Autonomous DevOps Agent with React Dashboard

BACKGROUND CONTEXT

Modern software development relies heavily on CI/CD pipelines. However, these pipelines frequently fail due to various issues including code quality issues, logic bugs, syntax errors, type errors, and integration issues between multiple files.

Developers spend 40–60% of their time debugging these failures instead of building new features. This challenge tasks you to build an agent that can autonomously detect, fix, and verify code issues.

CORE CHALLENGE

Build an Autonomous DevOps Agent with a React Dashboard that:

- Takes a GitHub repository URL as input via the web interface
- Clones and analyzes the repository structure
- Discovers and runs all test files automatically
- Identifies all failures and generates targeted fixes
- Commits fixes with [AI-AGENT] prefix and pushes to a new branch
- Monitors CI/CD pipeline and iterates until all tests pass
- Displays comprehensive results in a React dashboard

REACT DASHBOARD REQUIREMENTS

⚠️ **The React dashboard is the primary interface judges will use to evaluate your agent. It MUST be production-ready and publicly deployed.**

1. Input Section

- Text input for GitHub repository URL
- Text input for Team Name (e.g., "RIFT ORGANISERS")
- Text input for Team Leader Name (e.g., "Saiyam Kumar")
- "Run Agent" or "Analyze Repository" button
- Loading indicator while agent is running

2. Run Summary Card

- Repository URL that was analyzed
- Team name and team leader name
- Branch name created (TEAM_NAME_LEADER_AI_Fix)
- Total failures detected and total fixes applied
- Final CI/CD status badge: PASSED (green) / FAILED (red)
- Total time taken (start to finish)

3. Score Breakdown Panel

- Base score: 100 points
- Speed bonus applied (+10 if < 5 minutes)
- Efficiency penalty (−2 per commit over 20)
- Final total score displayed prominently
- Visual chart/progress bar showing score breakdown

4. Fixes Applied Table

Table with columns:

- File | Bug Type | Line Number | Commit Message | Status
- Bug types: LINTING, SYNTAX, LOGIC, TYPE_ERROR, IMPORT, INDENTATION
- Status: ✓ Fixed or ✗ Failed
- Color coding: Green for success, red for failure

5. CI/CD Status Timeline

- Timeline visualization showing each CI/CD run
- Pass/fail badge for each iteration
- Number of iterations used out of retry limit (e.g., "3/5")
- Timestamp for each run

BRANCH NAMING REQUIREMENTS

> ⚠️ **CRITICAL: Your agent MUST create branches with this EXACT format: TEAM_NAME_LEADER_NAME_AI_Fix**

Naming Rules:

- All UPPERCASE
- Replace spaces with underscores (_)
- End with _AI_Fix (no brackets)

- No special characters except underscores

Team Name	Leader Name	Branch
RIFT ORGANISERS	Saiyam Kumar	RIFT_ORGANISERS_SAIYAM_KUMAR_AI_Fix
Code Warriors	John Doe	CODE_WARRIORS_JOHN_DOE_AI_Fix

TEST CASE MATCHING — EXACT FORMAT REQUIRED

⚠️ **CRITICAL: Your agent's output MUST match our test cases line-by-line.**

Judges will evaluate based on exact matches:

Test Case	Expected Dashboard Output
src/utils.py — Line 15: Unused import 'os'	LINTING error in src/utils.py line 15 → Fix: remove the import statement
src/validator.py — Line 8: Missing colon	SYNTAX error in src/validator.py line 8 → Fix: add the colon at the correct position

TECHNICAL REQUIREMENTS

Frontend

- Must be built with React (functional components + hooks)
- Must be responsive (desktop + mobile)
- Must be deployed and publicly accessible
- Frontend code must be in /frontend folder in repository
- Must use proper state management (Context API, Redux, Zustand, etc.)

Backend / Agent

- Must generate results.json file at end of each run
- Must include API endpoint that triggers agent (REST or GraphQL)
- Must use multi-agent architecture (LangGraph, CrewAI, AutoGen, etc.)
- Code execution must be sandboxed (Docker recommended)
- Must have configurable retry limit (default: 5)

MANDATORY SUBMISSION REQUIREMENTS

⚠ **ALL of the following are MANDATORY. Missing any component will result in DISQUALIFICATION.**

#	Requirement	Details
1	Live Deployed Website URL	React dashboard must be publicly accessible. Must accept GitHub repo URL as input. Platforms: Vercel, Netlify, Railway, AWS, or any hosting service.
2	LinkedIn Video Demonstration	2–3 min max. Must tag @RIFT2026 on LinkedIn. Must show: live demo, architecture diagram, agent workflow walkthrough, results dashboard. Post must be public.
3	GitHub Repository + README	Public repo. README must include: project title, deployment URL, LinkedIn video URL, architecture diagram, installation instructions, environment setup, usage examples, supported bug types, tech stack, known limitations, team members.

SUBMISSION FIELDS

- Problem Statement selected (on RIFT website — 19th Feb, 6–8 PM window)
- GitHub Repository URL
- Hosted / Live Application URL
- Demo video link posted on LinkedIn tagging RIFT's official page

EVALUATION CRITERIA

Criterion	Points	Description
Test Case Accuracy	40	Exact match with test cases: correct bug types, line numbers, fixes applied
Dashboard Quality	25	All required sections, clear visualization, responsive design, live deployment
Agent Architecture	20	Multi-agent structure, proper tool integration, sandboxed execution, iteration handling
Documentation	10	Complete README, architecture diagram clarity, setup instructions accuracy
Video Presentation	5	Clear explanation of architecture, live demo quality, professional presentation

DISQUALIFICATION CRITERIA

⚠️ **Immediate disqualification for any of the following:**

- No live deployment URL
- No LinkedIn video posted
- Incomplete README
- Output does not match test cases
- Human intervention during agent execution
- Hardcoded test file paths
- Commits without [AI-AGENT] prefix
- Incorrect branch name format
- Pushing directly to main branch
- Plagiarized code from other teams

Good luck! We're excited to see what you build.

— RIFT 2026 Organizing Team