SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

Design and Implementation of a Lightweight Distributed Container Orchestration Engine with Custom Scheduling and Fault-Tolerant Consensus

PROJECT REPORT

Submitted by

SUCHARITHA B - E0122047

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Artificial Intelligence and Machine Learning)

Sri Ramachandra Faculty of Engineering and Technology,

Sri Ramachandra Institute of Higher Education and Research,

Porur,

Chennai -600116

March, 2026

CONTENTS

Abstract

This project involves designing and implementing a mini Kubernetes-style container orchestrator from scratch. The system will automatically manage, schedule, scale, and heal Docker containers across multiple simulated nodes running on a single laptop.

Key features include:

- Raft-based consensus for cluster coordination and leader election
- A custom predictive scheduler with resource forecasting and anti-affinity rules
- Automatic fault detection and self-healing
- Real-time cluster monitoring via a web dashboard

The project directly addresses real challenges faced by WebSyncz clients in logistics, manufacturing, and real estate, where managing dozens of microservices is complex and error-prone.

Current Status (as of 16 March 2026): Project has just started. A working 3-node simulation with basic scheduling will be completed by the end of Month 1.

Project Title Selection

The project title “Design and Implementation of a Lightweight Distributed Container Orchestration Engine with Custom Scheduling and Fault-Tolerant Consensus” was finalised after careful analysis during the first week of the internship at WebSyncz Technology.

A detailed review of ongoing client projects and internal workflows revealed that the majority of clients, particularly in the logistics, manufacturing, and real estate sectors, have transitioned to microservices-based architectures. These clients typically manage between 15 and 50 independent services, each deployed as Docker containers.

It was observed that the manual handling of container lifecycles , including provisioning, scaling during peak demand, restarting failed instances, and load balancing , remains highly labour-intensive and operationally risky. This creates significant challenges in reliability, scalability, and maintenance.

The development of an in-house, lightweight container orchestration engine was identified as a strategic solution. Such a system would automate these critical operations and could be leveraged both as an internal DevOps tool and as a value-added offering to clients. The chosen title accurately reflects the core technical focus and contributions of the project:

- Lightweight Distributed Container Orchestration Engine - emphasises the creation of a minimal yet functional orchestrator.
- Custom Scheduling - highlights the development of an intelligent, predictive scheduler tailored to enterprise requirements.
- Fault-Tolerant Consensus - underscores the implementation of distributed coordination mechanisms (such as Raft) to ensure reliability and consistency across nodes.

This title was preferred over broader or simpler alternatives because it clearly communicates the depth of systems engineering involved while directly aligning with WebSyncz Technology’s expertise in cloud infrastructure, DevOps, and enterprise software solutions.

Why Modern Companies Need Microservices & Containers

Modern enterprise applications are no longer built as one large monolithic system. Instead, they are broken into many small, independent services (microservices).

Example – Logistics Company (Blue Dart / DTDC style):

- Customer-facing website & mobile app
- Real-time GPS tracking system
- Warehouse inventory management
- Payment & billing system
- Driver mobile application
- Analytics dashboard
- Notification engine (SMS/Email)

Benefits of this approach:

- If the payment service crashes, tracking still works
- Different teams can develop and deploy independently
- Only the required service can be scaled during peak load
- Easier maintenance and faster feature releases

This architecture is now standard in the domains WebSyncz serves (logistics, manufacturing, real estate, education). However, managing hundreds of Docker containers manually is the biggest pain point ,which this project solves.

What is a Docker Container?

A Docker container is like a standardised shipping box.

Inside one box you pack:

- Application code
- All required libraries and dependencies
- Configuration files

Once packed, this box runs exactly the same way on any machine (laptop, server, or cloud) without “it works on my machine” issues. Containers start in 1–2 seconds and use very little resources.

My orchestrator will act as an intelligent traffic controller that automatically manages hundreds of such containers ,deciding where they run, scaling them, and healing them when something fails.

This is not a simple “run Docker commands” project. It is one of the most challenging systems engineering projects an intern can attempt because:

- Implementing Raft consensus for leader election and consistent cluster state across nodes
- Building a custom scheduler that solves an NP-hard optimisation problem (CPU, memory, network, future load prediction, anti-affinity)
- Achieving true fault tolerance ,automatic detection and recovery from node/container crashes
- Handling concurrency, network delays, and race conditions while maintaining strong consistency
- Doing everything on a single laptop by simulating multiple real nodes

Very few interns attempt something at this depth. Companies like Google, Amazon, and Meta highly value such projects because they mirror the core problems their production systems solve every day.

Connection to AI/ML Field

Although this is a pure Systems Engineering project, it has strong ties to my AI/ML background:

- The custom scheduler can later incorporate simple ML models (regression or reinforcement learning) to predict future resource usage and make smarter placement decisions.
- AI-based anomaly detection can be added to predict node failures before they occur.
- Large-scale AI workloads (model training, inference services, RAG pipelines) heavily depend on efficient container orchestration.

This project therefore bridges Systems Engineering + AI/ML, making me stronger in both domains ,exactly what top companies look for.

6-Month Roadmap (March 16 – September 15, 2026)

Month 1 (Mar 16 – Apr 15): Foundation

- Docker node simulation
- Basic API Server + Worker Agent
- Simple task submission & execution

Month 2 (Apr 16 – May 15): Distributed State & Consensus

- Implement Raft consensus
- Cluster state management

Month 3 (May 16 – Jun 15): Custom Scheduler & Fault Tolerance

- Predictive scheduler with scoring algorithm
- Auto-healing + rolling updates

Month 4 (Jun 16 – Jul 15): Dashboard & Observability

- React + TypeScript web dashboard
- Live metrics and logging

Month 5 (Jul 16 – Aug 15): Advanced Features & Testing

- Service discovery + load balancing
- Chaos testing (simulate node failures)
- Performance benchmarks

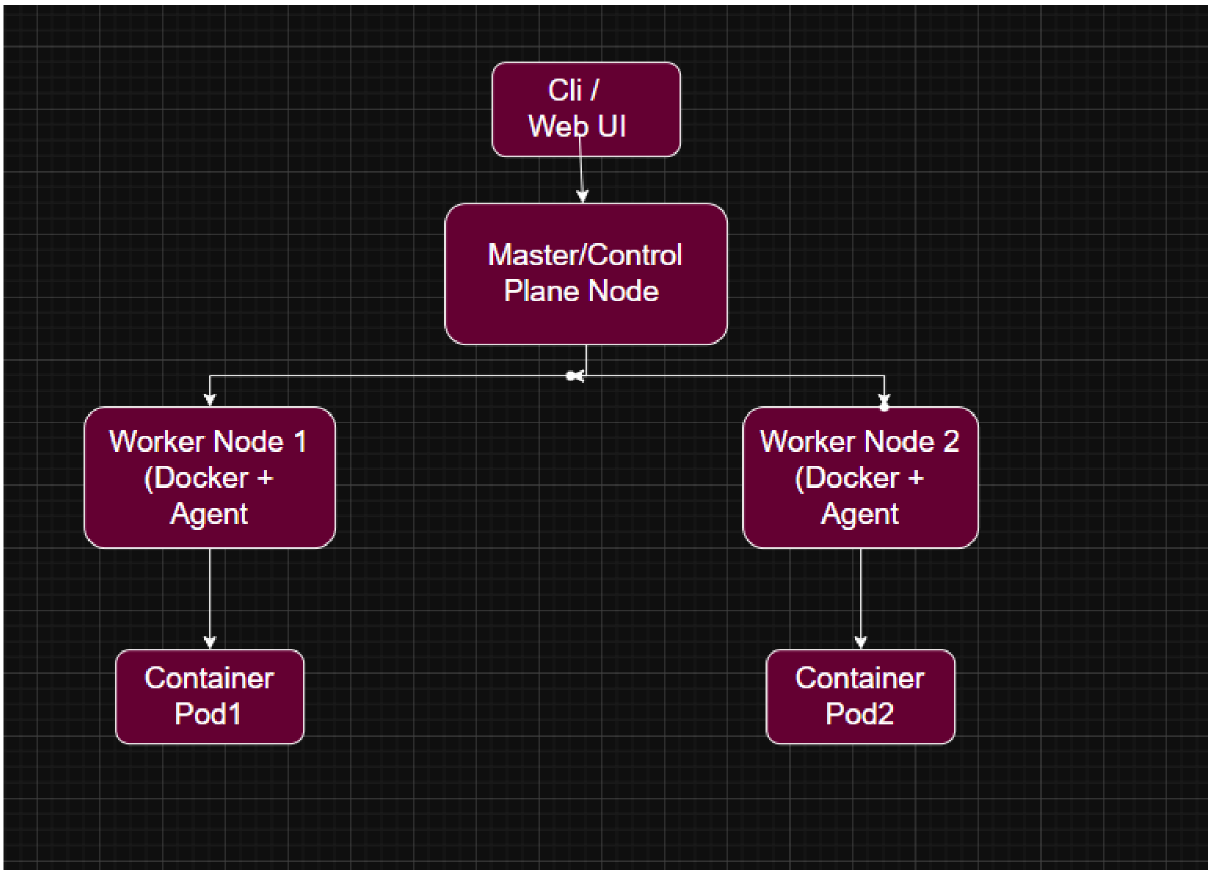
Month 6 (Aug 16 – Sep 15): Polish & Delivery

- Documentation, demo video, final report
- Presentation + handover to WebSyncz team

Key Milestones

- End of April: Working 3-node cluster
- End of June: Custom scheduler + auto-healing live
- End of August: Full dashboard + demo ready

High-Level Architecture Diagram



Core Components:

- Control Plane (Scheduler + Raft consensus)
- Worker Nodes (Docker runtime + agent)
- Custom Scheduler (your novel algorithm)
- Fault Tolerance & Self-Healing

Next Steps I Propose

1. Start implementation from next week with basic Docker simulation.
2. Share weekly progress updates every Monday.
3. Deliver a working 3-node cluster by the end of April.

Expected Outcomes & Benefits to WebSyncz Technology

Upon successful completion of the project by 15 September 2026, the following outcomes are anticipated:

- A fully functional, lightweight container orchestration engine capable of managing 5–10 simulated nodes with automatic scheduling, scaling, and self-healing capabilities.
- A custom predictive scheduler that optimises resource allocation based on real-time and forecasted load.
- A React-based web dashboard for real-time cluster monitoring, task submission, and visualisation.
- Comprehensive documentation, performance benchmarks, and a live

demonstration video.

Benefits to WebSyncz Technology:

- **Internal Efficiency**: The engine can be adopted as an internal DevOps tool to automate the management of microservices across ongoing client projects, reducing manual effort and operational risks.
- **Client Value Addition**: The system can be packaged and offered as a premium service to clients in logistics, manufacturing, and real estate sectors, enabling them to run private or hybrid cloud environments at lower cost.
- **Competitive Advantage**: WebSyncz will gain in-house expertise in distributed systems and container orchestration , a capability currently outsourced by many similar companies.
- **Future-Ready Foundation**: The codebase can be extended with AI-driven scheduling or integrated with existing client applications, creating opportunities for new revenue streams in cloud and DevOps consulting.

Risk Analysis & Mitigation Strategies

The following risks have been identified along with corresponding mitigation measures:

Risk	**Likelihood**	**Mitigation Strategy**

Technical complexity in implementing Raft consensus leading to inconsistency	Medium	Start with simplified consensus algorithms in Month 1; incrementally add full Raft features with extensive unit testing and simulation of network partitions.
Performance bottlenecks when scaling to multiple nodes on a single laptop	Medium	Use lightweight Docker simulation and resource monitoring from the beginning; conduct regular benchmarking and optimise code iteratively.
Delay in achieving functional milestones due to debugging distributed systems issues	High	Follow a strict weekly sprint model with clear deliverables; maintain daily progress logs and seek guidance from the internship mentor every Monday.
Scope creep (adding too many advanced features)	Low	Strictly limit features to the defined 6-month roadmap; any additional features will be marked as "future enhancements" in the final report.
Integration challenges with existing WebSyncz client infrastructure	Low	Design the engine with modular APIs and clear documentation to ensure easy integration; conduct compatibility testing in Month 5.

References

1. Kubernetes Documentation. "Kubernetes Architecture Overview." Official Kubernetes Website, 2025.
2. Burns, Brendan et al. *Designing Distributed Systems*. O'Reilly Media,

2018.

3. Ongaro, Diego and Ousterhout, John. “In Search of an Understandable Consensus Algorithm (Raft).” USENIX ATC, 2014.
4. Hindman, Benjamin et al. “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.” NSDI, 2011.
5. Docker Documentation. “Docker Engine Architecture.” Docker Official Docs, 2025.
6. Brewer, Eric. “CAP Theorem Revisited.” IEEE Computer, 2012.
7. Verma, Abhishek et al. “Borg: Google’s Cluster Manager.” USENIX ATC, 2015.