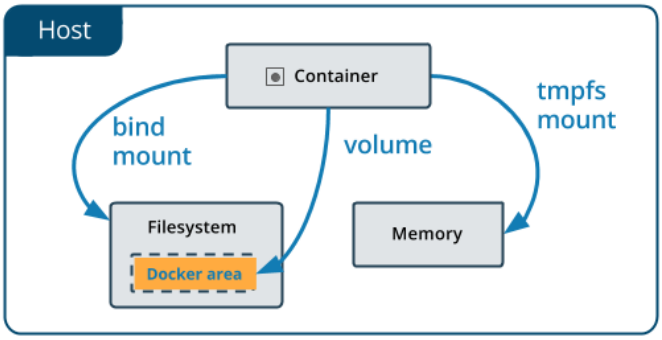
```
version: '3.0'
services:
  mysqldb:
    # platform: linux/x86_64
    image: mysql:5.7
    restart: unless-stopped
    # env_file: ./.env
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: memorygame
      MYSQL_TCP_PORT: 3306

    ports:
      - 3307:3306   #Host:Container
    volumes:
      - db:/var/lib/mysql

  app:
    depends_on:
      - mysqldb
            # condition: service_healthy
    # command: [./waitforit.sh"]
    build: ./
    restart: unless-stopped
    # env_file: ./.env
    ports:
      - 7001:7000
    environment:
      - DB_HOST=mysqldb
      - DB_USER=root
      - DB_PASSWORD=Madhav@19
      - DB_NAME=memorygame
      - DB_PORT=3306
    stdin_open: true
    tty: true
volumes:
  db:
    driver: local
```

From the above image(on right) we are creating 2 containers one for database (mysqldb) and one for nodejs (app). We have specified the Dockerfile for the nodejs server. Database takes the image from the dockerHub if not already present. We have also specified the ports for the containers and which hots ports are mapped to the containers ports.

Docker Volumes

Introduction

Usually when we restart a container all the data of the container will be lost. Like all the tables of the database will be lost. But we require persistence of data. To support this, Docker has come up with the concept of docker volumes. The major difference in volumes is that data from previous container instances will not be lost. The data will be stored inside the host file system.



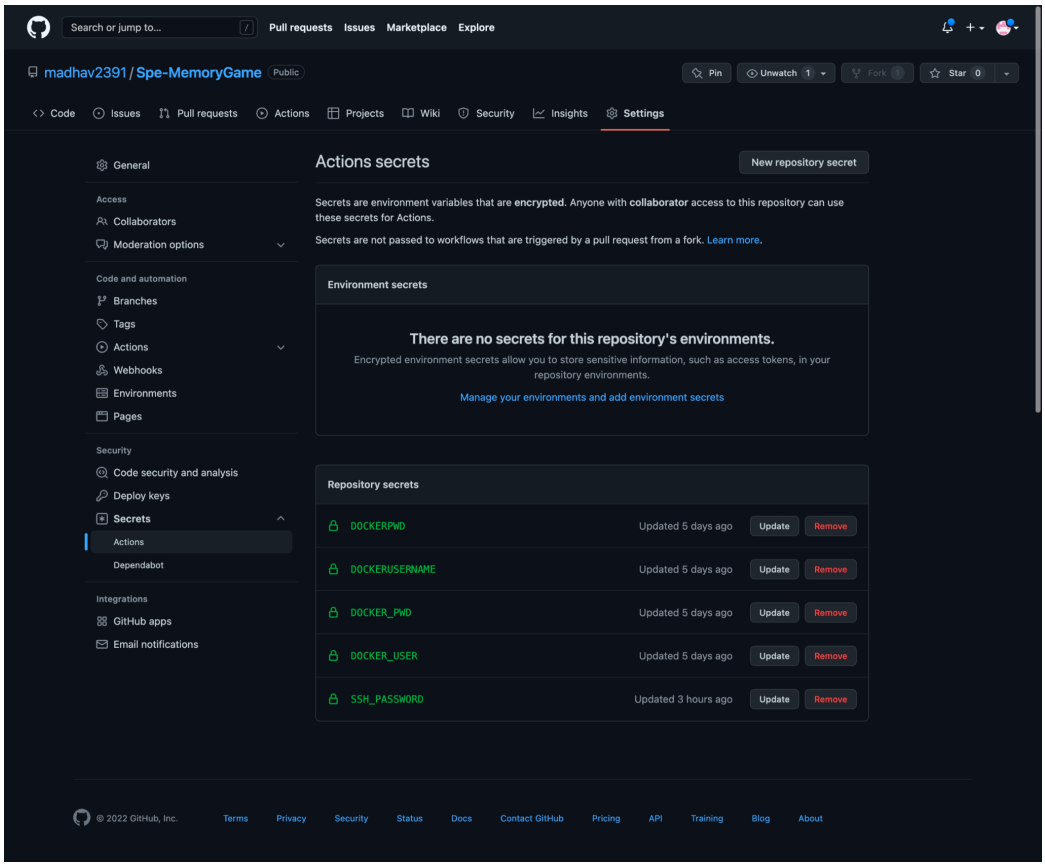
In the image of Docker, compose file lines 36-38 are added to support the persistence in data. Lines 15-16 are written so that my database has persistence of data. The file location in line 16 is where mysql stores its data. Note that this location is specific to MySql and other databases have different locations.

Github Actions Updated workflow:

By now we have our Docker Ready. We can start the Continuous Delivery process in the GithubActions.

```yaml
name: Node.js CI

on: [push]
jobs:
  Docker-build:

    runs-on: ubuntu-latest

#     strategy:
#       matrix:
#         node-version: [12.x, 14.x, 16.x]
#         # See supported Node.js release schedule at https://nodejs.org/en/about/releases/

    steps:
    - uses: actions/checkout@v3
    #- uses: actions/setup-node@v3
    - name: List files in the repository
      run: |
          ls
    - name: pwd  the repository
      run: |
          pwd
    - name: Build the Docker image
      run: docker build . --file ./Dockerfile --tag madhavpasumarthy/memgame

    - name: Docker Login
      env:
        Docker_Usr: ${{secrets.DOCKER_USER}}
        Docker_Pwd: ${{secrets.DOCKER_PWD}}
#       run: echo "$DOCKERUSER"
#       run: echo "$DOCKERPWD"

      run: docker login -u $Docker_Usr -p $Docker_Pwd

    - name: Build the Docker image
      run: docker push ${{secrets.DOCKER_USER}}/memgame
```

Before running this file we need to ensure that Secrets are added. To add a secret Go to the setting of this repo -> in the Security section we can find the Secrets -> navigate to the action in the secrets -> click on the New repository secret in the top right corner.



Add the name of the secret and the Value of the Secret (Docker Username and Password).

Now we can test our pipeline. Below is the description of the workflow

Line 26-27 builds the image from the Dockerfile of our application.

Line 29-36 We make a login to our Docker hub account. This account contains our images.

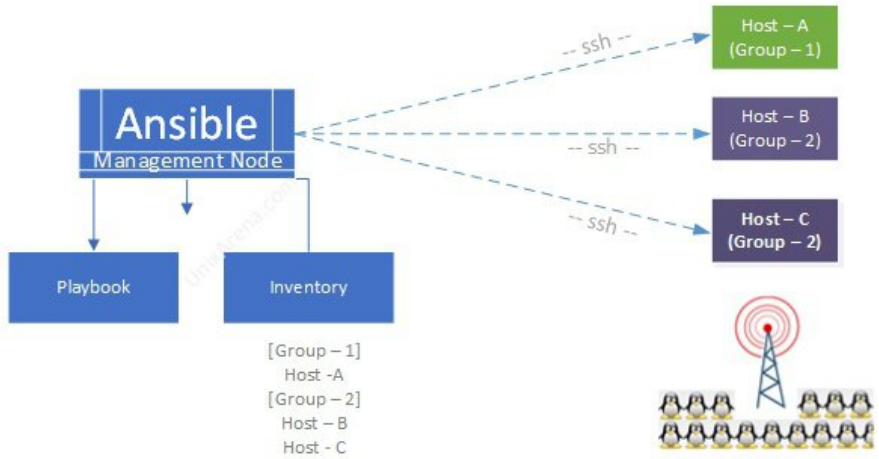
Line 38-39 we are pushing the image to docker hub. By this process we have completed the Continuous Delivery pipeline.

Ansible

Introduction

Ansible is a configuration tool and Automation tool. This uses infrastructure as a service approach to manage various hosts at a single time. Ansible is agentless while other

configuration management tools such as Puppet are not. As a result you can have more space resources for the application. Ansible will be talking to the clients using ssh. Ansible requires python to run on the client side. Ansible can manage both local and remote hosts.



Ansible connects to the server through SSH and pushes out little programmes i.e, Ansible Playbooks(Yaml file). These play books follow human readable syntax so the developer need not learn a new advanced language instead directly work on the program. This playbook contains the instructions for the client to execute. This playbook can contain different instructions for different instructions for different hosts/Groups.

Continuous Deployment:

Continuous deployment is DevOps release strategy where any commit that passes the automated checks is automatically released into the production environment.

After successfully pushing the images into DockerHub, it is time to pull the images and deploy them.

Inventory

Inventory file contains all the target machine details. Only targets added here can be used in the Playbook.

Ansible Playbook

As mentioned Playbooks contain the instructions to be executed in the target machines

In order to pull the images we use the below playbook:

```
---
- name: Deploy docker img
  hosts: aws
  tasks:
    - name: Pull app image
      docker_image:
        name: madhavpasumarthy/memgame:latest
        source: pull
    - name: Pull mysql
      docker_image:
        name: mysql:5.7
        source: pull
    - name: Copying docker-compose file
      copy:
        src: ./docker-compose.yml
        dest: ~/
    - name: docker compose up
      command: docker-compose up -d
```

Here we have specified the host to be aws since the managed host that we are deploying into is AWS. Then we are trying to pull the memgame image with the latest tag which is basically our app. We will also be pulling the database image which is mysql 5.7 from DockerHub. Since we are trying to deploy the app in AWS, we are also going to copy the docker-compose file and send it to the remote server where we can simply run the file.

- Now, we automate the process of building the containers and running the app by running *docker-compose up* in the ansible playbook itself. One problem for using this command is since the containers are always running up, the CI/CD pipeline doesn't stop. Hence it is essential to use the option "-d" which will let the compose exit after starting the containers, but the containers will continue to run in the background.

Now put the below code in the Github Actions for Continuous Deployment:

```
49    deploy:
50
51      needs: Docker-build
52      runs-on: ubuntu-latest
53
54      steps:
55      - uses: actions/checkout@v2
56      - uses: ./.github/actions/ansible
57        env:
58          SSH_PASSWORD: ${{ secrets.SSH_PASSWORD }}
```

This will trigger the action.yml file present in the actions folder which is:

```
1   name: "Ansible"
2   description: "Runs an Ansible playbook"
3   runs:
4     using: "docker"
5     image: "Dockerfile"
```

When this file is triggered, it sets off Dockerfile which is present in the same actions folder.

```
1   FROM alpine
2
3   ENV ANSIBLE_HOST_KEY_CHECKING=False
4
5   RUN apk add ansible gcc python3-dev libc-dev libffi-dev openssl-dev
6   RUN apk add python3 py3-pip openssh-client sshpass
7   RUN pip3 install --upgrade paramiko
8   RUN pip3 install docker
9
10  COPY hosts /hosts
11
12  COPY ansible.cfg /etc/ansible/ansible.cfg
13
14  COPY entrypoint.sh /entrypoint.sh
15
16  CMD ["sh", "/entrypoint.sh"]
```

The above Dockerfile is built on alpine as its base image. All Ansible's required modules and packages are installed. Once the environment is set up, all the required files are copied from the base to docker environment.

The ansible.cfg is as follows:

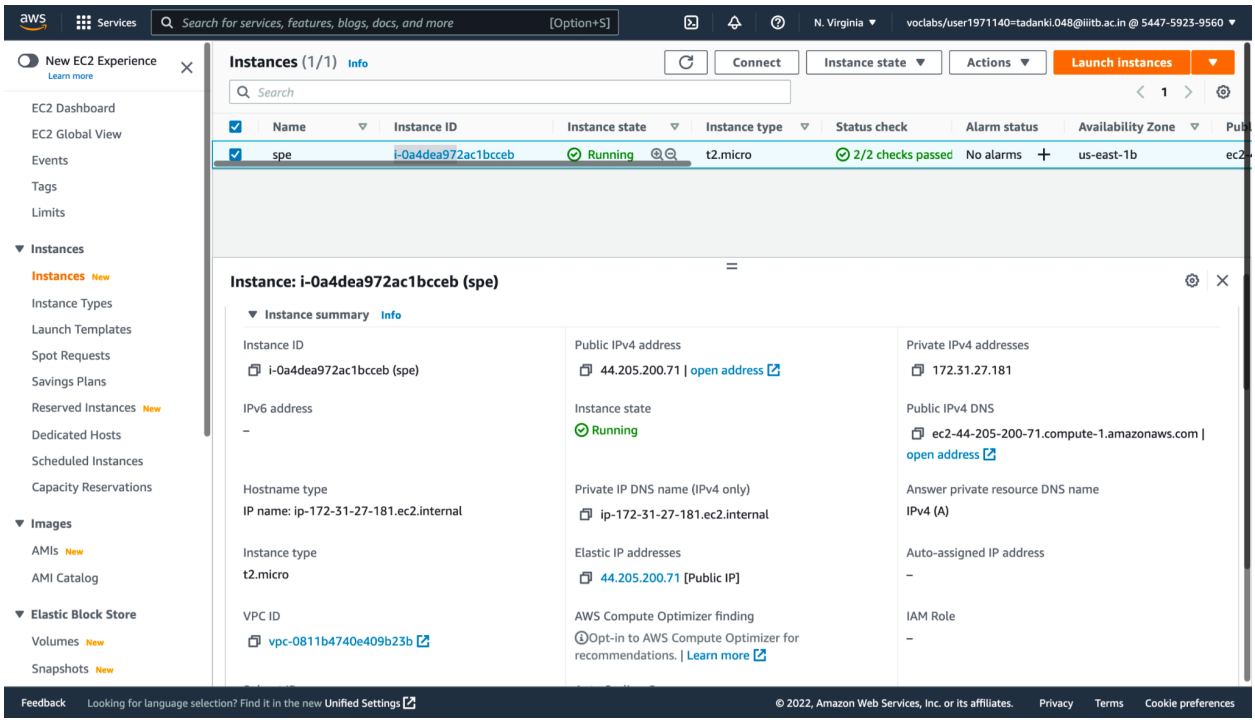
```
1   [defaults]
2   inventory           = /hosts
3   remote_tmp          = /tmp/.ansible-${USER}/tmp
```

When the dockerfile is successfully set up we will run a script file in the linux image created.

```
#!/bin/sh

echo "Ansible Entrypoint"

echo "This is the secret: $SSH_PASSWORD"

ansible-playbook playbook.yml --user ubuntu
```

All the secrets information is taken from the github secrets i.e(SSH username and SSH Password). Public IP Address of the slave is stored in /hosts file. Once the complete setup is done. Ansible command is run to run the playbook.yml file. After successful SSH into the slave machine, our playbook triggers and runs.

Ubuntu server in AWS



To complete implementation of Ansible we need to create a remote Target machine. This can be done by installing a VM ubuntu server on AWS.

Aws gives a dynamic ip address on every instance run. It will be tedious to update the ip every time in the inventory file so I have created an elastic Ip address which will be the same every time.

Openssh has to be installed in both local and target machines.

Aws provides a password file. We have to make configure so that ansible can run without this password file. For this we need to execute below commands

$ ssh –i “pass.pem” ubuntu@<public_ip4> # ssh to aws server

$ sudo passwd ubuntu # to set a password. Type the password in terminal

$ sudo vim /etc/ssh/sshd_config # opens the vim editor (make sure you have a vim editor in your laptop)

In the vim editor navigate to the passwordAuthentication and flip the value.

$ service ssh restart # to restart the instance

Now we can ssh using the password.

Make sure that the target machine has python and docker, docker compose installed. Make sure you can run docker in normal user mode. Steps are specified in the docker section.

We can ssh into the ubuntu server just by using

$ ssh ubuntu@44.205.200.71

To expose a port enter the following commands :

$ ssh -L 7001:localhost:7001 ubuntu@44.205.200.71 # for nodejs

$ ssh -L 3307:localhost:3307 ubuntu@44.205.200.71# for mysql

With these two commands we have mapped our laptop port of 7001 to aws 7001 and 3307 to 3307 of aws.

We can go to localhost:7001 to access our website.

We can create a 3007 port in MySql workbench to visualize the database.

We can also verify that the volumes for the database is working here.

Updated Github actions

Now we can Add the Continuous Deployment code to our github actions file. Make sure You created the secret for the ssh user password.

```
49    deploy:
50
51      needs: Docker-build
52      runs-on: ubuntu-latest
53
54      steps:
55      - uses: actions/checkout@v2
56      - uses: ./.github/actions/ansible
57        env:
58          SSH_PASSWORD: ${{ secrets.SSH_PASSWORD }}
```

Here we have specified that this step used the files from the ./github/actions/ansible folder. Files present in this folder have been explained above. By the end of this implementation we have deployed the docker images in our AWS ubuntu server.

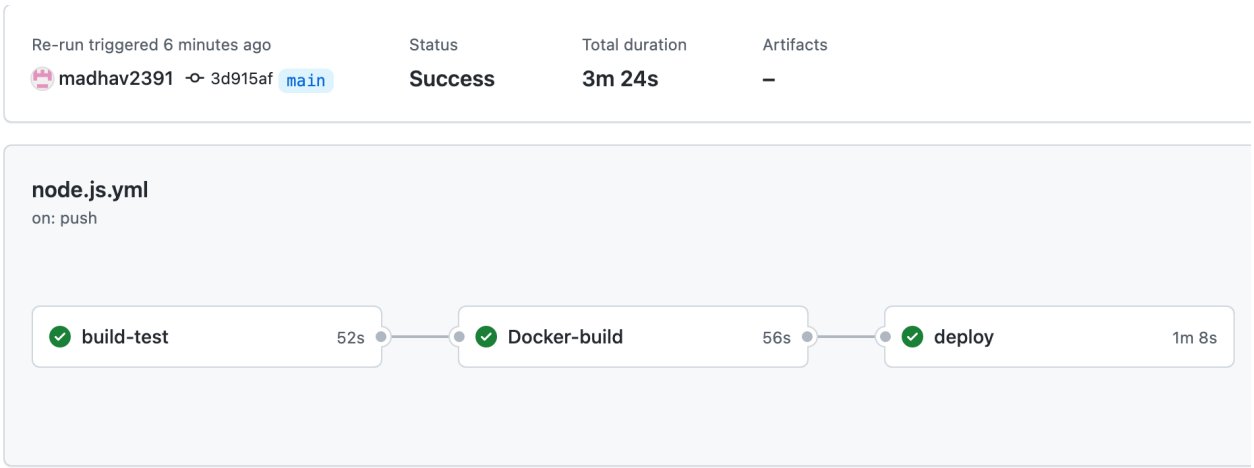
After Deployment if you want to run the code use

$ docker-compose up

And to stop :

$ docker-compose down # shuts down the containers and removes the docker network.

Final Pipeline of the Memgame



Summarizing the process, In the first stage we are building and running integration testing .

In the second stage i.e, Docker-build we are building the Docker image and pushing it to the Docker hub.

In the last stage i.e, deploy we are using ansible an agentless tool to deploy the images to managed hosts i.e, AWS server. docker-compose up is also being automatically triggered.

Below is the snapshot of the Deployed containers in AWS. We can see that the server (app) container and mysql container are running without manual intervention.

Continuous Monitoring is also done using sentry. This is explained in detail below.

Logging with Winston

Introduction

Logging is a process of recording information generated by application activities into log files. Messages saved in the log file are called logs. A log is a single instance recorded in the log file. Winston processes your app activities and generates useful information into a log file or a database. Afterwards, you can check all activities generated by your application.

Winston, is one of the best logging middlewares which has a lot of advantages or features:

- It's simple to use and is configurable.
- Winston provides logging levels. These levels indicate priority which gives you the ability to sort out critical logs from logs requiring less attention.
- Winston can send and save logs in different ways, such as files, databases, emails, and console.
- Winston provides you with several log formats. For example, when saving a log to a Mongo database, the log format needs to be in JSON format. Winston solves that for you.

- Winston helps you profile a block of code and measure the time the code takes to be successfully executed.

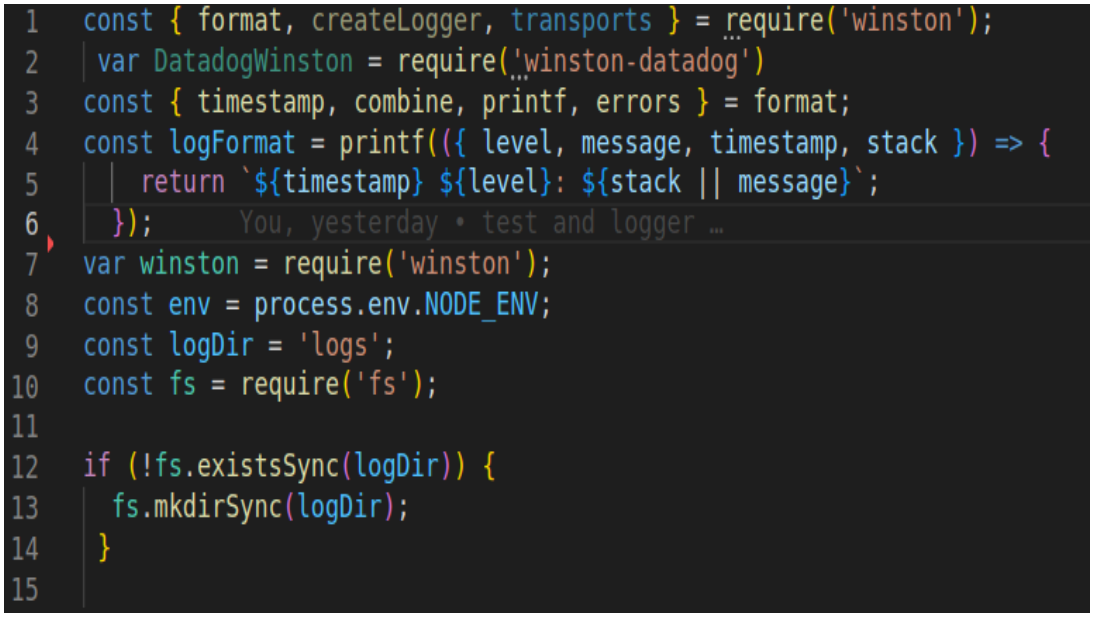
Configuration

In order to install winston use:

```
$ npm install winston
```

And import using:

```
var winston = require('winston');
```



```
const { format, createLogger, transports } = require('winston');
var DatadogWinston = require('winston-datadog')
const { timestamp, combine, printf, errors } = format;
const logFormat = printf(({ level, message, timestamp, stack }) => {
    return `${timestamp} ${level}: ${stack || message}`;
  });      You, yesterday • test and logger …
var winston = require('winston');
const env = process.env.NODE_ENV;
const logDir = 'logs';
const fs = require('fs');

if (!fs.existsSync(logDir)) {
  fs.mkdirSync(logDir);
}

```

```
const now = new Date();
var Logger = winston.createLogger({
        format: combine(
    //    format.colorize(),
      timestamp({ format: 'YYYY-MM-DD HH:mm:ss' }),
      errors({ stack: true }),
      logFormat
    ),
transports: [

    new winston.transports.File({
        name: 'error-file',
        filename: './logs/exceptions.log',
        level: 'error',
        json: false
    }),

    new winston.transports.File({
        name: 'info-file',
        filename: './logs/info.log',
        level: 'info',
        json: false
    }),
```

Now go to the files that you want to be logged and import this logger.js module.

```
var Logger = require('./../logger')
```

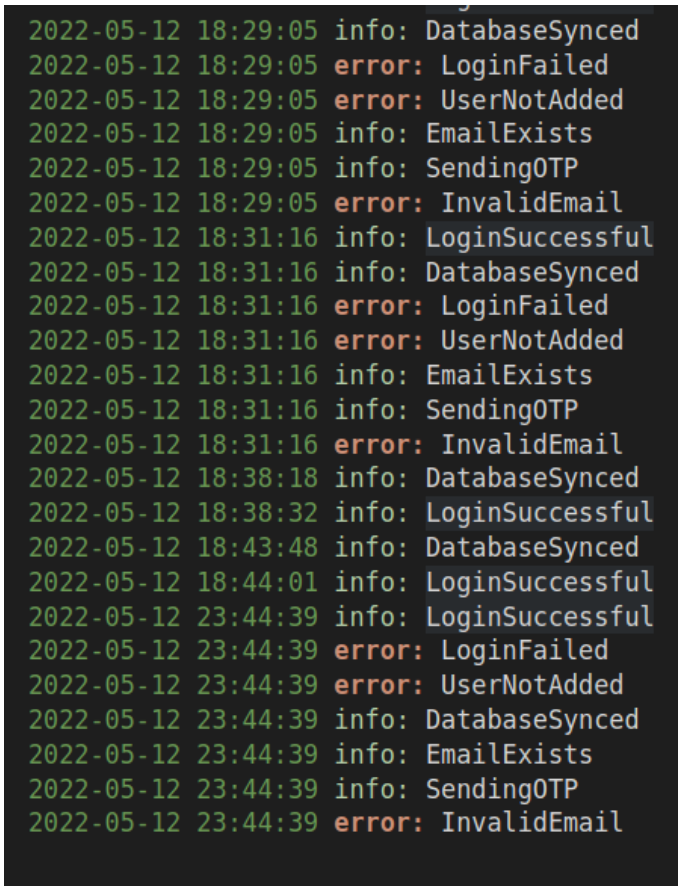
Then whenever you want an event of a method to be logged, do it so by including:

```
Logger.info("RegisteredSuccessfully");
```

If you want an error or exception to be logged, do it so by including:

```
Logger.error("InvalidEmail");
```

We are going to create a Log file which has logs with {*timestamp, log_level, log_info}* as the format. The log files will be stored in the Logs folder under *info.log*.



Continuous Monitoring

Automation plays an important role in DevOps and companies are increasingly adopting DevOps culture for its continuous integration and continuous delivery approach. Continuous Monitoring , is an automated process by which DevOps team can observe and detect compliance issues and security threats during each phase of the DevOps pipeline. There are many monitoring tools for each section for end-to-end support. In this project we'll be using Sentry, AWS and ELK Stack.

Sentry

Sentry is a service that helps you monitor and fix crashes in real time. It contains a full API for sending events from any language, in any application.

Since we are using Winston Logger, we can configure it to work along with Sentry to automatically send the log information and monitor.

```
$ npm install --save @sentry/node @sentry/tracing
```

```
$ npm install winston-sentry-log
```

Now import this into our logger file:

```
var sentry = require('winston-sentry');
```

Now add the below script for logging errors and infos:

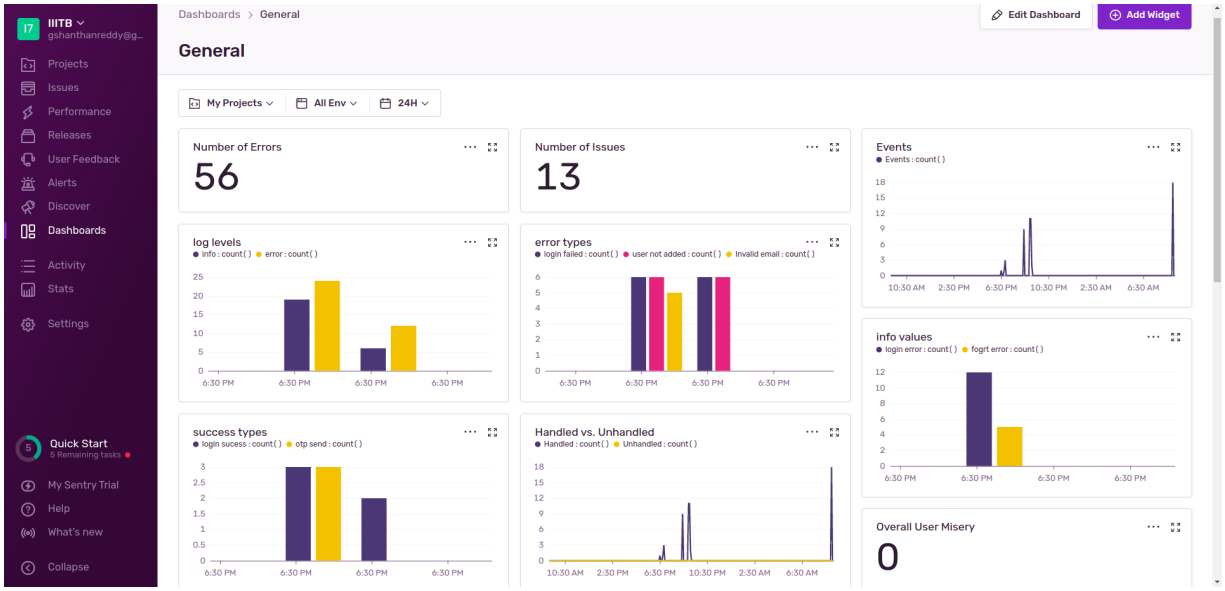
```
new sentry({
        level: 'error',
        dsn: "https://7787d3090f8a4eeda3837781e55af88d@o1245576.ingest.sentry.io/6402831"
}),


    new sentry({
      level: 'info',
      dsn: "https://7787d3090f8a4eeda3837781e55af88d@o1245576.ingest.sentry.io/6402831"
}),
```

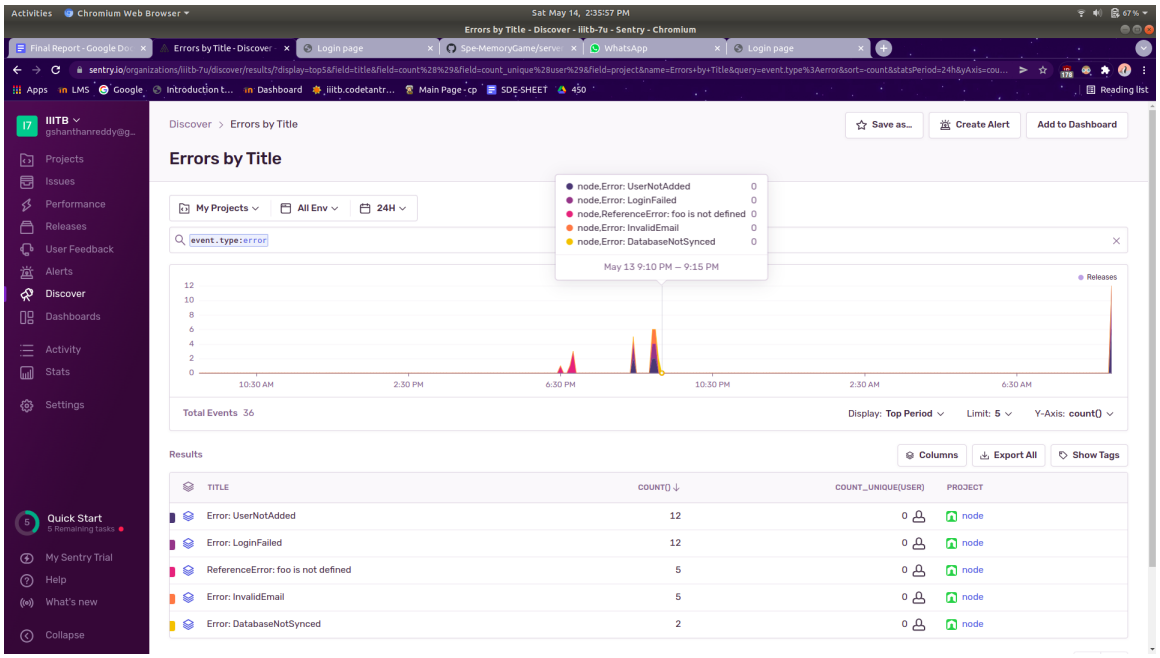
You can find your own DSN once you register into Sentry and select the Node js project.

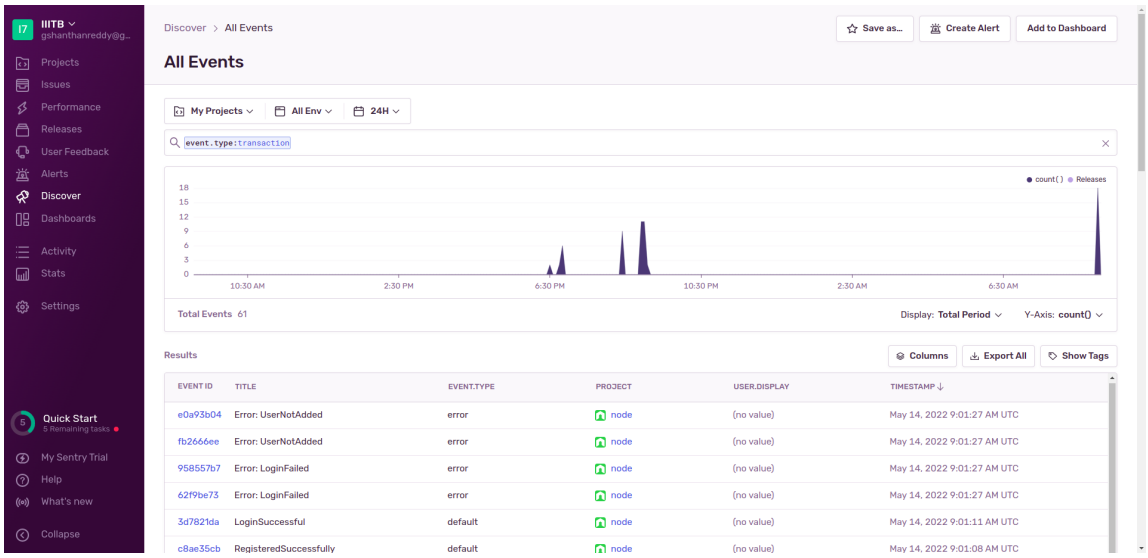
Next we need to add the below configuration into our Server.js:

```
const Sentry = require("@sentry/node");
// or use es6 import statements
// import * as Sentry from '@sentry/node';

const Tracing = require("@sentry/tracing");
// or use es6 import statements
// import * as Tracing from '@sentry/tracing';

Sentry.init({
  dsn: "https://7787d3090f8a4eeda3837781e55af88d@o1245576.ingest.sentry.io/6402831",

  integrations: [
    new Sentry.Integrations.Http({ tracing: true }),
    new Tracing.Integrations.Express({
    app,
    }),
    ],
  tracesSampleRate: 1.0,
});

app.use(Sentry.Handlers.requestHandler());
app.use(Sentry.Handlers.tracingHandler());
```

From now on, whenever an event occurs, the log information will be sent to our Node project inside Sentry and we can visualize them.







ELK Stack

Introduction

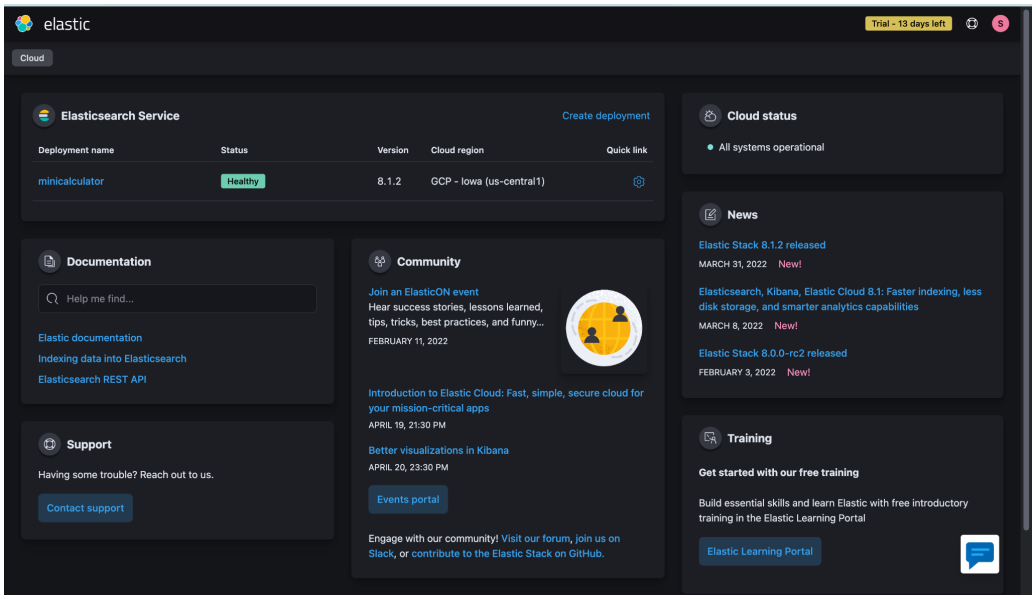
"ELK" stands for Elasticsearch, Logstash, and Kibana which are three open source projects. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then

sends it to a "stash" like Elasticsearch. Kibana visualizes data with charts and graphs in Elasticsearch.

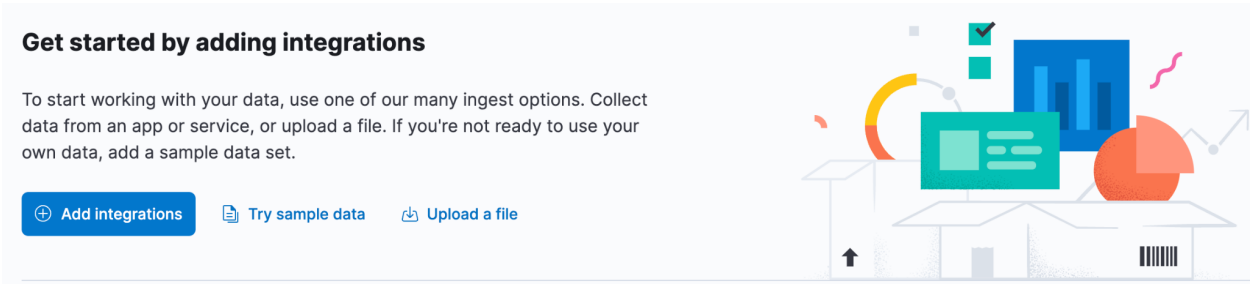
Configuration

Create an account in[The ELK Stack: From the Creators of Elasticsearch | Elastic](#).

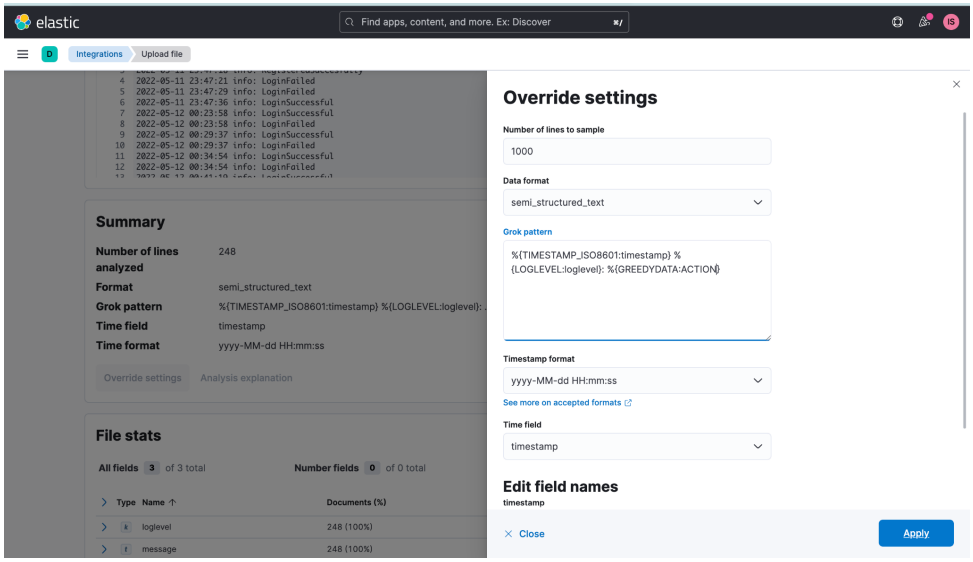
Select Create a deployment. Fill in the details and click on the deployment. Display will be similar to the below image



Next step is to upload a file. Click on the upload file option.

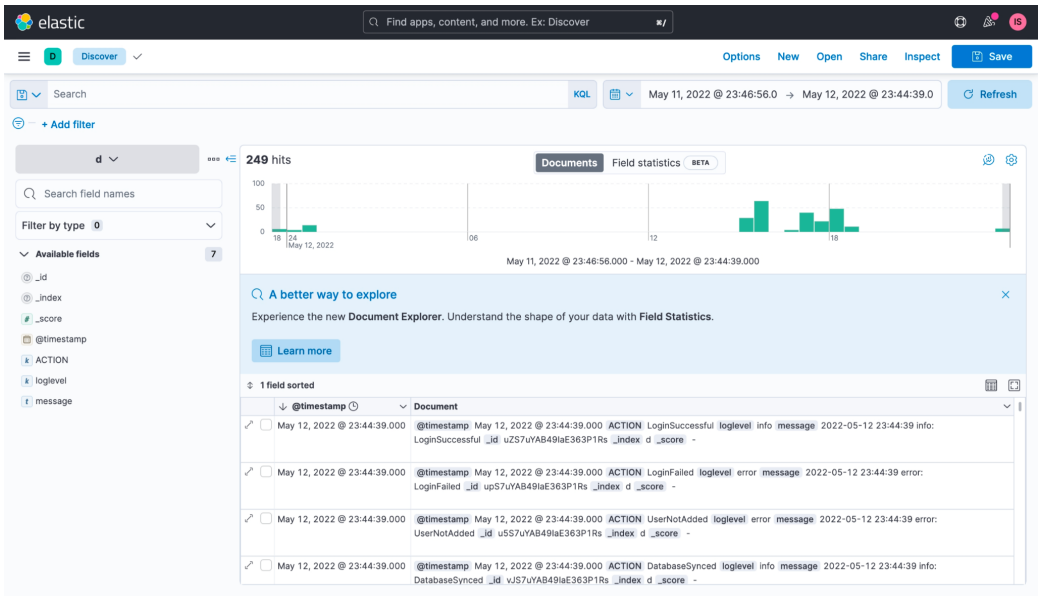


Once we have selected that option we will be asked to drag a file. Drop the log file created by the project. We will provide an overRide settings button. Click on It and update the grok pattern as shown in the below image.

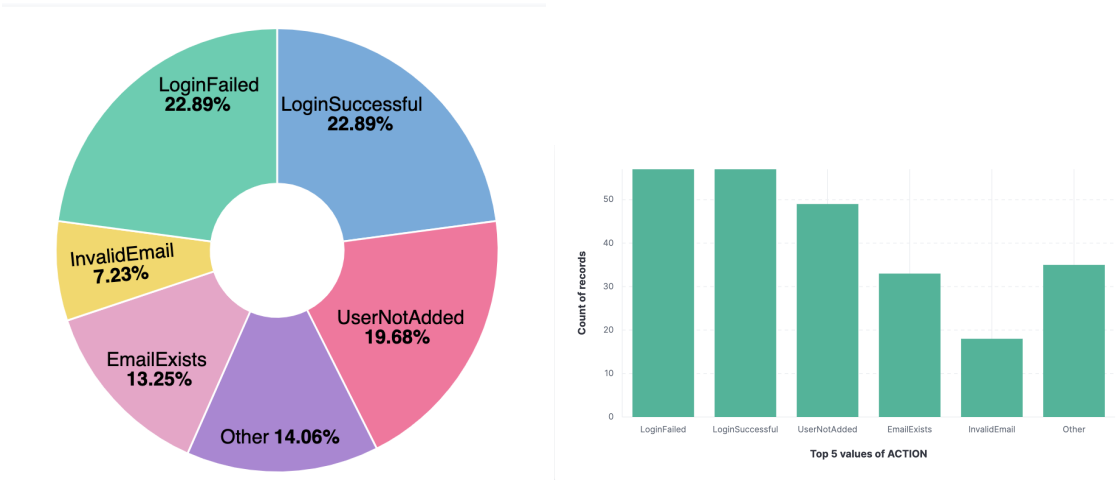


If the parsing was successful you get an option to import. Click on import.

Also fill in the index name. On click of import the pipeline will be completed you can see a graph and we can start visualizing the data.

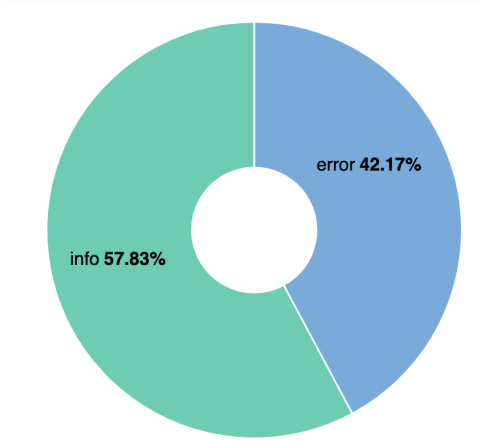


Select action from the Available fields and click on visualize to see distributions of the executed operations.



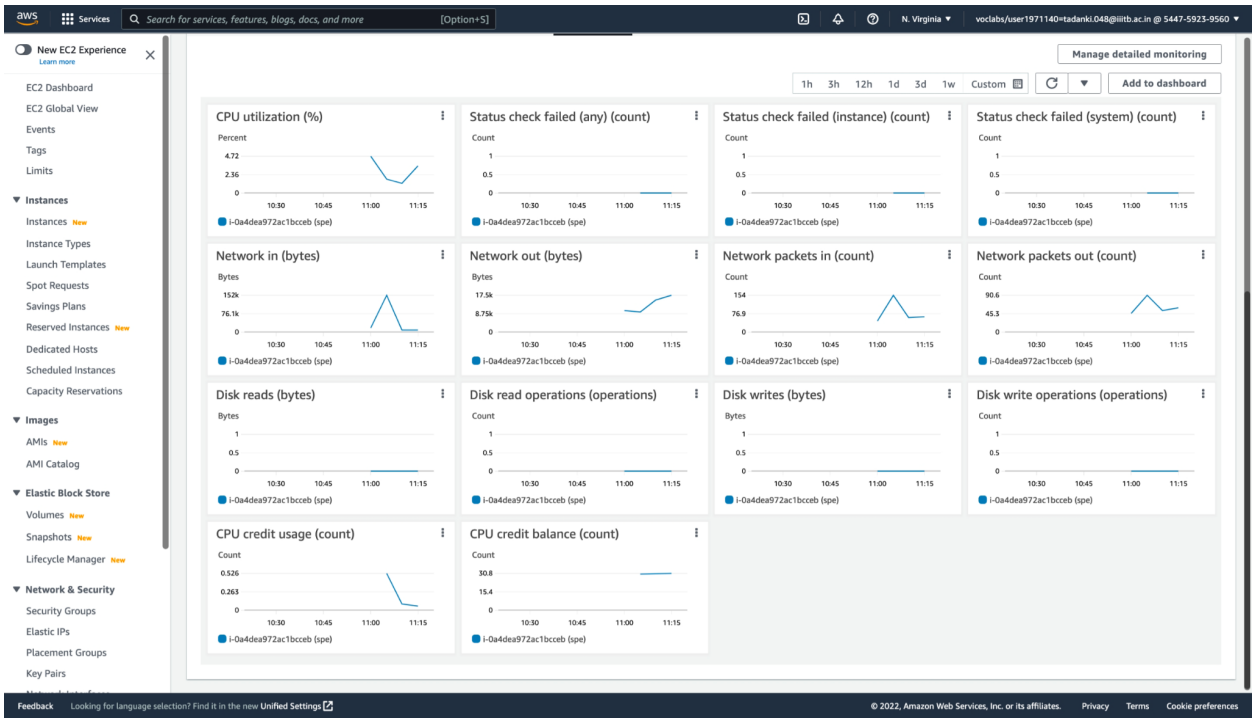
Above are two ways of visualizing the Actions.

We have also visualized the LogLevel Count of our logfile. Follow the above process to get a similar visualization. Below is a Visualization of the same.



Infrastructure and Network Monitoring with AWS:

Metrics like CPU Usage, bandwidth, latency, Disk utility and many more are provided by AWS EC2 Monitoring.



APIs and functionalities

APIs/login:

POST: When a request is sent to this api, the user will be sending a username which is an email and a password. We check the email and password in the database and if the login details match, it redirects us to the main page of our app and sends a response Code of 201 status. If something doesn't match it returns a 501 error stating the login was unsuccessful and redirects us to the same page.

APIs/signup:

POST: When a request is sent to this api, the request expects a JSON object containing details such as email, First name, Last name, Password. The server then adds the user according to the given information and sends a 200 response if successful and 500 response otherwise. If the email given is already present in the database, it gives a 400 response Status code.

APIs/forget:

POST: When a request is sent to this api, the request expects valid Google mail. If the email given is already present in the database, it gives a 404 response Status code. If the email is valid, then this redirects us to /otp.

APIs/otp:

POST: When a request is sent to this api, it means the valid email is requesting a change password. Hence, a dummy email that we created will send a unique 5 digit number.

APIs/otp/changepassword:

POST: When a request is sent to this api, it means we have provided the correct OTP that was sent to us in the email and now we are prompted to enter a new password. This value will again be updated in the database.

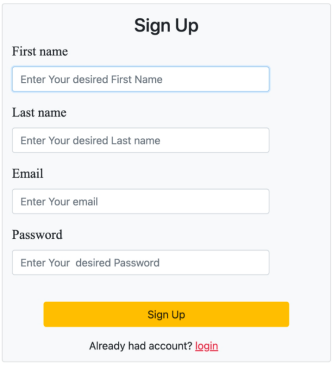
Functionalities

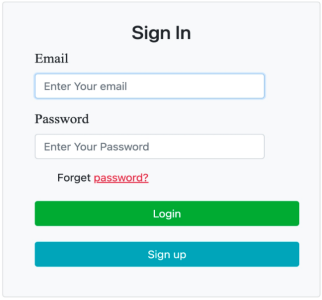
- Will be able to register, log in/out.
- An existing email cannot be registered again
- Will be able to reset password via OTP to email if forgotten
- Game instructions will be displayed on the home page
- Game difficulty can be selected(easy/medium/hard)
- Keep track of the time taken to complete the game.
- Rating will be based on no. of moves taken to complete the game.

Final views deployed from aws

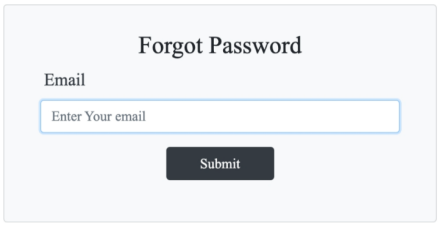
Sign Up View:

Sign in View:

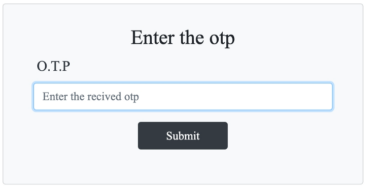




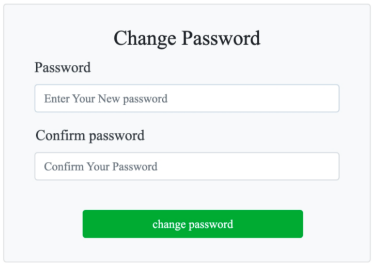
Forgot Password View:



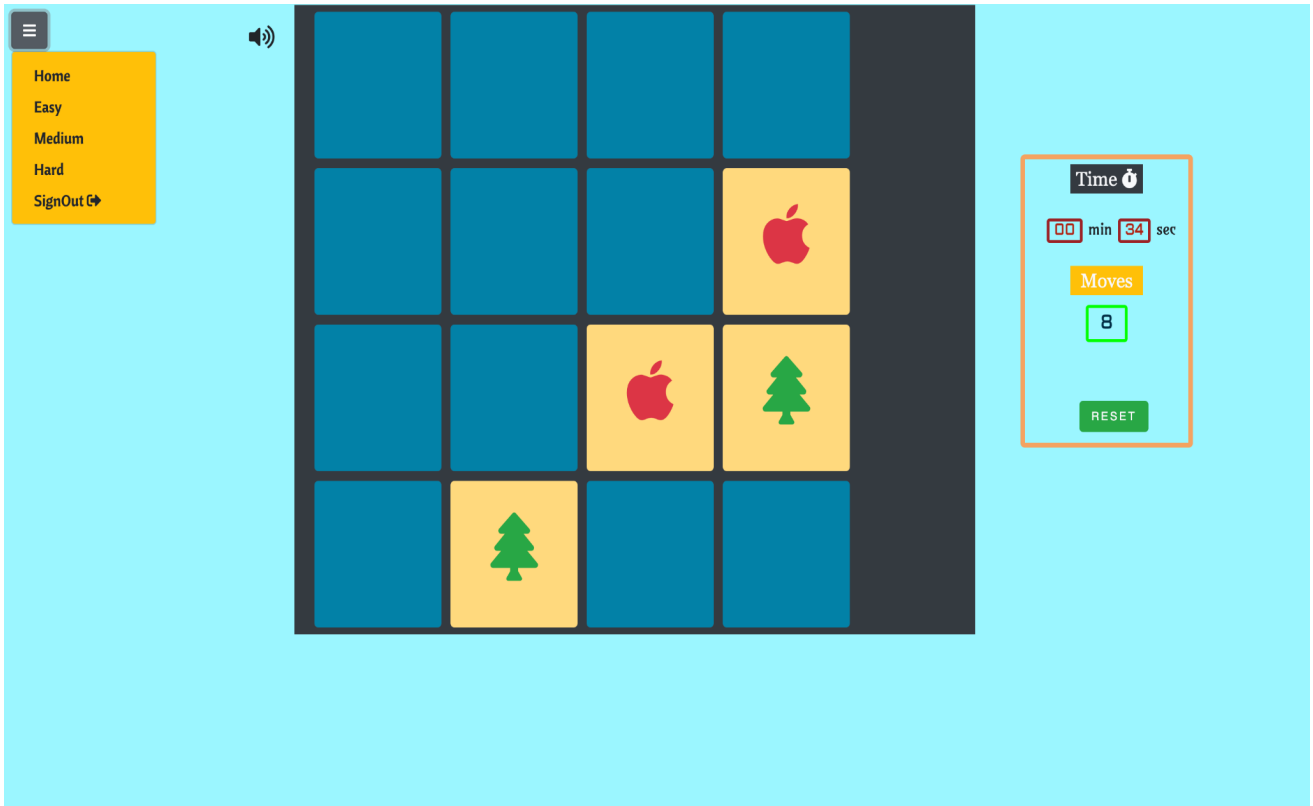
Enter otp View:



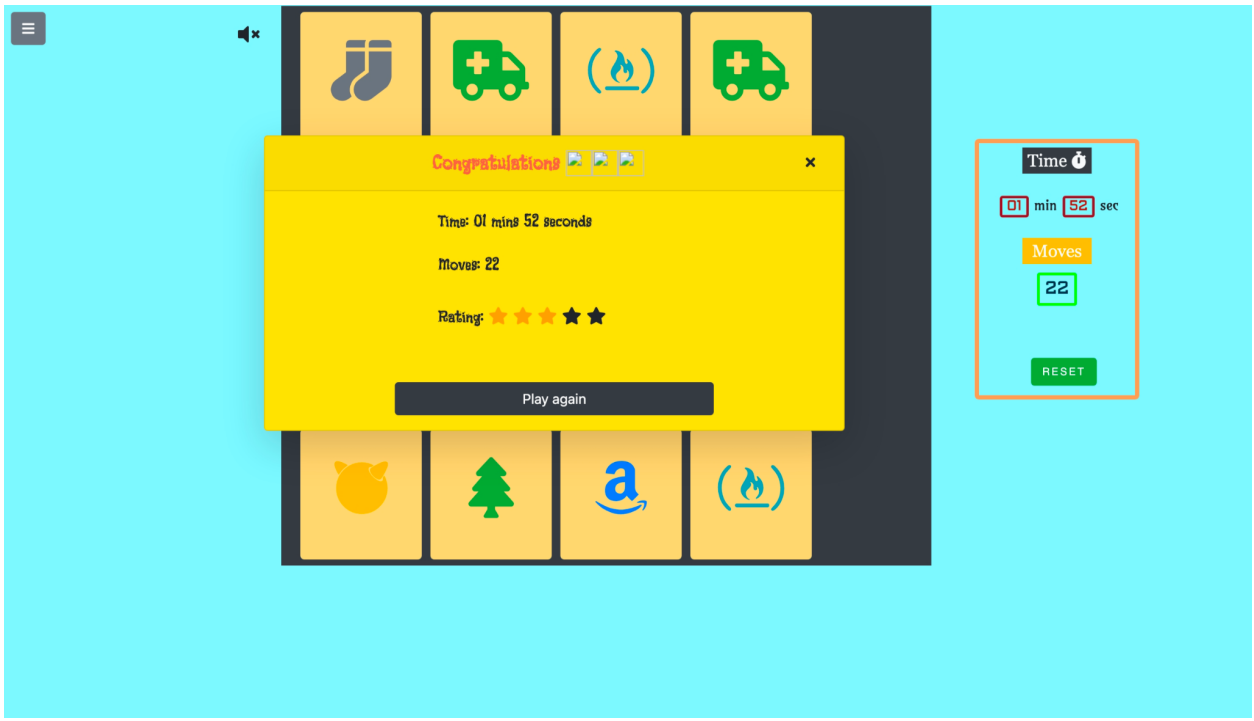
Change password View:



Game while progress View:



Game on completion View:



Game on completion View: