International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03885
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Cloud IDE: A Lightweight, Secure, and Open-Source Remote Development Platform

Priti Routh

Electronic and Communication

Institute of Engineering and Management

Kolkata, West Bengal, India

pritirouth535@gmail.com

Chirasmita Deb

Electronic and Communication

Institute of Engineering and Management

Kolkata, West Bengal, India

chirasmitadeb@gmail.com

Ratna Chakrabarty

Electronic and Communication

Institute of Engineering and Management

Kolkata, West Bengal, India

ratna.chakrabarty@iemedu.com

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03885
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Abstract— Existing cloud IDEs offer remote development capabilities but require users to contend with expensive setup, licensing charges and code storage insecurity. Cloud IDE resolves these issues by allowing an employee to connect to powerful employer computers over the internet through a web browser without the issues of high setup complexity, licensing fees, or the security ramifications of storing code.

Keywords—Cloud, Node. js, IDE's, Dev Tooling, Docker

I. INTRODUCTION

The growth of cloud computing has ushered unparalleled change in software development, the introduction of Cloud-Based Integrated Development Environments (Cloud IDEs) being an example. On such platforms, developers can create, test, and deploy code straight from the web browser, removing the pain of local settings and facilitating teamwork from distant parts. Cloud Integrated Development Environments (IDEs) have fundamentally changed the use of tools and resources by developers, providing solo and team developers a simple accessibility to use, at their own expense.

The creation and maintenance of local environments were the biggest challenges of software development, which were constantly facing compatibility, resource limitation or cooperation hurdles. The manually coded procedures of building compilers, debuggers, and version control systems were tedious and highly variable with varying development styles [1, 2].

By consolidating leading development tools within one browser-based environment, cloud IDEs address these challenges. The configurations consist of code editors, debuggers, version control systems, deployment tools, and cloud-based scalable infrastructure. The fact that developers can work in real time and access their environment from any connected device is a major benefit since it boosts flexibility.

The presence of Cloud IDEs in schools has provided the coding process to all the students, and programming environments are easily accessible irrespective of hardware. With such platforms being used in business environments, teams are able to collaborate and track projects remotely, with integrated development environments.

Auto-scaling and resource optimization capabilities of cloud computing make it possible to run resource-consuming tasks, like compilation or simulation of code, without overloading local computers [7][8]. The programmers can focus on coding instead of infrastructure.

But problems still exist. Though they have value, cloud IDEs can still become a security risk when deployed to run proprietary code on third-party servers, which need stable internet connections. Furthermore, commercial IDEs will

likely have usage limitations or fees that are not compatible with large companies [3][4].

To overcome these limitations, this paper presents Cloud-ID, an open-source self-hosted Cloud IDE. Cloud-ID is designed to leverage the advantages of cloud development while providing stronger security, control, and personalization. Cloud-ID provides infrastructure and data control, and thus it is apt for organizations seeking a secure, scalable, and collaborative development environment [2].

This research examines the Cloud-ID model, its application, and benefits, suggesting its capacity to revolutionize how coders engage with code in cloud procedures.

II. LITERATURE REVIEW

Software development has been affected by cloud computing to a great extent with the introduction of Cloud-Based Integrated Development Environments (Cloud IDEs). Cloud IDEs remove the need for local development and transfer it to the cloud, which is easier to set up and enhances collaboration work[1][7]. Technologies of containerization and virtualization have also increased these trends to a large extent[2].

Expansion of Cloud IDEs -

The first implementations like Cloud9 (now AWS Cloud9)[7] had built-in terminals and editors, thereby making web-centric development possible. It was later followed by Replit[3] and CodeSandbox, who built upon it by introducing multi-language execution and live collaboration. Front-end development was further supported by StackBlitz by enabling Node.js execution in-browser via WebAssembly[4].

While these tools improve the ease of access and flexibility, some of them create minimal but constricting usage restrictions, vendor lock-ins, or extra costs[10]. GitHub Codespaces, for example, is a powerful tool, but charges users who do not stay within their ecosystem after a grace period.

Security and Privacy Issues -

Data privacy issues are among the top concerns regarding Cloud IDEs. Cloud storage of code and sensitive resources allows access by unauthorized entities which makes it highly prone to breach or leakage[1]. AWS Cloud9[7] and Google Cloud Shell[8] offer robust security; however, certain companies opt for self-hosted environments to retain full control over their infrastructure.

Rise of Open-Source IDEs-

Open-source options such as Theia[11] and Eclipse Che[12] offer self-hostable, customizable solutions without vendor lock-in. Theia, for example, has a modular design and VS Code-like experience, driving products such as Gitpod.

The Road Ahead-

Cloud IDEs are transforming to incorporate automation, AI-based coding, and more extensive DevOps integration[9].With more organizations embracing cloud-first, secure, customizable, and open-source Cloud IDEs—such as cloud-id - demand will increase

III. METHODOLOGY

The methodology part of this paper describes the approach, design, and methods of studying the architecture, implementation, and operation of Cloud-ID, an open-source Cloud IDE. The process consists of three main phases: system design, implementation, and testing. Each phase is discussed in detail to gain an overall idea of the development process, motivation behind design choices, and technologies employed.

1. System Design

The architecture of Cloud-ID is rooted in the central philosophy of delivering an open-source, extensible, and secure Cloud IDE capable of addressing both individual developers and organizational needs[9]. The infrastructure is designed flexibly as the focus point, where the system can be customized based on user interface, functionality, and integration with third-party tools and services.

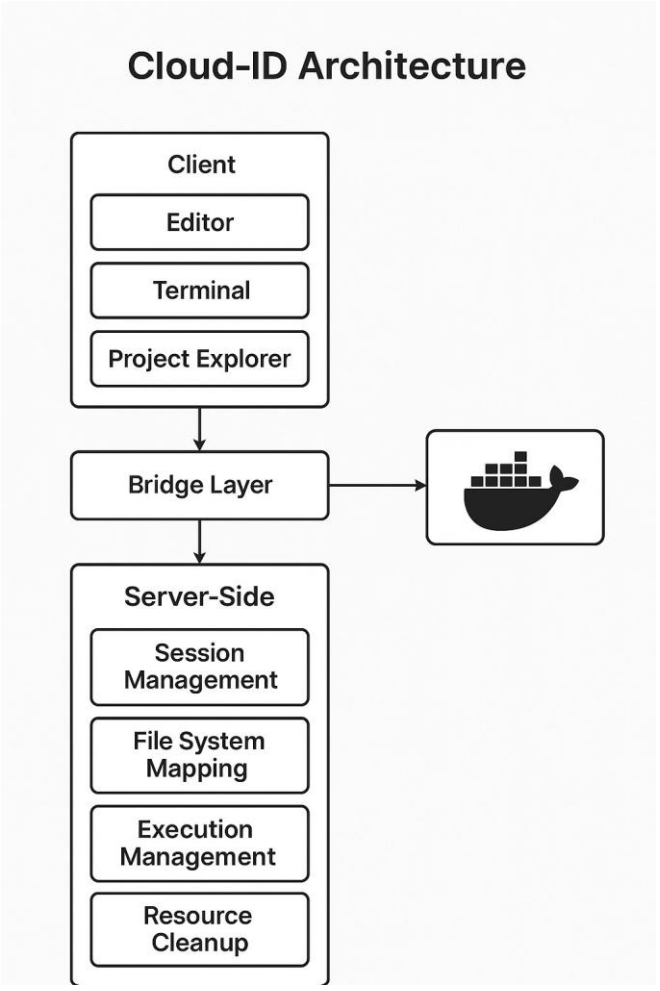

fig.[1]

1.1. Architecture

Cloud-ID uses a microservice architectural style with system partitioning into small components, which each can be independently developed and maintained. Key benefits include independent deployment and update of each component, greater agility in development and operations. These components include:

Frontend (User Interface): The UI is implemented using React.js. It also utilizes WebSockets enabling real-time interaction. The frontend deals with commands from users, creates the rendering of the development environment, and calls corresponding services in the backend using REST APIs and WebSockets.

Backend (Core Services): The backend comprises several specialized microservices that handle essential IDE operations. These include editing, compiling, debugging, and version control. Each service is responsible for a distinct function such as running code, managing repositories, or performing debugging.

Isolated Development Environments: Cloud-ID has employed the use of Docker containers for managing user sessions to ensure consistency and security. Each developer is provided with a dedicated container environment which guarantees that workspaces are isolated, reproducible, and dynamic in scale based on demand.

Integration with Cloud Storage: The system works in conjunction with cloud storage systems such as AWS S3 and Google Cloud Storage, allowing management and storage of user files, configurations, and logs. This ensures that users can access their projects from multiple systems while their progress remains stored in the cloud.

1.2. Why Security is Important

Because of the dangers involved with running and storing user code on remote servers, security is one of the main features of Cloud-ID's infrastructure. There are multiple layers of security that the platform uses.

End-To-End Encryption: Every communication between frontend and backend services and the cloud is protected under HTTPS and WSS (WebSockets Secure) protecting delicate information like code and login details during transmission.

Authentication & Authorization: Storing passwords becomes redundant because OAuth2 allows secure logins through verified third-party providers such as Google, GitHub, and GitLab. Resource restriction is controlled through role-based authorization which permits or limits resource access based on user status.

Data Isolation: Each session is allocated a Docker container which means user data is not exposed to the public, ensuring privacy. Additionally access to the cloud storage is extremely limited making data remains confidential and secure from unauthorized users.

2. Implementation

The development of Cloud-ID is done by following agile practices which allow for change and constant improvement. The overall development process can be broken down into three major activities: initial setup, feature implementation, and testing.

2.1. Initial Setup

During the setup phase, the development team made the following selections to aid in the creation of a robust scalable system:

Frontend: The modern architecture of component reuse in React.js guarantees the implementation of the client using this library. Moreover, WebSocket protocols will be used to facilitate instant collaboration and feedback between the server and client.

Backend: The service type will be implemented using Node.js and Express.js as they allow for the asynchronous, event-driven, and non-blocking model of service request handling.

Containerization: User environments were Dockered to provide reproducibility, isolation, and consistency throughout development sessions and across platforms.

Cloud Infrastructure: Both AWS and Google Cloud services were utilized. Amazon S3 served as a dependable file storage utility while Google Cloud's Kubernetes Engine took care of the backend's containerized services' management and orchestration.

2.2. Feature Implementation

With the foundation in place, the core functionality and collaborative features of the platform were implemented:

Code Editor: Based on Microsoft's Monaco Editor, the code editor supports multiple languages with features like syntax highlighting, autocompletion, and error detection.

Compiler & Debugger Integration: Each user's code is compiled and debugged inside an isolated Docker container, ensuring a clean and secure runtime environment.

Version Control: Git-based version control was integrated, allowing users to perform repository operations (clone, commit, push, etc.) with platforms like GitHub and GitLab directly within the IDE.

Real-Time Collaboration: Users can collaborate on the same project in real time. Using WebSockets, any changes made by one user are instantly reflected across all active sessions.

Cloud File Storage: Project data is stored in AWS S3 or Google Cloud Storage, allowing users to save, retrieve, and access files securely from any location.

2.3. Testing

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03885
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Ensuring the Cloud-ID platform's reliability necessitated a multi-layered testing approach throughout the development cycle. A combination of various testing methodologies was utilized to ensure the stability of the platform and each component's correct operation under different conditions:

Component-Level Verification: All microservices, such as the code editor, compiler engine, and versioning module, were verified separately with Mocha and Chai (Node.js). Low-level tests validated that standalone pieces were functioning as expected and satisfied design requirements.

System Integration Validation: For proper collaboration among modules, integration tests were employed to test the flow of data and logic among the user interface, backend services, and cloud infrastructure. This ensured that components functioned as expected when integrated.

Simulated User Behavior Testing: End-to-end workflow scenarios—e.g., authentication, project creation, editing, and code compilation—were tested using automated tools like Selenium. Such simulations mimicked end-user activity to verify that the application operated as anticipated within actual use cases.

Scalability & Performance Tests: Through the use of tools such as Apache Jmeter, the system was tested for high user concurrency. The team tested for response consistency, load tolerance, and resource capacity under such a scenario to gauge the platform's readiness for mass consumption.

3. Evaluation

After development, a comprehensive evaluation phase was conducted to determine both the practical usability and operational performance of Cloud-ID. The evaluation employed a mix of empirical data and user-led insights:

Qualitative User Assessment: Feedback was collected from early users using structured interviews and questionnaires. This measured how intuitive, functional, and satisfying the platform was from the user's point of view, identifying areas for improvement.

Operational Performance Monitoring: Real-time performance of the platform—e.g., system uptime, latency, and resource utilization—was monitored through observability stacks such as Grafana and Prometheus, which facilitated continuous improvement in service efficiency.

Competitor Benchmarking Research: Cloud-ID features were compared with leading alternatives including Replit, AWS Cloud9, and GitHub Codespaces using metrics such as startup performance, feature depth, security strength, and general user opinion, with the intention of determining Cloud-ID's position within the market as well as identifying areas to develop further.

IV. IMPLEMENTATION

Cloud-ID functions as a set of decoupled modules working together to provide a responsive, secure, and user-friendly development environment. The platform is designed with a modular architecture that enables independent scaling, easy maintenance, and replacement of components without affecting the entire platform.

1. Backend Infrastructure

Central to Cloud-ID is its backend framework, which orchestrates the key services that manage containers, user sessions, code execution, and file persistence.

Session Orchestration:

When a user starts a session, the backend creates a fresh Docker container from a standardized base image. This image is already pre-loaded with a Linux distribution and pre-installed with development tooling like Node.js, Python, Java, Git, Vim, and typical build utilities. Each container is run within an isolated user namespace, giving complete runtime isolation to ensure secure, individualized workspaces.

File System Mapping:

To ensure continuity across sessions, each user is assigned a dedicated Docker volume. This volume is attached to the container and retains project files, configuration preferences, and temporary data. As a result, users returning to the environment after disconnection or reboot will find their workspace intact, just as they left it .

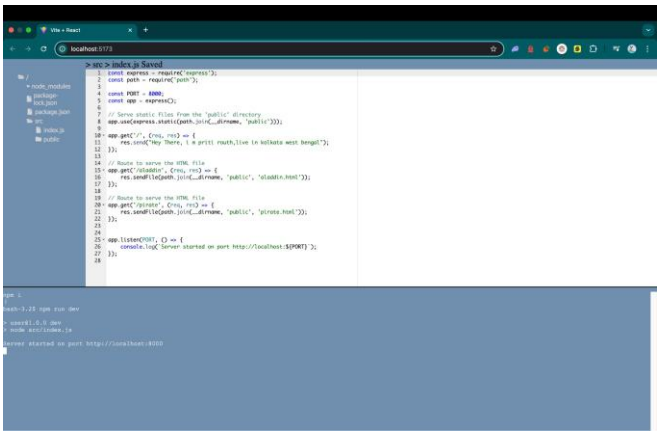

fig.[2]

Execution Management:

Commands typed via the browser-terminal are sent to the backend, which in turn runs them within the user's container. Outputs (standard output and standard error) are read in real-time and streamed back to the user. Synchronous command execution and interactive sessions are both supported[2]]6].

Resource Cleanup:

Containers are memory-hungry if not controlled. Thus, an inactivity timeout feature is introduced. A container is

forcefully terminated and unloaded if it happens to be idle (no current WebSocket[5] connection or command execution) for more than a configurable amount of time (e.g., 30 minutes).

2. Bridge Layer (Real-Time Communication Server)
The bridge layer plays the role of middleware between frontend client and backend containers, focusing mainly on real-time bi-directional communication[5].

WebSocket Management:
A secure WebSocket (WSS) server is employed to maintain persistent, low-latency connections with the frontend[5]. Each connection is authenticated using JWT (JSON Web Token) tokens generated upon login. This ensures that only valid users can communicate with their corresponding sessions.

Session Mapping:
Each user's socket connection is associated with their own Docker container session. Upon receiving a message (command, file edit, terminal input) through WebSocket, the bridge server sends it appropriately, maintaining isolation between user sessions[5].

Load Handling:
For improving scalability, the bridge server is made stateless wherever possible, so multiple instances of the bridge can be run behind a load balancer, giving horizontal scaling capability for large-scale deployments[7][8].

3. Client-Side (Frontend)
The client-side application offers an intuitive, responsive, and light-weight user interface for users to interact with their development environments[4].

Editor:
The frontend's backbone is the built-in code editor, which is achieved through powerful libraries like Monaco Editor (the same editor used by Visual Studio Code) or CodeMirror[10][11]. The editor can include features like syntax highlighting, autocompletion, error checking, and multi-file editing.

Terminal:
An embedded, browser-based terminal emulator (i.e., xterm.js) is included, allowing users to execute shell commands directly within their container from their browser[6]. This offers an uncomplicated, near-native developer experience without the need for local software installations.

Project Explorer:
The project explorer UI shows the user's file system structure, supporting typical file operations such as making, removing, renaming, and moving folders/files. Changes are real-time synced with the backend volume storage[2][4].

Session Management UI:
Login and authentication screens handle user identities. After successful authentication, the system restores a previous session or creates a new one. The UI also shows session status (e.g., active, disconnected) and provides actions such as disconnecting or resetting the environment[10].

4. Security Measures
Security is a fundamental cornerstone of Cloud-ID's design, and several layers of protection are integrated into each module.

Secure Communication:
All frontend-backend traffic is exchanged using WSS (WebSocket Secure) connections, secure traffic encryption and avoiding man-in-the-middle attacks[5].

Isolated Networking:
Each container runs on its own isolated virtual network. Users' containers can't be accessed or seen by other users unless specifically allowed for, avoiding lateral movement during a potential breach[2]]7].

User-Specific Permissions:
Role-Based Access Control (RBAC) is enforced, limiting what a user can do based on the role they are playing (e.g., admin versus standard user). Every user session is tied to their identity[9][10], so they only have access to their own workspace.

File Storage Security:
User volumes are encrypted when stored, so data is still safe even if server disks are breached[7][8].

Audit Logging:

Significant activities like login attempts, container creation, and administrative activities are logged centrally for security audit and forensic purposes[9].

5. Optimizations in Resource and Scalability
Bandwidth Optimization:
Rather than uploading full files for every save, Cloud-ID's editor client computes diffs (changes) and only sends the smallest difference set, significantly cutting back on bandwidth use[4].

Lazy Loading:
Non-critical resources like project history or settings are only loaded when they are required to ensure a rapid initial load[4][10].

Auto-Scaling (Future Improvements):
The architecture is designed to accommodate Kubernetes integration for auto-scaling of containers and bridge servers as per the system load. This guarantees seamless functioning even during traffic bursts[7][8][12].

V. Acknowledgment

The writer would like to take this opportunity to express warmest thanks to Professor Mrs Ratna Chakrabarty for her precious advice, guidance, and constant encouragement throughout the duration of this project. Her inspiration and confidence in my work helped pave the way for this study,

giving me a chance to prove myself and test my abilities in a purposeful and scholarly environment.

We also express our sincere gratitude to the open-source community for their foundational contributions to cloud computing, containerization, and browser-based application development. Projects such as Docker[2], Kubernetes[12], Node.js, and open-source code editors have paved the way for platforms like Cloud-ID to materialize. Special thanks to platforms like Repl.it[3], StackBlitz[4], and CodeSandbox for pioneering the concept of accessible cloud IDEs, serving as critical inspirations for this project.

It has really been a great and life-changing experience, and I appreciate the chance to be able to give this project life under her guidance.

VI. REFERENCES

The following articles are the references used in order to complete this research paper

[1] Blewitt, William, Gary Ushaw, and Graham Morgan. "Applicability Docker Documentation – https://docs.docker.com/

[2] Replit Official Documentation – https://docs.replit.com/

[3] StackBlitz: Online IDE for Web Applications – https://stackblitz.com/

[4] CodeSandbox Documentation – https://codesandbox.io/docs

[5] WebSockets: Real-Time Communication – https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[6] xterm.js: Terminal for the Web – https://xtermjs.org/

[7] Amazon Web Services (AWS) - Cloud Computing Basics – https://aws.amazon.com/what-is-cloud-computing/

[8] Google Cloud Platform (GCP) - Cloud Computing Concepts – https://cloud.google.com/docs

[9] "A Survey of Cloud-based Integrated Development Environments" - Academic Paper Source

[10] GitHub - Cloud-ID Project Repository –

[11] Eclipse Theia Documentation,-https://theia-ide.org/

[12] Eclipse Che Documentation, https://www.eclipse.org/che/