PRACTICAL TRAINING REPORT

Name: Ajay Vikram P
Roll N.O: 201CS204
Project Duration: 11 weeks
Organization: Qualcomm
Project Mentor: Diwakar Panchamgam

Project 1 - Diff Tool

Goal: To develop a tool that displays the difference between the before and after versions of all the files in a particular build

Motivation: Every version of a set of code released is called a build, and every change made is associated with an index called a changelist. Given the large size of each build, individuals may need to introspect where the majority of changes have come from in the particular build. A simple, practical solution is the Diff Tool, which parses the .c and .h files in the before and after versions of a build, keeps track of the number of lines added, deleted and modified for each file and displays a simple web page describing the current changelist, the files contained in them and the Lines of Code(LOC) associated with all the containers (functions, structures, unions, etc.) in the files.

Tools: Source Insight, Araxis Merge, Python, Flask, HTML, CSS, JavaScript

Working: The overall working of the tool is described as follows:

- ➢ The user calls the main Python file with the changelist of the build as an argument
- ➢ The Python script starts running
  - ○ The before and after .c and .h files of the build are fetched using Araxis Merge
  - ○ Source Insight is run to get the metadata for all the files
  - ○ Algorithm to compare before and after files and keep track of LOC is run
  - ○ Data is structured and stored in a dictionary
- ➢ The Flask Application is called
- ➢ Web Page containing the Diff between the before and after files are displayed

Advantages: The main advantage is the reduction of user effort. Tools like Araxis Merge already provide a Diff Tool, but the main disadvantage is the unavailability of a high-level summary. It only shows the code and the changes made, but not the high-level containers and which containers had the most LOC affected. To provide this high-level abstraction, the Diff Tool proves to be useful. The user no longer needs to manually perform micro-tasks to generate and analyse the Diff between 2 builds. A single Python file must be run with only the changelist number as an argument. The entire pipeline, being automated, takes a few seconds to run and automatically displays the web page at the end, showing the user the summary of the files in the build along with the LOC.

Project 2 - Memory Analyzer

Goal: To develop a tool to analyze and visualize the memory requirements of a build

Motivation: Every version of a set of code that is released is called a build. Each build has various modules and submodules containing code developed by respective teams. To realize memory requirements as to which team has contributed more to the overall increase in memory, a tool is needed. Memory Analyzer helps to provide a comprehensive summary of module level, sub-module level, file level and function level code to better analyse the memory impacts.

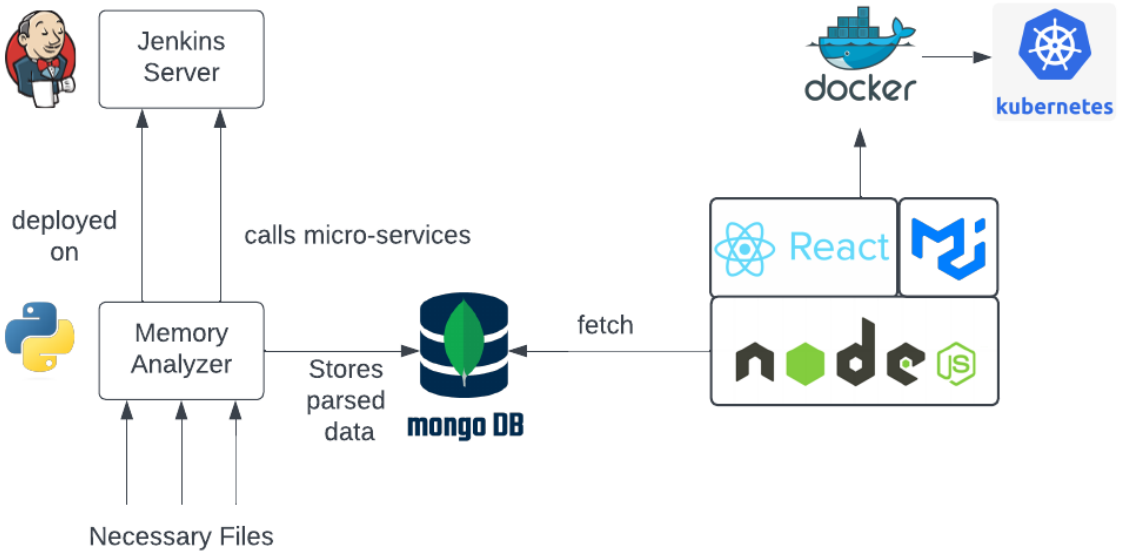
Tools: Jenkins, Python, MongoDB, ReactJS, NodeJS, Material UI, Docker, Kubernetes, Autoscope, Araxis Merge, Source Insight

Improvements: The following improvements were made to Memory Analyzer 1.0

- ➢ Shifted from local installation to cloud deployment
- ➢ Faster loading time and optimized data fetch from the database
- ➢ Provided feature for analyzing heap memory
- ➢ Improved scalability through load balancing
- ➢ Enhanced User Interface
    - ○ Graphical visualization for modules and submodules
    - ○ Lines of Code for each file (uses Diff Tool from Project 1)
    - ○ Tree-structured memory analysis for low-level data
    - ○ Option to delete builds
- ➢ Integrated User Authentication for the Web Page
- ➢ Email notification to user on memory analysis completion

Working: The main Python script for the Memory Analyzer is deployed on the Jenkins server as a downstream job. Our job is called once a user creates a build and internally calls other microservice jobs to perform related tasks to generate necessary files for memory analysis. The files, once generated, are parsed and pushed to the MongoDB database. The Web Application is developed using ReactJS and uses Material UI to enhance UI elements. NodeJS is used on the server side to communicate with the database. The Web Application, containerised using Docker, is deployed on Kubernetes, which is a container management platform. The user who created the build is notified via email once the entire pipeline is completed. The mail contains the link of the web page to view the results of memory analysis. The user is prompted to provide authentication via login and redirected to the home page of the Memory Analyzer.

Diagrammatic Flow:



QUALCOMM India Private Limited

Registered Office:
DLF Centre, 3rd Floor,
Parliament Street,
New Delhi – 110001
India

www.qualcomm.com.in

INTERNSHIP COMPLETION CERTIFICATE

This is to certify that Ajay Vikram P worked as an Interim Engineering Intern at Qualcomm from May 8th 2023 to July 28th 2023.

Ajay's performance, behavior, and conduct remains good during the internship.

We wish him all the best for his future endeavors.

July 24th 2023

Date

Aarathi Kumar

Director Staffing

21/11/2023

Other offices:
Mumbai: Unit no. 1102, Platina Building, G Block 11th Floor, Plot # C-59, Bandra Kurla Complex, Mumbai, India 400051 India. Tel: +91-22-67041400; Fax: +91-22-67041500
Hyderabad: 5th Floor, Bldg # 8 Mindspace, Hitec City, Madhapur, Hyderabad, 500 081, Andhra Pradesh, India. Tel: 91-40-3011-0000; Fax: 91-40-3011-0001
Bangalore: Plot no. 125-127, EPIP IInd Phase, Whitefield, Bangalore, Karnataka 560 066, India. Tel: 91-80-3984-1800; Fax: 91-80-3984-2001