**Vortexify: A Builder**

**“From Code to Container – Vortexify builds it all.”**

------------------------------------------------------------------------------------------------------------------------

🧾 SOFTWARE REQUIREMENTS SPECIFICATION

Project Name: Vortexify: A Builder

1. Introduction

1.1 Purpose

The purpose of Vortexify is to provide an automated platform for deploying Dockerized applications directly from GitHub repositories to a remote Docker-enabled VMware virtual machine. The platform divides responsibilities across three modules — Heart, Brain, and Skin — to maintain separation of concerns, modularity, and scalability.

1.2 Scope

Accept GitHub repository links from users.

Deploy applications present in Docker-compatible GitHub repos.

Provide users with a live app link after deployment.

Offer a dashboard to manage user credentials and deployment history.

Ensure modularized execution via Spring Boot (Brain), Python (Heart), and Laravel (Skin).

React and C++ can be optionally embedded for enhancements (Future Work).

1.3 Intended Audience

DevOps Enthusiasts

Full-stack Developers

Students and Educators

Deployment/Test Automation Engineers

1.4 Definitions, Acronyms, and Abbreviations

VMware: Virtualization platform hosting the Dockerized environment.

Dockerfile: A script file containing instructions to build a Docker image.

SRS: Software Requirements Specification

REST API: Web service-based communication

UI: User Interface

1. Overall Description

2.1 Product Perspective

Vortexify is a standalone platform divided into three modules:

Heart Module (Python/C++) – for automation and remote execution.

Brain Module (Spring Boot) – for backend logic and deployment tracking.

Skin Module (Laravel) – for UI interaction and data handling.

* 1. Product Functions

Accept GitHub URL and trigger deployment

Build Docker images and run containers via SSH

Return live deployment link

Track deployment history

Enable user registration/login/password reset

Log container performance

* 1. User Classes

Guest Users – Can register and submit GitHub links

Registered Users – Can deploy, view history

Admin (Future scope) – Can monitor all users and performance logs

2.4 Operating Environment

Client: Web Browser (Chrome, Firefox, etc.)

Server Requirements:

Java 17+

Python 3.x

Laravel 10+

MySQL

Docker

VMware Workstation with Ubuntu VM (Docker installed)

1. Specific Requirements

3.1 Functional Requirements

Skin Module (Laravel)

FR1: User Registration/Login

FR2: GitHub Repo Submission Form

FR3: Display Deployment History & Live Links

Brain Module (Spring Boot)

FR4: Accept API Requests from Laravel

FR5: Validate GitHub links

FR6: Trigger Python script and monitor execution

FR7: Save deployment logs with user ID and link

FR8: Token-based password reset flow

Heart Module (Python + C++)

FR9: SSH into remote VM

FR10: Clone GitHub repo

FR11: Build Docker image and run it

FR12: Extract container IP/port

FR13: Return link to Spring Boot

FR14: Log container performance (CPU/RAM) via optional C++ script

* 1. Non-Functional Requirements

NFR1: Response time should be under 3 seconds for form submission

NFR2: Docker containers must be deployed within 60 seconds (on average)

NFR3: Logs must be stored persistently

NFR4: Secure communication between services (API key/token)

NFR5: Modular and scalable codebase

4. External Interface Requirements

4.1 User Interfaces

Laravel blade views or React UI components

Login/Register/Deploy page

Deployment history dashboard

4.2 Hardware Interfaces

VMware virtual machine with Docker installed

4.3 Software Interfaces

Laravel ↔ Spring Boot via REST API

Spring Boot ↔ Python via subprocess or API

Python ↔ VM via SSH (paramiko)

4.4 Communication Interfaces

HTTP/HTTPS API

SSH (port 22)

5. Future Scope

Multi-container app support (docker-compose)

Auto SSL setup

Auto teardown after expiration

Notification system (email or in-app)

CI/CD Integration

6. Appendix

Project name inspired by the swirling energy of automation – Vortexify.

Modular naming convention:

Heart = Python Automation

Brain = Spring Boot Backend

Skin = Laravel UI

Optional React and C++ integrations for future expandability.

7. Limitations with One stack:

**Building Vortexify with only one stack wouldn't give you the same flexibility, control, and modular power that your multi-stack design offers.**

**❌ What Happens If You Use Only One Stack?**

| **Single Stack Used** | **Limitations** |
| --- | --- |
| **Only Laravel** | **Hard to manage deployment automation & VM-level scripting; PHP isn't suited for system-level tasks.** |
| **Only Spring Boot** | **Too heavy for frontend/UI management, difficult to handle SSH/remote scripting gracefully.** |
| **Only Python** | **Great for automation, but lacks strong native backend structure and UI support.** |
| **Only C++** | **Not web- or API-friendly; complex for anything beyond performance logging.** |
| **Only React** | **Frontend only — no backend or automation capabilities.** |

**✅ Why Multi-Stack *Is* the Right Choice for This Project**

| **Technology** | **Strength You’re Leveraging** |
| --- | --- |
| **Spring Boot** | **Central controller via API; handles token-based auth, deployment orchestration, and history tracking.** |
| **Laravel** | **Clean MVC UI, user management, and database handling.** |
| **Python** | **Automation wizard: handles Git clone, SSH, Docker execution — fast and scriptable.** |
| **C++** | **Super-efficient resource monitoring and file-based logging from inside the VM.** |
| **React (Future)** | **Dynamic frontend for real-time updates and modern UX (e.g., password reset workflows).** |

**we could force everything into one tech, but we’d lose clarity, performance, maintainability, and power.**

**Our multi-stack approach is more modular, realistic, and industry-aligned — especially for full-stack + DevOps projects.**