SyncLab: A Scalable, Containerized Collaborative Coding Environment with Embedded OS and Real Time Multimedia Interaction

Syed Hisham Akmal, Saad Hussain, Sumanth P S, Chethan K

Shruthi B M, Assistant Professor, Dept of CSE, JSS STU, Mysore

Syed Hisham Akmal, Dept of CSE, JSS STU, Mysore
Saad Hussain, Dept of CSE, JSS STU, Mysore
Sumanth P S, Dept of CSE, JSS STU, Mysore
Chethan K, Dept of CSE, JSS STU, Mysore

---***---

Abstract—With increasing needs for remote software development environments, conventional coding platforms find it increasingly difficult to provide smooth collaboration, real time interactions, and safe execution of code in distributed teams. This paper introduces SyncLab, a contemporary, browser based collaborative development environment meant to close these gaps by combining containerized virtual operating systems, embedded code editors, secure video conferencing and real time white boarding into one unified interface. Creating isolated, scalable development sessions without local installation, SyncLab is developed with the use of Docker, React, VNC, WebRTC and Agora SDK. The system focuses on accessibility, this is achieved through noVNC based desktop virtualization and OAuth2 secured user sessions, creating a seamless experience for developers, educators and interviewers. The concepts behind SyncLab's architecture are built from existing Desktop-as-a-Service (DaaS) models and addresses the laminations of existing platforms such as CodeSandbox, GitHub Codespaces, and Replit. Experimental implementation and literature evidence emphasizes SyncLab's efficiency in resource optimization, user experience and cross platform, making it an excellent solution to the pending issues in modern software engineering.

Key Words**:** Containerization, Collaborative Coding, Docker, VNC, WebRTC, Desktop-as-a-Service (DaaS), a Virtual Operating System, noVNC, Agora SDK, Software Development Tools, Cloud based Development.

1.INTRODUCTION

There has been a radical reshaping of the landscape of software development during the world-wide move towards remote and hybrid environments. This transformation has increased the calls for the existence of platforms that promote collaborative programming, real time communication, and simplified infrastructure management. Developers, teachers, and technical crews are increasingly needing to have environments not only functional but intuitive, secure, and scalable as well. Popular for obvious reasons, traditional Integrated Development Environments (IDEs) and online editors just do not provide the seamless experience of code execution, team collaboration, and user interaction over the web.

SyncLab is a proposed solution for these increasing demands to provide a browser based fully integrated platform for development. The environment also provides a containerized embedded Linux environment with web browser accessibility utilizing Virtual Network Computing (VNC) and noVNC technologies. The system provides a Visual Studio Code-like editor, live collaborative whiteboard, secure file-sharing interface, and real-time video conferencing without need for complex local configurations. SyncLab presents an extension of the DaaS paradigm using the containerization technologies such as Docker to provide isolated, reproducible, and environments on demand [1].

The addition of remote desktop access using web protocols; in the form of VNC and noVNC that are used in tools such as GUIdock-VNC, also allows for SyncLab to deliver GUI-rich Linux environments directly in the browser without dependency on the client-side installation [2]. This is consistent with changes within the containerization community, where reproducibility, portability, and graphical interface delivery over the cloud have become essential requirements [3].

In addition, SyncLab explains several limitations observed in other platforms. Tools such as Replit, GitHub Codespaces, and CodeSandbox provide partial support, and this is often characterized by restricted collaboration alternatives, steeper learning curves or dependence on third party's repository. What makes SyncLab unique is: real time collaborative functions, built in operating system, ability to support multiple use cases, and ability to be beginner friendly and then grow in capabilities of the experts.

The rest of this paper is organized as follows: The related works and technological foundations that were used to formulate the SyncLab architecture are discussed in Section II. Section III outlines the system design such as container orchestration, virtual access, and session management. The implementation details are presented in section IV while in section V, we see the results and performance evaluation. Finally, Section VI closes the paper and proposes directions for further work.

2. LITEARTURE SURVEY

In building a dynamic, browser-based collaborative coding environment, such as SyncLab, several base technologies and ideas are needed. The literature describes several ways to support remote accessibility, reproducibility, and support for GUI in containerized environments. This section recapitulates some of the central findings of previous works that guided the architecture and implementation of SyncLab.

A. Desktop-as-a-Service and Seamless Application Delivery

Baun, Bouché provides an extensive analysis of different DaaS models in the age of current DaaS models and suggest a novel architecture of a DaaS capable of running unmodified Linux and Windows desktop applications fully inside a web browser. The authors emphasize the importance of keeping down dependencies on client-side software and promote browser delivery of desktop environments via remote III. METHODOLOGY rendering protocol like VNC and RDPs. The proposed model supports cross platform app deployment, session persistence and centralized resource control – all things which share alignment with SyncLab s objectives.

The paper then goes on to outline an increased demand for scalable and secure multiplatform access amid the BYOD (bring your own device) trends specifically in educational and enterprise environments. These findings justify containerized workspace and browser level session management in SyncLab to maintain access and operation consistency across heterogeneous environments.

B. Graphical Desktop Access Using Browser-Based VNC

Mittal et al. [2] introduce GUIdock-VNC, a solution Mittal et al. [2] present GUIdock-VNC, a tool meant to enclose containerized bioinformatics workflows in a browser accessible GUI using Virtual Network Computing (VNC) and noVNC. By using HTML5 and WebSockets, GUIdock VNC allows complete graphical interaction inside normal web browsers unlike X11-based methods. It lessens user-side configuration load and does away with the need for specialized client software.

The authors show that launching complicated, GUI intensive programs like Cytoscape inside Docker containers, accessed effortlessly via a browser interface, is possible. Well suited for use in cloud environments with strong access control systems, GUIdock-VNC also supports OAuth2 authentication. SyncLab's design is built on these ideas, particularly in its embedded virtual operating system, secure login system, and noVNC-driven graphical interface.

C. Containerization for Simulation and Visual Performance

Liang [3] looks at Project Chrono, a physics-based multi physics simulation library, and how it works with Docker containers. By providing pre-configured environments through Docker, the study seeks to simplify difficult software builds and improve reproducibility. Particularly important is the inclusion of noVNC and XFCE for browser-based real time simulation visualisation. Running GUI apps in containers, according to Liang, causes little performance loss and greatly simplifies the user experience by means of abstraction of dependency control.

The project also demonstrates how GPU-accelerated applications—e.g., NVIDIA CUDA-based simulations—can be run from inside containers, so offering insight on handling compute-intensive activities remotely. SyncLab keeps responsiveness by means of effective resource orchestration and browser-based rendering even as it uses Docker to provide embedded coding environments and user-defined scripts, hence adopting a comparable approach.

These studies taken together emphasize the need of integrating containerization, remote GUI access, and safe session management to provide a smooth cloud-native development experience. SyncLab combines these concepts into a single collaborative platform comprising video conferencing, real-time code editing, whiteboarding, and file sharing, therefore providing an all-in-one solution for distributed development and education.

3. METHODOLOGY

The foundation of SyncLab's development is a containerized, modular architecture intended to offer a smooth, browser-based collaborative coding environment. The technological stack, architectural choices, integration procedures, and system workflows that underpin SyncLab's core functionality are described in this section. Best practices in real-time communication, remote GUI access, and containerization, as noted in earlier studies [1]– [3], inform the methodology.

A. Containerized Virtual Operating Environment

An embedded Linux operating system running in a Docker container serves as the central component of SyncLab, providing a safe and separate workspace for every user session. Docker is selected due to its portability, reproducibility, and lightweight virtualization, which enable quick environment instantiation across networks and devices. Because each workspace comes pre-configured with the necessary development tools, editors, and terminal access, users are guaranteed not to have to carry out any local installations or system configuration.

A new Docker container is dynamically spawned by the /api/new-instance API endpoint, which starts the container creation process. The container is registered in the PostgreSQL-backed session database, and a Redis-based port management system is used to assign a unique VNC port.

B. GUI Access via VNC and noVNC

SyncLab uses noVNC, an HTML5-based VNC client, on the frontend and a VNC server (TigerVNC) operating inside the container to offer browser-based access to the containerized operating system. This feature enables users to use a web browser to interact with a complete GUI environment, including editors, file explorers, and terminals.

This design makes the platform accessible from any device that supports contemporary web standards and does away with the need for specialized desktop software. By providing cross platform compatibility, real-time rendering, and lower bandwidth consumption through differential screen updates, noVNC is in line with the tactics suggested in [2] and [3].

C. Real-Time Code Collaboration and Whiteboarding

SyncLab offers a real-time collaborative whiteboard built with WebSocket-based bi-directional communication to improve teamwork beyond code. This feature, which is essential for design discussions, interview situations, and educational use cases, enables users to collaboratively sketch, annotate, and visualize concept.

D. Secure Video Conferencing Integration

There is no need to switch between apps because video conferencing is an integrated feature in every room. Joining audio/video channels requires secure token-based. authentication.

E. File Management and Custom Scripts

To facilitate resource sharing, users can upload and download code files and folders from within the workspace using RESTful API endpoints (/api/upload/:roomId, /api/download/: roomId). Uploaded files are mounted into the active Docker container, ensuring they are immediately accessible in the GUI environment.

Additionally, SyncLab supports custom scripts during room creation, enabling predefined tasks, data setup, or environment-specific configurations. These scripts are executed automatically upon container initialization, offering flexibility for educators, project leads, and automation enthusiasts.

F. Authentication and Session Persistence

Clerk, a third-party authentication provider, manages user authentication and is integrated with PostgreSQL to preserve workspace snapshots, user profiles, and session logs. Role based access control (RBAC) and OAuth2 integration are enabled by the system, guaranteeing safe and efficient login procedures.

Redis is used to map rooms to the corresponding Docker instances to preserve session state. A consistent user experience requires that users be able to reconnect to their prior sessions without losing any data in the event of a disconnection.

G. System Workflow Overview

The entire system follows this sequence:

1. Authentication: Users log in via Clerk OAuth2; backend verifies credentials and creates a user session.
2. Room Creation: A new Docker container is instantiated with a unique VNC port and initialized with optional scripts.
3. Room Access: noVNC connects the user’s browser to the container's GUI over WebSockets.
4. Collaboration: Users engage in real-time coding, whiteboarding, and video conferencing.
5. Persistence: Files and snapshots are saved to PostgreSQL for retrieval or analysis.

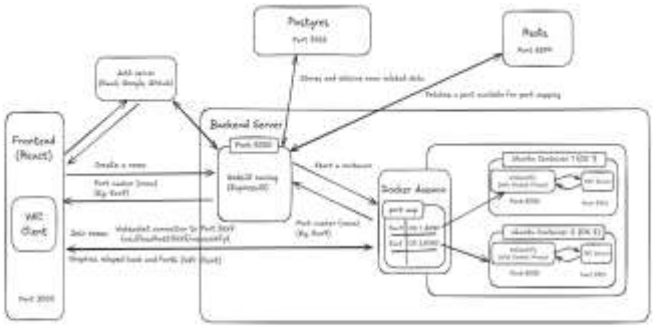

Fig. 1: Architecture Diagram

4. RESULT AND DISCUSSION

Several feature-level validations were carried out to assess SyncLab's practical performance and teamwork effectiveness. Concurrent accessibility, modular container behavior, dynamic environment provisioning, and overall system stability under load were all highlighted in these tests. Workspace isolation, package customization, resource management, and collaboration tools are among the essential elements that have been verified.

A. Functional Validation

1) Workspaces

Users can work in several separate development environments in the same collaborative space thanks to SyncLab's workspace feature. This feature guarantees that every user, or group of users, can operate freely and independently. A distinct file system structure, a specific VNC port mapping, and an isolated collection of active processes are all maintained by the Docker container that supports each workspace. For instance, Workspace 2 is completely independent under */home/user2/CodeFiles*, whereas Workspace 1 functions with its own directory structure at */home/user1/CodeFiles*. To prevent graphical session routing overlap, the containers linked to these workspaces are mapped to distinct ports, such as 6080 and 6081.

An array of websockify port mappings for every active room is stored in the backend system, which is built with Node.js and Express and uses Prisma as the ORM. The task of dynamically updating the active workspace and rerouting user sessions to the proper container falls to an endpoint called /api/switch-workspace. A dropdown interface on the frontend makes it simple for users to switch between the available workspaces. The active workspace ID and port are maintained by React's state management, which guarantees that the VNC viewer reconnects properly without session loss or graphical issues. By supporting several separate coding environments in one room, this architecture prevents resource conflicts like file collisions and port contention.

2) Custom Package Installation

SyncLab offers an integrated package installation mechanism that allows for flexible and secure environment customization. With the help of this feature, users can customize their workspace to meet development needs by installing new software packages on demand within their isolated containers. Users engage with a responsive package selector on the frontend, which was constructed with PACKAGE_LIBRARY.js. Packages are arranged according to their functions, such as text editors, networking tools, and development tools. To improve usability, the interface offers real-time search, filtering, and multi-selection.

To prevent malicious or superfluous installations, the backend first verifies the chosen packages against a predetermined security policy before processing these requests. After validation, the server runs a docker exec command, which installs the package inside the appropriate container. The installation procedure is atomic to preserve system integrity, and intermediate states are closely monitored to prevent discrepancies. Additionally, SyncLab has sophisticated features that let users save and restore their container's custom states, like environment snapshotting via docker commit. A transparent and effective installation process is ensured by the integration of dependency resolution and progress monitoring features.

3) Port and Resource Management

For SyncLab to scale across several concurrent sessions without experiencing system failure or contention, effective port and resource management is essential. A distinct set of ports is needed for WebSocket communication and VNC connections in every room and workspace in SyncLab. SyncLab uses Redis as a central repository for port availability tracking to make this possible. To guarantee that no two sessions are given the same port, the system makes an atomic reservation of ports from the Redis pool whenever a new room or container is created.

SyncLab starts a systematic cleanup procedure when users exit a room. The associated container receives a SIGTERM signal from the backend server, which initiates a graceful shutdown. Docker rm -f is used to forcefully remove the container if it does not terminate within the specified timeout period. The backend removes the room metadata from the Prisma-managed PostgreSQL database and returns the ports to the Redis pool after container termination. An API endpoint at /api/leave-room/: roomId is used to manage this lifecycle, and a background worker keeps an eye out for any orphaned containers or unreleased ports to ensure system availability and hygiene.

4) Whiteboard and In-Room Chat

SyncLab incorporates a live whiteboard and real-time chat system into every session to further improve collaborative development. The whiteboard, created with the Fabric.js library, offers an interactive canvas with basic diagramming tools, object manipulation, and freehand drawing capabilities. Updates made by one participant are immediately reflected for other participants in the room because WebSocket connections synchronize the canvas state across all users.

The in-room chat feature, which allows participants to text each other in real time, is a great addition to the whiteboard. To reduce latency and provide a lightweight messaging layer that functions independently of other components, the chat is implemented using raw WebSocket protocols. Users can participate in design discussions, offer feedback, or exchange ideas without ever leaving the development environment thanks to this dual system of textual and graphical communication, which increases engagement and productivity.

The system was tested by multiple users across different operating systems and browsers (Windows, macOS, Linux, Chrome, and Firefox). Each user was able to:

- Authenticate using OAuth2 (Google, GitHub, Email).
- Successfully create a room and instantiate an isolated Ubuntu/ Kali/ Debian container.
- Access a full Linux GUI environment in-browser using noVNC with minimal latency.
- Execute development tasks such as code editing, compilation, and system navigation.

B. Comparison with Related Work

1) Compared to GUIdock-VNC [2]

GUIdock-VNC provided a foundational structure for GUI container deployment over web. However, SyncLab extends this concept by introducing:

- Real-time port allocation via Redis
- Persistent session storage via PostgreSQL
- OAuth2-based multi-provider authentication

While GUIdock-VNC emphasized single-session containers, SyncLab supports multi-session environments with authentication and isolation built-in.

2) Compared to Project Chrono + Docker Integration [1]

Though [1] focused on Docker use for simulation orchestration, SyncLab applies Docker in an interactive, user centric scenario. SyncLab's port-mapped architecture and browser-based accessibility present a more direct, GUI focused use-case compared to the simulation-heavy approach in [1].

3) Compared to Web-Based Desktop Virtualization [3]

The architecture in [3] highlights containerized desktop applications. SyncLab advances this by integrating web native GUI access (via noVNC), centralized session orchestration, and lightweight backend logic with containerized resource provisioning.

5. CONCLUSION

This paper presented SyncLab, a lightweight, container based virtual development environment that provides browser-accessible Linux GUIs using Docker, noVNC, and a Node.js-based orchestration layer. The system successfully addresses challenges in remote software development, such as environment setup complexity, resource isolation, and accessibility, by leveraging modern containerization and real time communication technologies.

Key findings from the implementation and evaluation include:

- Browser-Based GUI Accessibility: Through noVNC integration, users experienced a responsive Linux GUI environment directly within their web browser, with minimal input lag.
- Scalability and Reliability: The system supported concurrent multi-user sessions with minimal resource contention, thanks to Redis-based dynamic port allocation virtualization. and Docker's lightweight
- Secure and Extensible Architecture: Integration with OAuth2 providers ensures secure user authentication and facilitates role-based access, laying the groundwork for educational, enterprise, or team-based applications.

Compared to existing literature and tools, SyncLab extends functionality by combining session persistence, scalable container deployment, and real-time web-based GUI interaction into a unified platform. These results have practical implications for educational institutions, hackathon platforms, remote teams, and cloud-based IDE services.

Future work will focus on integrating GPU support, adopting WebRTC for lower-latency communication, and developing automated resource cleanup and session analytics dashboards to enhance usability and maintainability.

REFERENCES

[1] C. Baun and J. Bouché, "Closing the Gap between Web Applications and Desktop Applications by Designing a Novel Desktop-as-a-Service (DaaS)," *Open Journal of Cloud Computing*, vol. 8, no. 1, 2023.

[2] V. Mittal et al., "GUIdock-VNC: Using a Graphical Desktop Sharing System to Provide a Browser-Based Interface for Containerized Software," *GigaScience*, vol. 6, 2017.

[3] T. Liang, "Integration of Docker Containers and Project Chrono," Simulation-Based Engineering Lab, University of Wisconsin-Madison, Technical Report 2023-04.

[4] R. Afreen, "Bring your own device (BYOD) in higher education: Opportunities and challenges," International Journal of Emerging Trends & Technology in Computer Science, vol. 3, no. 1, pp. 233–236, 2014.

[5] M. Afrin, J. Jin, A. Rahman, A. Rahman, J. Wan, and E. Hossain, "Resource Allocation and Service Provisioning in Multi-Agent Cloud Robotics: A Comprehensive Survey," IEEE Communications Surveys & Tutorials, vol. 23, no. 2, pp. 842–870, 2021.

[6] S. A. Algarni, M. R. Ikbal, R. Alroobaea, A. S. Ghiduk, and F. Nadeem, "Performance evaluation of xen, kvm, and proxmox hypervisors," International Journal of Open-Source Software and Processes (IJOSSP), vol. 9, no. 2, pp. 39–54, 2018.

[7] M. Amaral, "Kubevirt scale test by creating 400 vmis on a single node," in Free and Open-source Software Developers' European Meeting, 2022.

[8] F. Bellard, "QEMU, a fast and portable dynamic translator," in USENIX annual technical conference, FREENIX Track, vol. 41. California, USA, 2005, p. 46.

[9] A. Binu and G. S. Kumar, "Virtualization techniques: a methodical review of XEN and KVM," in Advances in Computing and Communications: First International Conference, ACC 2011, Kochi, India, July 22-24, 2011. Proceedings, Part I 1. Springer, 2011, pp. 399–410.

[10] M. Bolte, M. Sievers, G. Birkenheuer, O. Nieh orster, and A. Brinkmann, "Non-intrusive ¨ virtualization management using libvirt," in 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010). IEEE, 2010, pp. 574–579.