Indian Institute of Technology, Kharagpur
Department of Computer Science and Engineering

Design Lab (CS59001) Documentation

Development of AI Agents for Automatic Analysis of Build Logs

10th April, 2025

Rushil Venkateswar
20CS30045

Contents

1. Abstract

This document provides a comprehensive overview of the CircleCI Snap-In project, which integrates CircleCI with the DevRev platform via webhooks. The project facilitates synchronization of pipeline, workflow, and job data to DevRev and further enhances developer productivity by generating AI-driven insights from build logs. This report details the system architecture, implementation strategies, testing methodology, and potential future enhancements.

2. Introduction

2.1 Background and Motivation

Modern software development environments rely on continuous integration (CI) and continuous deployment (CD) tools to maintain rapid development cycles. However, fragmented toolchains can reduce productivity. The CircleCI Snap-In project addresses these challenges by connecting CircleCI to the DevRev platform. By synchronizing data from CircleCI (projects, workflows, and jobs) and mapping it to DevRev (parts, issues, and tasks), the system offers a unified view of development activities and leverages AI to provide actionable insights.

2.2 Problem Statement

The integration of disparate systems like VSCode, Jenkins, CircleCI, and Replit with the DevRev platform poses significant challenges in terms of data parsing, real-time synchronization, and providing insightful feedback. The main goal is to:

- Seamlessly connect CircleCI data to DevRev using webhooks.
- Generate intelligent insights by analyzing build logs [Kum25].
- Improve developer workflows with real-time actionable recommendations.

3. System Architecture

3.1 Overview

The architecture consists of three main components:

1. **Data Synchronization:** Using CircleCI webhooks, events related to pipeline, workflow, and job status are captured and transmitted to the DevRev platform.
2. **Mapping and Data Flow:** Data from CircleCI is mapped as follows:
    - *CircleCI Project → DevRev Part*
    - *CircleCI Workflow → DevRev Issue*
    - *CircleCI Job → DevRev Task*
3. **AI Insights Generation:** Build logs are analyzed to detect root causes of failures, error messages, and to propose potential fixes. An AI agent then posts a comment on the associated DevRev issue.

3.2 Requirements

3.2.1 CircleCI Build System Setup

For a brief video tutorial refer here.

3.3 Data Flow Diagram

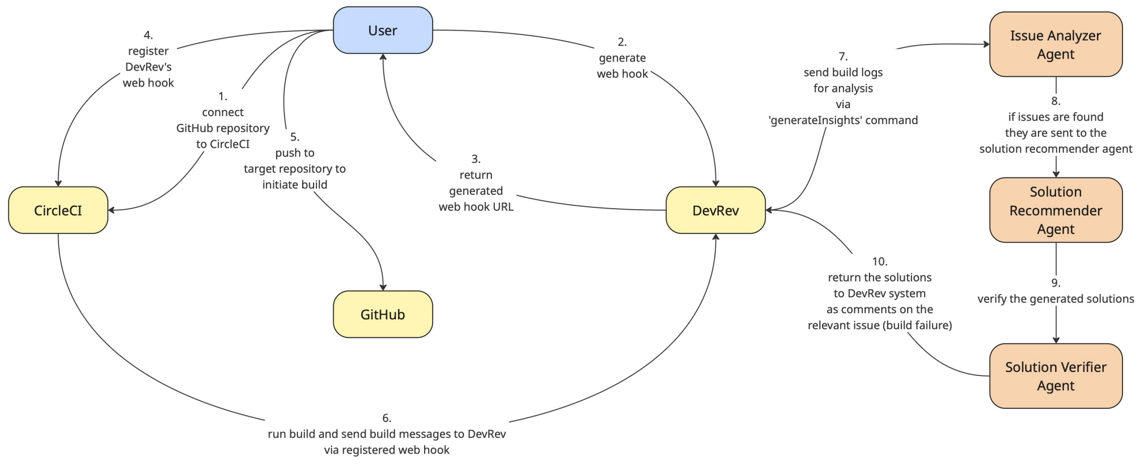
User
4. register DevRev's web hook
1. connect GitHub repository to CircleCI
5. push to target repository to initiate build
2. generate web hook
3. return generated web hook URL
CircleCI
GitHub
DevRev
6. run build and send build messages to DevRev via registered web hook
7. send build logs for analysis via 'generateInsights' command
Issue Analyzer Agent
8. if issues are found they are sent to the solution recommender agent
Solution Recommender Agent
9. verify the generated solutions
Solution Verifier Agent
10. return the solutions to DevRev system as comments on the relevant issue (build failure)

Figure 1: Data Flow Diagram of system showing broad components

4. Implementation Details

4.1 Integration with CircleCI

The integration relies on CircleCI's webhook mechanism. The snap-in listens for events such as workflow completions or job completions. The event payload is parsed and mapped to the corresponding DevRev entity. Then we invoke an endpoint on the DevRev system to create the said entity. This persists the information sent by the webhook.

```
import { ApiUtils } from "./utils";
import { WorkflowCompletedWebhook, JobCompletedWebhook, WebhookType } from "./types";

async function handleEvent(event) {
  console.debug("[server] Handling event with payload ID: ", event.payload.id);
  // Extract token and endpoint from event context
  const token = event.context.secrets["service_account_token"];
  const endpoint = event.execution_metadata.devrev_endpoint;
  const apiUtils = new ApiUtils(endpoint, token);
  // [Further event handling and mapping logic...]
}
```

Listing 1: Event Handling Example

```
// create a part for the project if not existing already
let part_id = "";
const parts_response = await apiUtils.getParts(dev_user_id);
if (parts_response.success) {
const parts = parts_response.data.parts;
const part = parts.find((p: Part) => p.name === project_name);
if (!part) {
  const part_response = await apiUtils.createPart({
    type: PartType.Product,
    name: project_name,
    owned_by: [dev_user_id],
  });
  part_id = part_response.data.part.id;
  if (part_response.success) {
    console.debug(
      `[server] Created part for project ${project_name} successfully`,
    );
  } else {
    console.error(
      `[server] Failed to create part for project ${project_name}. Err: ${part_response.message}`,
    );
  }
} else {
  part_id = part.id;
  console.debug(`[server] Part for project ${project_name} already exists`);
}
}
```

Listing 2: Creation of DevRev Part

```
// create work in DevRev if not already existing
const works_response = await apiUtils.getWorks(dev_user_id);
let work_id = "";
if (works_response.success) {
const works = works_response.data.works;
const work = works.find((w: Work) => w.title === title);
if (!work) {
  const work_response = await apiUtils.createWork({
    type: type,
    applies_to_part: part_id,
    owned_by: [dev_user_id],
     title :  title ,
    body: body,
  });

  work_id = work_response.data.work.id;
  if (work_response.success) {
    console.debug(`[server] Created work for ${title} successfully `);
  } else {
    console.error(
      `[server] Failed to create work for ${title}. Err: ${work_response.message}`,
    );
  }
} else {
  work_id = work.id;
  console.debug(`[server] Work for ${title} already exists `);
}
}
```

Listing 3: Creation of DevRev Work (Issue/Task)

```
// create a timelineEvent for the work
const timeline_event_response = await apiUtils.createTimeLine({
type: TimelineEntriesCreateRequestType.TimelineComment,
object: work_id,
body: body,
});
if (timeline_event_response.success) {
console.debug(
  `[server] Created timeline event for work ${work_id} successfully`,
);
} else {
console.error(
  `[server] Failed to create timeline event for work ${work_id}. Err: ${timeline_event_response.message}`,
);
}
```

Listing 4: Creation of DevRev Timeline Event

4.2 AI Insights Generation

The AI insights component leverages a multi-agent architecture to provide in-depth analysis [Luc24] and actionable recommendations for build failures [Any]. The process is decomposed into several distinct stages:

4.2.1 Data Acquisition and Preprocessing

- **Data Retrieval:** Build logs are pulled from CircleCI using authenticated API calls.

- **Preprocessing:** The logs undergo cleaning and formatting, where noise is filtered out, and relevant error patterns are highlighted.

4.2.2 Agent-Based Analysis

The system employs a sequence of specialized AI agents, each with a distinct role:

1. **Issue Analyzer Agent:**
   - **Purpose:** To parse and understand build logs, extracting error messages and identifying potential failure points [Ale24] [Alt24].
   - **Inputs:** Preprocessed build logs, context of the codebase, and build pipeline metadata.
   - **Output:** A list of identified issues along with a preliminary classification of failure types.

```
export class IssueAnalyzerAgent {
  private llmUtils: LLMUtils;

  constructor(llmUtils: LLMUtils) {
    this.llmUtils = llmUtils;
  }

  async processArtifacts(artifacts: Artifact[]): Promise<IssueAnalysis> {
    const safeStringify = (data: any) => {
      try {
        return typeof data === "string" ? data : JSON.stringify(data);
      } catch {
        return "[Non-serializable data]";
      }
    };

    const combinedData = artifacts
      .map(artifact => `=== ${artifact.path} ===\n${safeStringify(artifact.data)}`)
      .join('\n\n');

    const systemPrompt = `You are an expert CI/CD issue analyzer. Your task is to:
1. Identify and extract error messages from build logs
2. Categorize issues into errors, warnings, performance issues, and security concerns
3. Provide a concise summary of the key problems

Return your analysis in JSON format with these fields:
- errors: string[]
- warnings: string[]
- performanceIssues: string[]
- securityConcerns: string[]
- summary: string`;

    const humanPrompt = `Analyze these build artifacts:
{input}`;

    const response = await this.llmUtils.chatCompletion(
      systemPrompt,
      humanPrompt,
      { input: combinedData }
    );

    return response as IssueAnalysis;
  }
}
```

Listing 5: Prompt and Definition of IssueAnalyzerAgent

2. **Solution Recommender Agent:**
   - **Purpose:** To analyze the issues flagged by the Issue Analyzer and generate tailored recommendations for troubleshooting and resolving the failures [Cli] [Mic24] [Tas].

- **Inputs:** The list of issues, additional context from the codebase, and the build pipeline configuration.
- **Output:** A set of actionable recommendations that might include code changes, configuration tweaks, or further diagnostic steps.
- **Future Enhancement:** Integration with CircleCI API to automatically trigger corrective actions, pending user confirmation.

```
export class SolutionRecommenderAgent {
  private llmUtils: LLMUtils;

  constructor(llmUtils: LLMUtils) {
    this.llmUtils = llmUtils;
  }

  async generateRecommendations(analysis: IssueAnalysis): Promise<SolutionRecommendation[]> {
    const systemPrompt = `You are a CI/CD solutions expert. Given a set of issues:
1. Provide actionable recommendations for each major issue
2. Include specific commands or configuration changes where applicable
3. Estimate your confidence level (low, medium, high) for each recommendation

Return an array of recommendations in JSON format with:
- issue: string
- recommendedSolution: string
- confidenceLevel: string`;

    const humanPrompt = `Analyze these CI/CD issues and provide solutions:
{input}`;

    const response = await this.llmUtils.chatCompletion(
      systemPrompt,
      humanPrompt,
      { input: JSON.stringify(analysis) }
    );

    return response as SolutionRecommendation[];
  }
}
```

Listing 6: Prompt and Definition of SolutionRecommenderAgent

3. **Solution Verifier Agent:**

   - **Purpose:** To validate the recommendations produced by the Solution Recommender by cross-checking them against known resolution patterns or historical data.
   - **Inputs:** Both the identified issues and the proposed solutions.
   - **Output:** A confidence score or verification status indicating whether the recommendations are likely to resolve the issues.

```
export class SolutionVerifierAgent {
  private llmUtils: LLMUtils;
  private historicalSolutions : Record<string, string[]> = {
    "dependency error": ["Update package versions", "Clear cache and retry", "Check compatibility matrix"],
    "timeout": ["Increase timeout limit", "Optimize test suite", "Split into smaller jobs"],
    "memory issue": ["Increase resource allocation", "Optimize memory usage", "Enable swap space"]
  };

  constructor(llmUtils: LLMUtils) {
    this.llmUtils = llmUtils;
  }

  async verifySolutions(recommendations: SolutionRecommendation[]): Promise<VerifiedSolution[]> {
    const systemPrompt = `You are a CI/CD solution verifier. For each recommended solution:
1. Verify if it matches known patterns from historical data
2. Assess the likelihood of the solution being correct
3. Provide verification status (confirmed, likely , uncertain)
4. Include any matching historical solutions

Return verified solutions in JSON format with:
- issue: string
- recommendedSolution: string
- confidenceLevel: string
- verificationStatus: string
- historicalMatches?: string[]`;

    const humanPrompt = `Verify these recommended solutions against historical patterns:
{recommendations}

Known historical patterns:
{historicalData}`;

    const verifiedSolutions: VerifiedSolution[] = [];

    for (const recommendation of recommendations) {
      const response = await this.llmUtils.chatCompletion(
        systemPrompt,
        humanPrompt,
        {
          recommendations: JSON.stringify(recommendation),
          historicalData: JSON.stringify(this.historicalSolutions)
        }
      ) as VerifiedSolution;
      verifiedSolutions.push(response);
    }

    return verifiedSolutions;
  }
}
```

Listing 7: Prompt and Definition of SolutionVerifierAgent

4.2.3 Agent Communication and Feedback Loop

- The agents operate in a chained manner, where the output of one agent serves as the input for the next.
- A dedicated communication protocol ensures that agents can pass contextual metadata (e.g., confidence levels, processing logs) efficiently.
- In cases where the Solution Verifier flags low confidence in recommendations, the system can trigger a re-analysis, prompting the Issue Analyzer to revisit the logs or asking for manual intervention.

4.2.4 Integration with DevRev

After the agents have processed the data:

- The final insights, including issue summaries and troubleshooting recommendations, are posted as comments on the relevant DevRev issue.
- This process closes the feedback loop by enabling developers to view AI-generated insights directly in the DevRev interface, thus streamlining the debugging and resolution workflow.

4.2.5 Future Directions in Agentic AI

To further enhance the system, planned improvements include:

- **Automated Execution:** Linking the Solution Recommender Agent with CircleCI to automatically apply fixes once recommendations are verified.
- **Self-Learning Capabilities:** Implementing a feedback mechanism where successful resolutions are fed back into the AI models to improve future performance.

- **Enhanced Error Handling:** Developing more robust fallback strategies to handle cases where the AI agents may produce ambiguous or conflicting recommendations.
- **Visualization Tools:** Integrating real-time dashboards to monitor the performance of each agent and the overall system health.

4.3 Mapping Between Systems

The document defines the following mappings:

- **CircleCI Project** → **DevRev Part**: Ensures that the project information is encapsulated in DevRev.
- **CircleCI Workflow** → **DevRev Issue**: Each workflow run generates a timeline event in DevRev.
- **CircleCI Job** → **DevRev Task**: Job-level details are tracked as tasks.

5. Testing and Deployment

The codebase can be found at: CircleCI-Snap-In.

The video demonstration of the Snap-In can be found here.

5.1 Local Testing

Local tests can be performed by simulating events. Test events, including API keyring values, are added under `src/fixtures`:

- Run `npm install` to install dependencies.
- Execute tests using `npm t`.

5.1.1 Test Cases

Description of Test Cases in codebase:

Table 1: Test Cases for Generate Insights Function

Test Suite	**Test Cases**
Generate Insights Function	Main function execution (skips if API key missing)
IssueAnalyzerAgent Tests	Process artifacts with error logs Handle timeout issues in build logs Handle memory-related issues
SolutionRecommenderAgent Tests	Generate recommendations for dependency issues Generate recommendations for timeout issues
SolutionVerifierAgent Tests	Verify solutions against historical patterns Verify memory issue solutions
LLM as Judge Test	Evaluate the quality of agent outputs

Table 2: Test Cases for Sync CircleCI Data Function

Test Suite	**Test Cases**
Webhook of type-1	workflow-completed for GitHub OAuth and Bitbucket Cloud
Webhook of type-2	job-completed for GitHub OAuth and Bitbucket Cloud
Webhook of type-3	workflow-completed for GitLab, GitHub App and Bitbucket Data Center
Webhook of type-4	job-completed for GitLab, GitHub App and Bitbucket Data Center

5.2 Activating the Snap-In

After successful local testing, follow these steps:

1. Authenticate with the DevRev CLI:

```
source .env
echo $DEVREV_TOKEN | devrev profiles set-token --org $DEV_ORG --usr $USER_EMAIL
```

2. Start the test server:

```
cd ./code
npm run test:server
```

3. Create a snap-in version:

```
devrev snap_in_package create-one --slug <slug_name>
devrev snap_in_version create-one --manifest ./manifest.yaml --testing-url <url>
```

4. Install the Snap-In Draft and configure required keys as prompted.

6. Future Work and Enhancements

Future enhancements include:

- Addition of automatic execution of recommendations pending user acknowledgement on the recommender agent side.
- Extending integrations to include other tools like VSCode, Jenkins, and Replit.
- Implementing additional AI agents to cover code generation, debugging, and release note generation.
- Enhancing error handling and robustness in the data synchronization process.
- Incorporating dynamic configuration of webhooks and secret management.

7. Conclusion

The CircleCI Snap-In project provides a robust solution for synchronizing CI/CD data between CircleCI and DevRev, enabling a unified view of development activities and enhancing productivity through AI-generated insights. The modular architecture not only simplifies the integration process but also lays the foundation for future enhancements in the agentic systems domain.

References

[1] P. Kumar, *Ai agents for software development*, Accessed: March 18, 2025, 2025. [Online]. Available: https://www.saffrontech.net/blog/ai-agents-for-software-development.

[2] Lucas, *The role of ai agents for developers*, Accessed: March 18, 2025, 2024. [Online]. Available: https://composio.dev/blog/ai-agents-for-developers-role/.

[3] A. Anywhere, *Agentic workflows in automation anywhere*, Accessed: March 18, 2025. [Online]. Available: https://www.automationanywhere.com/rpa/agentic-workflows.

[4] Alex, *Ai agents in sdlc*, Accessed: March 18, 2025, 2024. [Online]. Available: https://waydev.co/ai-agents-in-sdlc/.

[5] A. Altus, *Genai agents and workflows in software development*, Accessed: March 18, 2025, 2024. [Online]. Available: https://outshift.cisco.com/blog/genai-agents-agentic-workflow-software-development.

[6] ClickUp, *Ai release notes generator by clickup*, Accessed: March 18, 2025. [Online]. Available: https://clickup.com/features/ai/release-notes-generator.

[7] Microsoft, *Generating release notes with genai*, Accessed: March 18, 2025, 2024. [Online]. Available: https://microsoft.github.io/genaiscript/blog/creating-release-notes-with-genai/.

[8] Taskade, *Taskade release notes generator*, Accessed: March 18, 2025. [Online]. Available: https://www.taskade.com/generate/programming/release-notes.