National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Vinay Nukalapati
Student ID: 24106470

School of Computing
National College of Ireland

Supervisor: Diego Lugones

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. VINAY NUKALAPATI ……

Student ID: ……24106470……..……

Programme: ……Cloud Computing…… **Year:** …2025……..

Module: ……MSc Research Project ……..……

Lecturer: …… Diego Lugones ……..……

Submission Due Date: ……28/01/2026……..……

Project Title: Self-Healing CI/CD Pipeline Using LLM and Google ADK Framework

Word Count: ……2067…… **Page Count:** ……10……..……..……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……VINAY NUKALAPATI ……

Date: ……28/01/2026……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

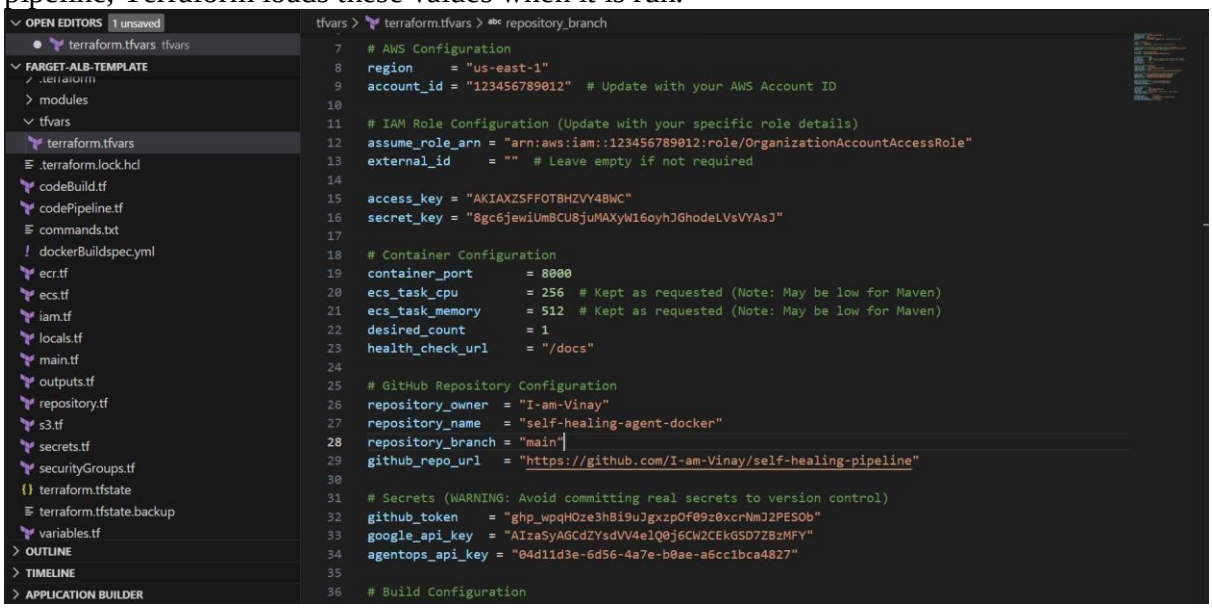
Configuration Manual

Vinay Nukalapati
Student ID: 24106470

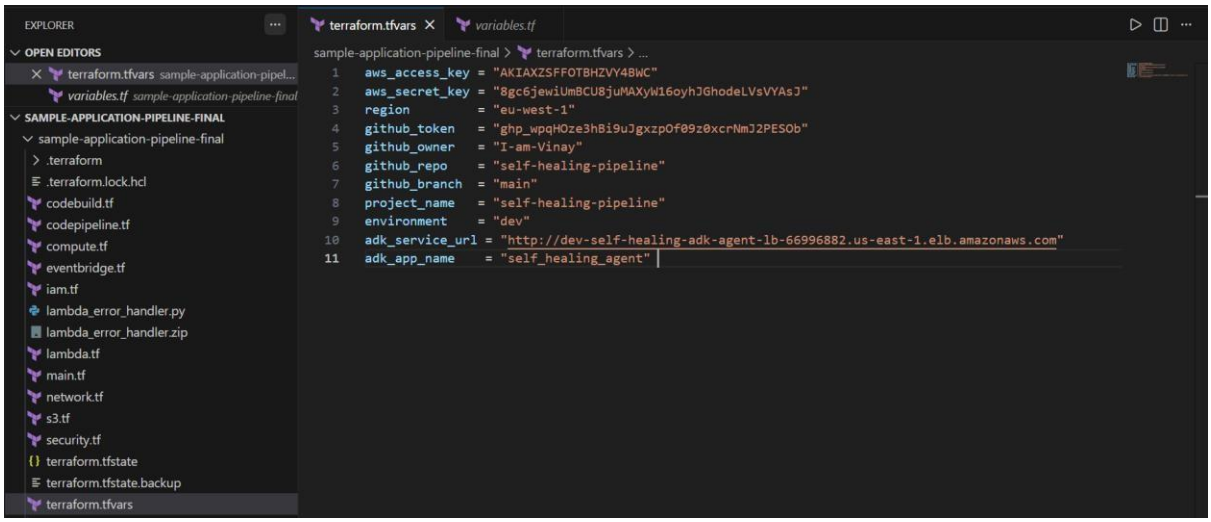
1 Understanding the configuration file terraform.tfvars

One of the important configuration files that Terraform uses to provide input values to a defined variable in code is the terraform.tfvars file. Terraform does not require sensitive or environment specific values to be hardcoded within .tf files but they can be placed within a special file named .tfvars. This enhances the security, maintainability, reusability.

All the configuration parameters needed to deploy the ADK multi-agent self-healing service in AWS ECS Fargate are provided in this terraform.tfvars file. In order to configure VPC networking, an Application Load Balancer, ECS cluster, ECR repository, and deployment pipeline, Terraform loads these values when it is run.

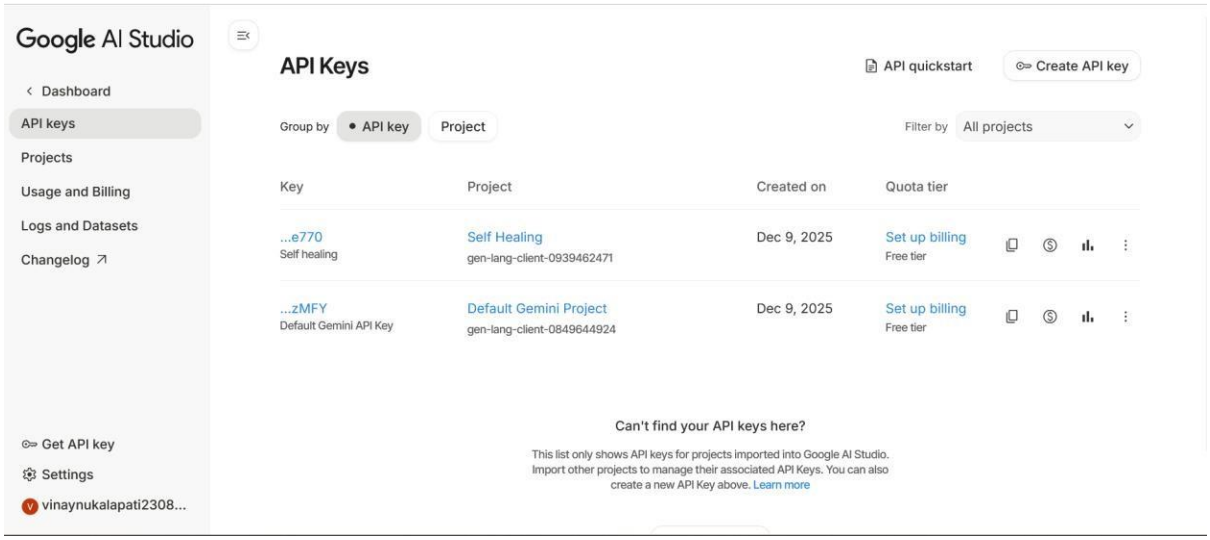


This terraform.tfvars file contains configuration values required to set up the CI/CD pipeline responsible for building, testing, and deploying the sample application. The pipeline integrates AWS CodePipeline, CodeBuild, Lambda, EventBridge, CloudWatch Logs, IAM roles, EC2, and S3.



2 Creating and Setting Up the Gemini API Key

Large Language Model (LLM) functionality in Self-Healing ADK multi-agent service can only be enabled using a Gemini API key. It is a key that enables the ADK workflow to provide error logs, accept remediation recommendations, and iterate reasoning. The API key has to be generated in Google AI studio and has to be set up safely as an environment variable within the terraform.tfvars file.



Follow the steps below to create and set up the Gemini API key.

Step 1: Visit Google AI Studio

Go to the official Google AI Studio portal:
https://aistudio.google.com/
You must be logged in with a valid Google account to continue.

Step 2: Navigate to the API Keys Section

1. On the left menu, select API keys.
2. Click Create API key.
3. Choose “Create API key in new project” or select an existing Google Cloud project.

This automatically generates a new AI API key associated with the project.

Step 3: Copy the Generated API Key

1. Once created, the key will be displayed only once.
2. Click Copy and store it securely.

Step 4: Enable Billing (If Required)
Some Gemini models require an active billing account.
Steps:

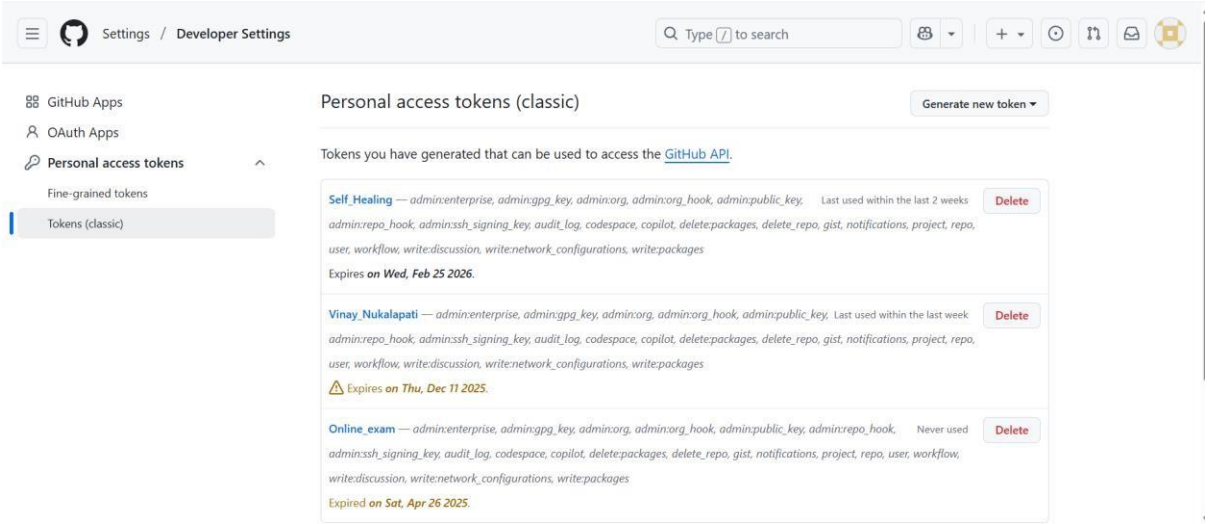
1. Go to Google Cloud Console → Billing.
2. Attach a billing account to the selected project.
3. Ensure the "AI Services / Generative AI" API is enabled under APIs & Services.

Step 5: Add the Gemini API Key to ADK Environment Variables

1. Add its value in terraform.tfvars

3 GitHub Access Key Setup

A GitHub Personal Access Token (PAT) is required to authenticate AWS CodePipeline and the ADK Self-Healing service with GitHub. This token enables cloning repositories, accessing source code, and creating pull requests during the remediation workflow.



Follow the steps below to create the GitHub Access Key.

Step 1: Generate GitHub Access Token

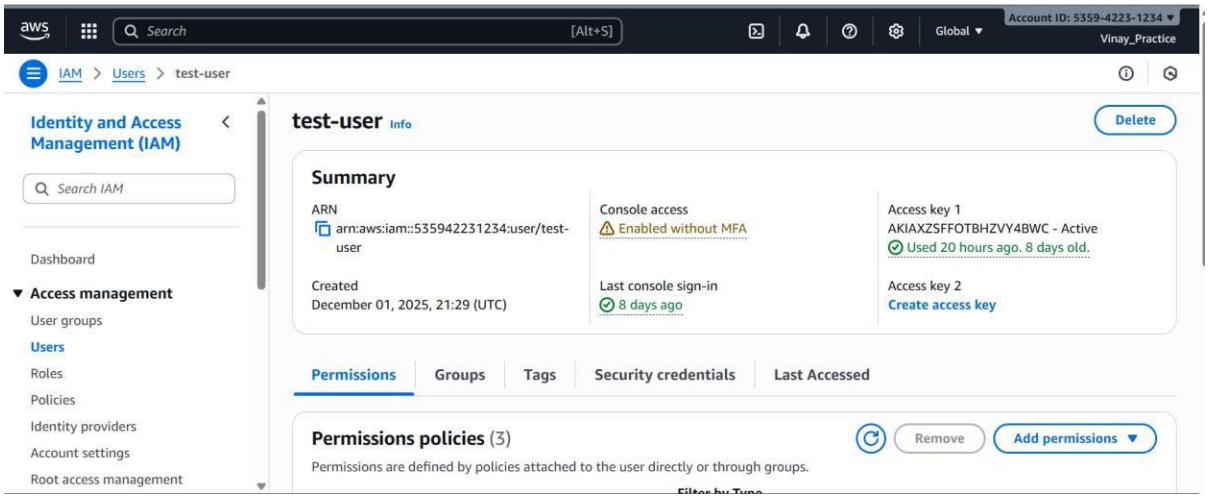
1. Log in to GitHub and navigate to
   Settings → Developer Settings → Personal Access Tokens → Tokens (classic)
2. Click Generate new token.
3. Provide a name for the token (example: self-healing-ci-token).
4. Select all the required scopes
5. Set token expiration and click Generate Token.
6. Copy the token immediately as it will not be shown again

Step 2: Configure the Token in the Project
Add the copied token to both the Terraform configuration files.

In terraform.tfvars (CI/CD Pipeline Project) and In terraform.tfvars (ADK Fargate Deployment)

4 AWS Access Key and Secret Key Setup

In order to allow CLI and ADK deployment pipeline to communicate with AWS services, it is necessary to generate an AWS Access Key ID and Secret Access Key. These credentials enable the secure programmatic access of AWS resources including ECS, Lambda, Code Pipeline, S3 as well as CloudWatch. An operating AWS cloud account is required to carry out the following steps with access to IAM
services.

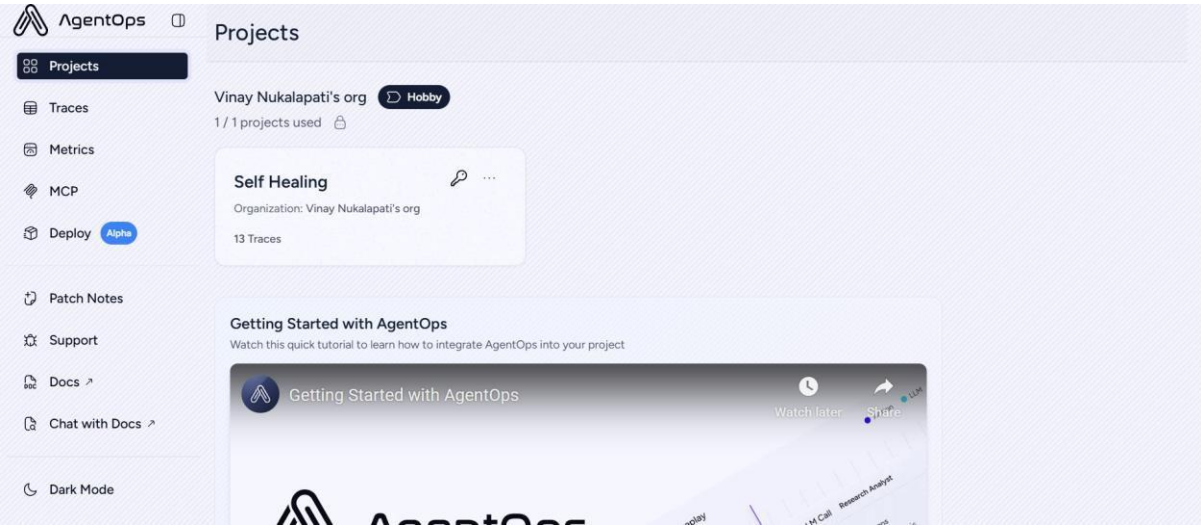


Steps:

1) Log in to AWS Console:
   - Navigate to https://console.aws.amazon.com
2) Create an IAM User:
   - Go to IAM > Users > Add User
   - Choose Programmatic access
   - Attach policies like Administrator Access (or custom least-privilege policies)
3) Create Access Keys:
   - Under IAM tab on the application menu, go to Users > [Your User] > Security credentials
   - Click Create access key
   - Select the use case to be the Command Line Interface (CLI).
   - Access the credentials – Access key ID, Secret access key.
   - Add the accessed credentials to the terraform.tfvars files.

5 AgentOps API Key Setup

The ADK workflow employs the use of AgentOps to monitor agent performance, tracing on session, debug logs, and metrics of execution. In order to do this, an AgentOps API key should be generated and set as an environment variable in the ADK service.



Follow the steps below to create and set up the AgentOps API key.

Steps:

1) **Create AgentOps Account**
   - Visit the AgentOps website: https://agentops.ai/
   - Sign up using your Google or GitHub account.
   - Log in to the AgentOps dashboard.
2) **Complete Initial Project Setup**
   - Once logged in, AgentOps will prompt for basic project information such as:
     - Project name
     - Project description
     - Intended use case
   - Fill in the required details and click Continue.
   - AgentOps will automatically create a default project for your workspace.

3) **Obtain the Default Project API Key**
   - Navigate to API Keys section.
   - AgentOps automatically provides a default API key for the project.
   - Copy the API key for use in the ADK deployment.

This API key provides authentication to the AgentOps service. Add the AgentOps API Key to terraform.tfvars file.

6 **Understanding ADK Commands:** adk web and adk api_server

The Agent Development Kit (ADK) provides built-in CLI commands that allow developers to run the agent workflow either through a web UI or a REST API server. These commands are used during development, debugging, and deployment of the self-healing multi-agent system.

The two most commonly used commands in this project are:

- **adk web** → Runs the ADK workflow in a local browser-based interface
- **Purpose:**

This command launches a local web-based interface to manually test or run workflows.
It is mainly used during development to visually inspect how agents behave, test log inputs, or validate agent outputs.

- **adk api_server** → Runs the ADK application as an API service, required for AWS Lambda integration
  Purpose:
  This command starts the ADK workflow as an HTTP-based API service.
  Your AWS Lambda function requires this mode to send error logs to the ADK system for automated remediation.

7 How to Check Logs on ADK Agents

The ADK self-healing service runs inside an ECS Fargate container, and all logs generated by the agents (log analyzer, repository setup agent, fix loop, PR creator, etc.) are automatically sent to AWS CloudWatch Logs. These logs are important for debugging issues, verifying that the agents executed successfully, and checking the actions taken by each agent.

Follow the steps below to view the agent logs.

Step 1: Navigate to CloudWatch in AWS Console

1. Log in to AWS Console
2. Go to CloudWatch service
3. On the left panel, click Logs → Log groups

Step 2: Open the Latest Log Stream
Every ECS task creates a log stream.
Click the most recent log stream to view real-time logs coming from the ADK container.
This contains:

- Workflow start and end logs
- AgentOps initialization logs
- Build error messages received from Lambda
- Actions taken by each agent
- Fix attempts and loop iterations
- GitHub PR creation logs
- Any Python exceptions or failures

8 Agent Workflow

In the ADK (Agent Development Kit) system, the workflow is designed around multiple agents that work together to complete a task. Each agent performs a specific action such as log analysis, code fixing, build verification, etc.

The entire workflow relies on shared state and three main agent types:

1. Sequential Agent
   A SequentialAgent runs agents one after another, in a fixed order.
   How it works:
   1. Agent A runs

2. Its output is stored in the state
3. Agent B reads the updated state and runs
4. Agent C runs next
5. … and so on

Why it is used:

- Ensures predictable, step-by-step execution
- Perfect for workflows with clear stages

2. Parallel Agent

A ParallelAgent runs multiple agents at the same time when tasks do not depend on each other.

How it works:

- Multiple agents execute concurrently
- Their outputs are merged into the state
- The workflow continues after all parallel tasks finish

When used:

- When independent checks or tasks can run simultaneously
- To reduce overall processing time

3. Loop Agent

A LoopAgent repeats a set of agents until a condition is met.

How it works:

1. Run a group of agents
2. Check a condition (e.g., did the build succeed?)
3. If NO → repeat the loop
4. If YES → exit the loop

Why it is used:

- To allow iterative problem solving
- Useful when multiple attempts may be needed

9 Agents and Tools Used in the System

Self-Healing CI/CD system is a multi-agent system based on the Agent Development Kit (ADK). All the agents involved in the workflow execute a particular task e.g. log analysis, repository setup, error correction, build verification or finalizing the fix. The agents operate a package of tools, which are permission-restricted and small functions that enable the agents to communicate with files, GitHub, build systems, and other resources.

Below is the list of agents and tools used in the system.

9.1 Agents Used in the System

1. Log Analyzer Agent

- Extracts useful information from error logs
- Identifies error type, file path, and stack trace
- Writes structured error data into the workflow state

2. Repository Setup Agent

- Clones the GitHub repository
- Creates a new fix branch
- Detects the build system (Maven, Gradle, npm, etc.)
- Stores repository path and build commands in the state

3. Error Remediation Agent

- Reads affected files
- Modifies code to fix errors
- Installs missing dependencies
- Writes fix information into the state

4. Build Verification Agent

- Executes the detected build command
- Captures new errors and success/failure status
- Parses build logs and updates the iteration count

5. Loop Decision Agent

- Evaluates build results and iteration count
- Decides if the loop should continue or stop
- Triggers exit when build is successful or attempts are exhausted

6. Code Finalizer Agent

- Stages and commits updated files
- Pushes the fix branch to GitHub
- Creates a pull request summarizing the fix

9.2 Tools Used in the System

Each agent relies on ADK tools — restricted functions that define what the agent is allowed to do.

Below are the major tools grouped by category.

A. Log Analysis Tools

Used by the Log Analyzer agent.

1. build_log_storage_tool

Stores incoming log segments for reference.

2. parse_error_details_tool

Extracts error type, category, and line information.

3. extract_stack_trace_tool

Identifies stack trace segments from logs.

4. extract_file_info_tool

Detects which file caused the error.

5. categorize_error_type_tool

Classifies the error as syntax, import, type, dependency, etc.

B. Repository Setup Tools

Used by the Repository Setup agent.

1. get_github_config_tool

Loads GitHub repo URL and token from environment.

2. git_clone_tool

Clones the target repository into a working directory.

3. git_create_branch_tool

Creates a fix branch (e.g., self-healing-fix-123).

4. detect_build_system_tool

Identifies whether the project uses Maven, Gradle, npm, or pip.

5. list_directory_tool

Lists files in the repo for scanning or detection.

C. Error Remediation Tools

Used by the Error Remediation agent.

1. read_file_tool

Reads contents of source files for analysis.

2. write_file_tool

Writes corrected code back to the file.

3. install_dependency_tool

Adds missing dependencies using appropriate package managers.

4. list_directory_tool

Assists in navigating project structure.

D. Build Verification Tools

Used by the Build Verification agent.

1. run_build_tool

Runs the build using detected commands (mvn, gradle, npm).

2. parse_build_output_tool

Extracts success/failure, new errors, and logs.

3. build_log_storage_tool

Keeps build logs for iterative debugging.

E. Loop Control Tools

Used by the Loop Decision Agent.

1. exit_loop_tool

Ends the fix loop when:

- Build succeeds
- No progress is detected
- Max iterations reached

F. Code Finalization Tools

Used by the Code Finalizer Agent.

1. git_add_files_tool

Stages modified files for commit.

2. git_commit_tool

Commits changes to the local repository.

3. git_push_tool

Pushes commit to GitHub on the fix branch.

4. create_pull_request_tool

Creates a PR summarizing:

- Error
- Fix applied
- Build verification results

References

HashiCorp. (2024) *Terraform Documentation*. Available at: https://developer.hashicorp.com/terraform/docs

Amazon Web Services. (2024) *IAM User Guide: Creating Access Keys*. Available at: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html

Amazon Web Services. (2024) *Amazon ECS Developer Guide*. Available at: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/

Amazon Web Services. (2024) *AWS Lambda Developer Guide*. Available at: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

GitHub. (2024) *Managing Your Personal Access Tokens*. Available at: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens

Google Cloud. (2024) *Setting Up API Keys*. Available at: https://cloud.google.com/docs/authentication/api-keys

Google ADK. (2024) *Agent Development Kit Documentation*. Available at: https://github.com/google/ai-agent-development-kit

AgentOps. (2024) *AgentOps Platform Documentation*. Available at: https://www.agentops.ai/docs

Amazon Web Services. (2024) *AWS CodePipeline User Guide*. Available at: https://docs.aws.amazon.com/codepipeline/latest/userguide/