e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

COMPREHENSIVE STUDY OF ONLINE CODE EXECUTION SYSTEM USING CONTAINERIZED ENVIRONMENT

Prathmesh Shelar*1, Amit Potghan*2, Sami Shaikh*3, Mayur Saibane*4, Datta Kale*5, Prof. Priyanka R. Balge*6

*1,2,3,4,5Department Of Computer Engineering, JSPM Narhe Technical Campus, Pune, Maharashtra, India.

*6Prof. Department Of Computer Engineering, JSPM Narhe Technical Campus, Pune, Maharashtra, India.

ABSTRACT

As the demand for interactive learning and coding platforms continues, online code execution systems have become important tools in education, coding competitions, and software development. This paper provides a comprehensive review of an online code completion system designed to provide users with a secure, efficient, and scalable environment for writing, testing, and executing code in a variety of programming languages. We discuss the design of the system, focusing on key components such as code submission, execution engine, and return results. The main focus is on security mechanisms, including sandboxing technology to prevent malicious policies and control strategies to ensure the integrity of multiple users. Evaluate performance metrics that demonstrate the system's ability to handle concurrent transmissions while maintaining low latency and high throughput. The user experience is also delivered through an intuitive interface that facilitates code editing, instant feedback, and access to test data.

Keywords: Online Code Compilation, Sandboxing, Programming, Execution Engine, Coding, Programming Language.

I. INTRODUCTION

In recent years, online code execution systems have become pivotal for developers, educators, and learners, providing a platform to write, run, and test code remotely. These systems leverage containerized environments, enabling secure, scalable, and efficient code execution across various programming languages and frameworks. By incorporating advanced technologies such as Docker, Kubernetes, and virtualization, these platforms facilitate an isolated and controlled environment where users can execute code seamlessly within local and global networks.

This study focuses on designing and optimizing an online code execution platform specifically for JavaScript and Next.js environments, underpinned by Docker for containerization, PostgreSQL for reliable database management, and Prisma for a streamlined data layer. The proposed system will support a variety of use cases, from learning and prototyping to production-grade testing, by allowing users to interact with code in real-time through virtual machines (VMs) that simulate actual network conditions and development environments. Using Kubernetes for orchestration, the platform is built to efficiently handle multiple concurrent user requests, dynamically allocate resources, and ensure high availability and fault tolerance.

The architecture of this system is designed to address the security, scalability, and resource management challenges inherent in online execution platforms. We will explore how container orchestration and database management practices can be optimized to deliver an efficient and robust solution. This paper focuses on study of working and architecture of one of such code execution engine Judge0.

Judge0 allows programmers to setup an execution engine that evaluating a certain code against multiple test cases defines in database by the programmer. This paper discusses the implementation and use of a system for code evaluation hosted online through the help of Judge0.

II. METHODOLOGY

1. Judge0

Judge0 (pronounced as "judge zero") is a robust, scalable, and open-source online code execution system. We use it as our primary judge of code where each file is executed and tested against each test case by Judge0. The test cases are stored in SQL database and it is pipelined through Prisma. The database is only used during primary testing for working of Judge0 then it becomes obsolete as the size of test cases increases and hinders and obstructs our performance. So, it is suggested that for full scale deployment use of cloud platforms such as Amazon's AWS S3 (Object Store) to store and interact with test cases.

2. Web Application Framework

This particular implementation of the Judge0 using next.js. We implement next.js for both frontend as well as backend of the program. It allows for highly interactive webpages and Tailwind allows styling of these pages. The webapp (short for Web Application) uses PostgreSQL for database and Prisma as an ORM (Object-Relational Mapping) tool for pipelining database connectivity. To evaluate program through Judge0, we send multiple requests to Judge0 that contain user program and sets of inputs for program and expected outputs. Once the request is under processes, code is evaluated so that if the actual output of code matches with expected output for each test case separately then the submission is approved otherwise denied.

3. Docker

Docker is essential to this online code execution platform because it makes it possible to run user-submitted code in isolated, secure environments. Every user's code executes in a separate container, offering a sandboxed environment that improves security and guards against interference. Through the packaging of dependencies and configurations required for every programming language, containerization guarantees consistency across executions. Because containers can be spun up and down in response to demand, Docker also makes resource management more effective. The platform can scale easily to accommodate different levels of user activity thanks to Docker's integration with orchestration tools like Kubernetes. Stability and responsiveness are essential for a high-performance code execution system, and this method guarantees both.

4. SQL Database (PostgreSQL)

PostgreSQL is an essential component of this online code execution platform because it serves as a dependable and robust relational database management system for data storage and management. It is in charge of managing problem data, code submissions, user information, and execution outcomes. Because of PostgreSQL's robust JSON support, which facilitates relational data modeling and makes it simpler to store structured data like code snippets and test cases, this platform favors it. With support for big datasets and multiple users at once, PostgreSQL is also very scalable, making it appropriate for an expanding user base. It also works well with Prisma, a contemporary ORM, which simplifies database operations and facilitates data access in Next.js and Node.js environments.

We do have to take the limitations of a SQL database in consideration and when implementing the system at scales where the test case keeps on increasing this slows the speed of transactions that a SQL database is capable of providing. Here online storage solutions simply outclass the performance provided by PostgreSQL.

5. Prisma

Prisma is the ORM (Object-Relational Mapping) layer in this online code execution platform, which streamlines database interactions and improves data consistency. Prisma, which serves as a bridge between application code and PostgreSQL, allows for rapid querying, data validation, and type-safe operations, all of which are especially valuable for complicated data pipelines such as user profiles, code submissions, and execution outcomes.

Prisma's intuitive schema-based modeling makes it simple to define and manage database structures, as well as alter data models as the platform evolves. As a robust ORM, Prisma allows for the pipelining of data-related processes, ensuring that database interactions are clean, quick, and free of errors. It also supports migrations.

6. Virtual Machines (VMs)

Virtual machines (VMs) constitute the backbone of this online code execution technology, providing greater security, flexibility, and scalability. They serve as the foundation for Docker containers, offering an additional layer of isolation between the underlying hardware and the user code. This configuration assures that, even if a vulnerability exists in the containerized environment, the host system is protected by the VM's isolation layer.

The virtual machines have an impact on the project because they provide a stable and adaptable environment in which different operating systems and configurations can be operated concurrently, as well as support for a variety of programming languages and runtime environments. This adaptability is critical for managing different code execution requirements in a controlled and predictable manner. Furthermore, VMs provide better resource allocation because they can be scaled up or down based on demand, allowing the platform to adapt to changing workloads. By utilizing VMs, the project strikes a compromise between performance and security, delivering a dependable user experience for both learners and professionals.

As the VMs provide a more recoverable environment we setup all essential runtimes as well as our Judge0 engine on an Ubuntu VM. Ubuntu is optimal for performance of Judge0 and if during a cyberattack any harm is caused to the Operating System (OS) of VM it is highly recoverable. Also, hosting database and Judge0 on separate machines allow for an increased layer of insulation against attacks.

7. Local/Global Network

The flexibility of Judge0 allows us to either host the webapp locally or globally. What it means is that you can either host the execution engine on small device and only allow a very small number of users to interact with it or host it on larger networks with competently capable devices or host on cloud platform, and make a permanent website that is available all day long.

8. Kubernetes

Kubernetes is critical to the scalability and stability of the Judge0-based online code execution platform. Kubernetes is a container orchestration system that automates the deployment, scaling, and maintenance of containerized applications, making it perfect for dealing with the dynamic nature of code execution services. When user demand changes, Kubernetes can automatically scale the platform by adding or removing containers based on the amount of code contributions, guaranteeing efficient resource use without the need for operator intervention. This scalability is critical for sustaining performance during peak traffic periods, as Kubernetes can spread traffic evenly across numerous instances while preventing any single instance from becoming overwhelmed. Kubernetes further increases the platform's resilience by including capabilities such as self-healing and fault tolerance. If a container dies, Kubernetes may immediately restart or replace it, resulting in minimal downtime and uninterrupted operation. The platform can be deployed across numerous servers or virtual machines, allowing Kubernetes to efficiently manage resources in a distributed environment while optimizing CPU and memory use.

III. MODELING AND ANALYSIS

The model for webapp follows Figure 1. The figure represents the working and handling of requests where multiple Judge Bots (workers) process multiple requests that Judge0 receives. The workers function as independent threads that judges individual cases for a program that is under process.

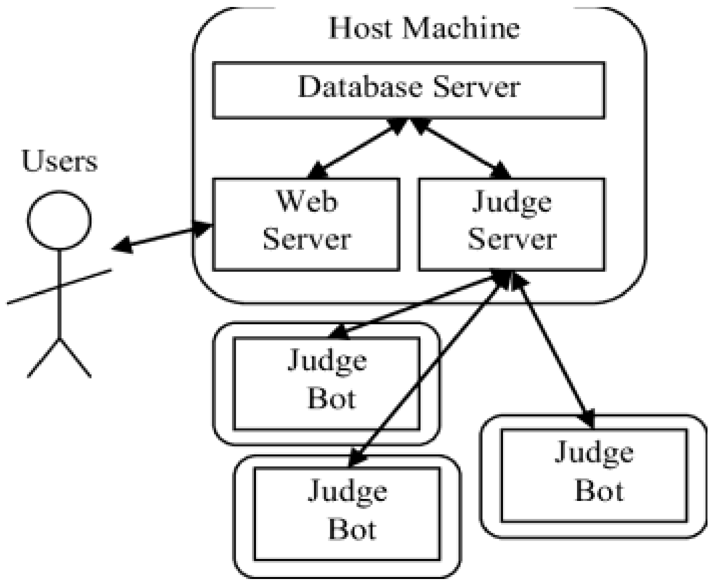


Figure 1: Overview of Judge0 Execution Engine.

The next figure shows the mode of communication between the web server and Judge0 execution engine.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

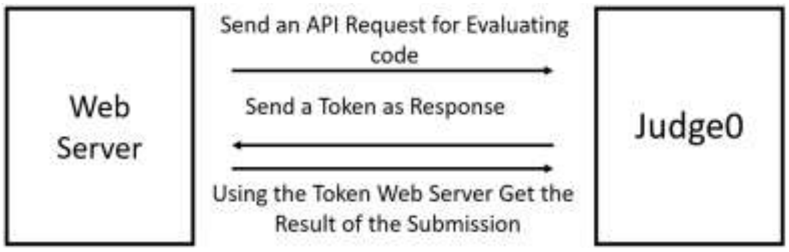
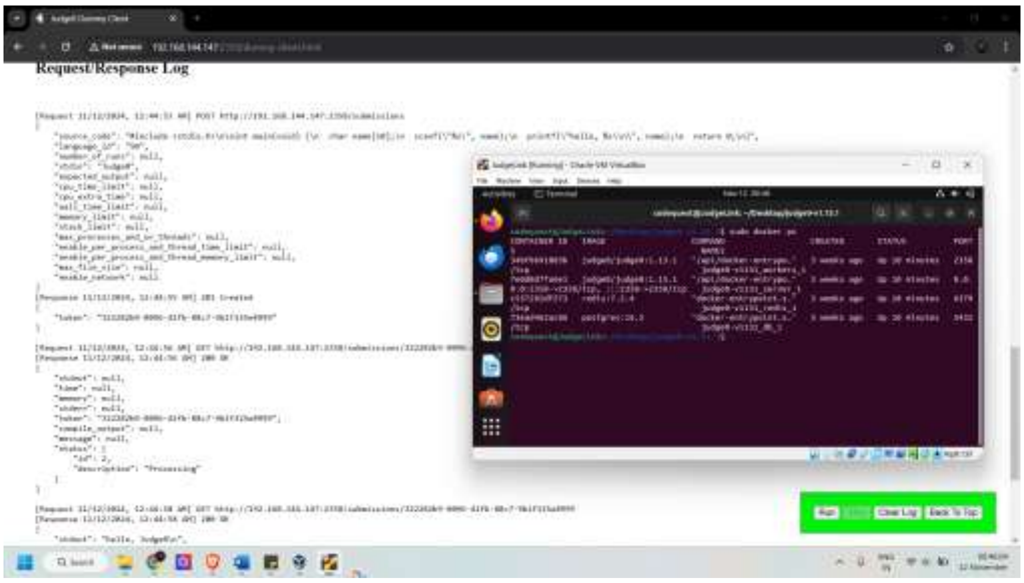

Web Server
Send an API Request for Evaluating code
Send a Token as Response
Using the Token Web Server Get the Result of the Submission
Judge0

Figure 2: Communication between Judge0 and Web Server

IV. RESULTS AND DISCUSSION

During the paper we discussed the working and functionality of online code execution platforms. We also discussed on how we can implement these systems. These webapps could entirely be hosted on a single machine and operate from one machine for all its locally connected devices. Following are the figures of complete implementation of such one system.



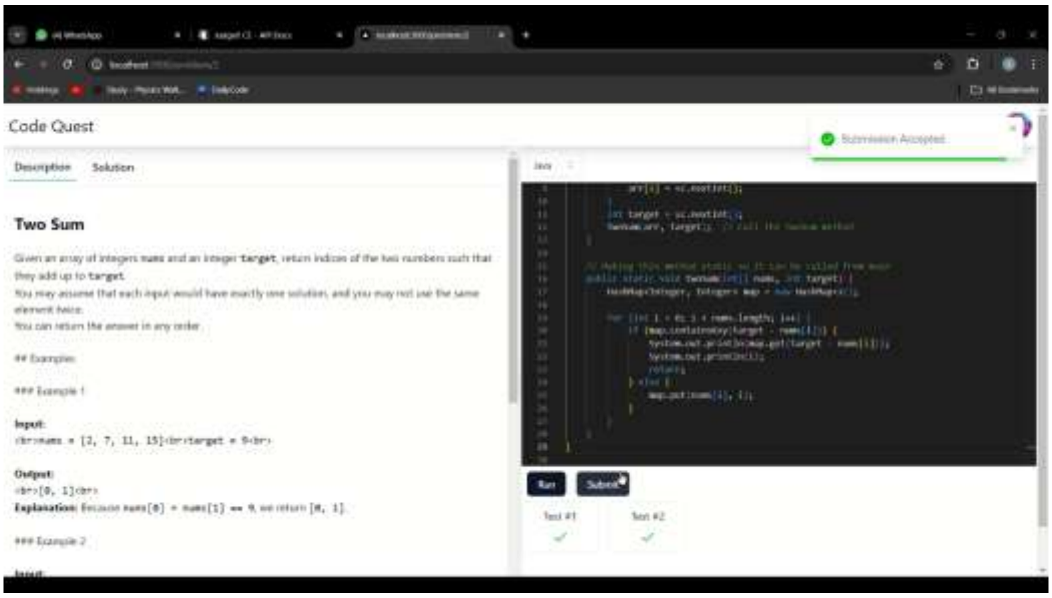
Request/Response Log

```
[Request [illegible] POST http://[illegible]/submissions
{
    "source_code": "#include <stdio.h>\n\nint main(void) {\n  char name[10];\n  scanf(\"%s\", name);\n  printf(\"hello, %s\n\", name);\n  return 0;\n}",
    "language_id": "50",
    "number_of_runs": null,
    "stdin": "Judge0",
    "expected_output": null,
    "cpu_time_limit": null,
    "cpu_extra_time": null,
    "wall_time_limit": null,
    "memory_limit": null,
    "stack_limit": null,
    "max_processes_and_or_threads": null,
    "enable_per_process_and_thread_time_limit": null,
    "enable_per_process_and_thread_memory_limit": null,
    "max_file_size": null,
    "enable_network": null
}
[Response [illegible] 201 Created
{
    "token": "[illegible]"
}

[Request [illegible] GET http://[illegible]/submissions/[illegible]
[Response [illegible] 200 OK
{
    "stdout": null,
    "time": null,
    "memory": null,
    "stderr": null,
    "token": "[illegible]",
    "compile_output": null,
    "message": null,
    "status": {
        "id": 2,
        "description": "Processing"
    }
}

[Request [illegible] GET http://[illegible]/submissions/[illegible]
[Response [illegible] 200 OK
{
    "stdout": "hello, Judge0\n",
```

```
CONTAINER ID   IMAGE   COMMAND   CREATED   STATUS   PORTS   NAMES
[illegible]
```

Figure 3: Judge0 working on a Virtual Machine when Web Server runs on physical hardware

From Figure 3 we can conclude that it is possible to build the web application on a single machine the next figure shows more prominent utilization of the online code execution system ready for deployment.



Two Sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.
You may assume that each input would have exactly one solution, and you may not use the same element twice.
You can return the answer in any order.

Examples

Example 1

Input:

nums = [2, 7, 11, 15]
target = 9

Output:

[0, 1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2

```java
            arr[i] = sc.nextInt();
        }
        int target = sc.nextInt();
        twoSum(arr, target); // Call the twoSum method
    }

    // Making this method static so it can be called from main
    public static void twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(target - nums[i])) {
                System.out.print(map.get(target - nums[i]));
                System.out.println(i);
                return;
            } else {
                map.put(nums[i], i);
            }
        }
    }
}
```

Figure 4: Working implementation of Judge0 (Online code judging program)

V. CONCLUSION

This project demonstrates the creation and deployment of an online code execution system in containerized settings, utilizing Docker, PostgreSQL, Prisma, Kubernetes, and Judge0 for best performance, scalability, and security at initial stages. The system is intended to address the demands of both educators and students of coding by offering an interactive, efficient, and secure environment for coding practice and execution. Docker isolates and manages execution environments, guaranteeing that each user's code executes independently and without interference. PostgreSQL, along with Prisma as the ORM, provides a strong and efficient database solution that supports data integrity and complicated queries, hence improving the backend architecture.

Kubernetes improves scalability and fault tolerance, allowing the system to manage a large number of concurrent users, which is critical for both large educational institutions and smaller online coding platforms. Judge0 provides a dependable and safe online code execution environment, with real-time feedback for user code contributions. A virtual machine infrastructure supports both local and global network setups, providing consistent access and performance across several locations.

This project not only fits current requirements for online code execution, but it also lays the groundwork for future additions such as sophisticated monitoring, a chance for parallel programming, automated scaling and improved user administration. Our system integrates technologies to create an accessible and productive learning environment for users at all levels, adapting to the ever-changing online education and coding scene.

VI. REFERENCES

[1] Došilović, Herman Zvonimir & Mekterovic, Igor. (2020). Robust and Scalable Online Code Execution System. 1627-1632. 10.23919/MIPRO48935.2020.9245310.

[2] Mizuno, Takahisa & Nishizaki, Shin-ya. (2014). Distributed Online Judge System for Interactive Theorem Provers. EPJ Web of Conferences. 68. 10.1051/epjconf/20146800016.

[3] Porubän, J.; Nosál', M.; Sulír, M.; Chodarev, S. Teach Programming Using Task-Driven Case Studies: Pedagogical Approach, Guidelines, and Implementation. Computers 2024, 13, 221.
https://doi.org/10.3390/computers13090221

[4] Lindberg, R.S.N., Laine, T.H. and Haaranen, L. (2019), Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. Br J Educ Technol, 50: 1979-1995. https://doi.org/10.1111/bjet.12685.

[5] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. 2018. "A Survey on Online Judge Systems and Their Applications." ACM Comput. Surv. 51, 1, Article 3 (January 2019), 34 pages. https://doi.org/10.1145/3143560.

[6] X. Lu, D. Zheng and L. Liu, "Data Driven Analysis on the Effect of Online Judge System," 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Exeter, UK, 2017, pp. 573-577, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.90.