Meta Reality Labs Developing for Continuous Integration

Executive Summary

The document "Developing for Continuous Integration" provides a comprehensive guide to developing software using continuous integration (CI) practices. The document covers various topics, including understanding the CI workflow, choosing a build system, testing stages, logging systems, and landing code.

The document explains that there are three build systems available within the RLDI AOSP Supported Workflow: Skycastle, RBE, and Chesterfield. It provides an overview of each build system, highlighting their features and benefits. The document also discusses the importance of testing in the development process and introduces various testing frameworks, such as Jest-E2E, Unified CI, and Breakpad.

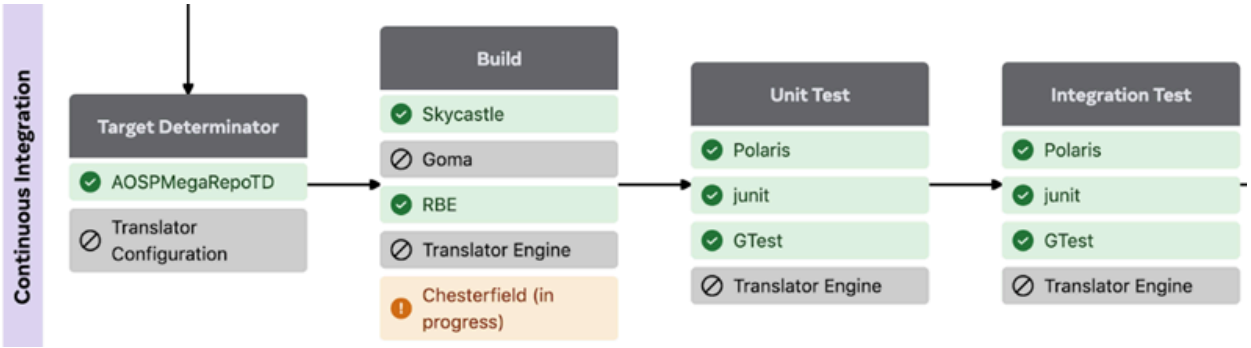
In addition, the document covers logging systems, including Logview, Log Insights, and RL Crash Task Controls. It provides an overview of each logging system and highlights their features and benefits. Finally, the document discusses the process of landing code using Picknic, which is the only choice within the RLDI AOSP Supported Workflow.

Overall, the document provides a comprehensive guide to developing software using continuous integration practices, covering various topics related to building, testing, logging, and landing code.

Understanding the CI Workflow Swim Lane

This page describes the stage you need to follow to complete the CI integration, based on the choices you selected and installed with the Supported Workflow Tool.

When working with the Supported Workflow Tool for continuous integration, you are dealing with the Continuous Integration swim lane, as partially shown in the following image:



Next, you will walk through the stages in the continuous integration swim lane.

Continuous Integration Swim Lane

The **Continuous Integration** swim lane of the Supported Workflow tool helps you select the necessary technologies for your continuous integration efforts.

First, you need to identify a target with the Target Determinator (`AOSPMegaRepoTD`).

Using the Target Determinator

A **target determinator** is a tool used in software development to determine which builds, tests, and other actions need to be executed based on changes made to the codebase. It helps automate the process of identifying affected targets and scheduling jobs accordingly. The only target determinator available in the AOSP Supported Workflow is

`AOSPMegaRepoTD`. Once a build run is scheduled and a manifest is selected, the "target determinator" is run to determine which targets and flavors should actually be built.

Follow these steps to use `AOSPMegaRepoTD` to determine a target:

1. Create a product config, similar to this one.
2. Create a translator target config, similar to this one.
3. Diff jobs should be triggered when relevant diffs are created.
4. For continuous build pipeline, you'd need to onboard onto continuous build pipeline, see the wiki here.

Building Your Code

When you build your code, you have the option of selecting Skycastle, RBE, or Chesterfield (in progress) within the AOSP Supported Workflow.

Skycastle Builds

Skycastle is an engine for defining continuous integration (CI) workflows. Skycastle is a feature of Sandcastle, which is described in Implementing CI with Sandcastle.

Skycastle is a platform for defining and executing CI/CD workflows at Meta. It allows developers to create, test, and debug complex workflows in a declarative format using Starlark, a Python-like language.

To build with Skycastle, you can follow these general steps:

1. Create a workflow file: Write a Skycastle workflow file (`.sky` file) that defines the actions and dependencies for your build process. You can use the `arc skycastle schedule` command to schedule a workflow run.
2. Define actions: In your workflow file, define the individual actions that make up your build process. These actions can include running scripts, compiling code, or executing tests.
3. Specify dependencies: Define the dependencies between actions, so that Skycastle knows which actions need to be executed before others can start.
4. Run the workflow: Use the `arc skycastle schedule` command to schedule a workflow run. This will trigger the execution of your build process on the Skycastle platform.
5. Monitor the workflow: You can monitor the progress of your workflow run on the Skycastle UI, where you can see the status of each action and any logs or error messages.
6. Debug and iterate: If there are issues with your workflow, you can use the Skycastle UI to debug and iterate on your workflow file.

Some benefits of building with Skycastle include:

- Declarative syntax: Skycastle's Starlark syntax makes it easy to define complex workflows in a clear and concise way.
- Parallelism: Skycastle can execute multiple actions in parallel, making your build process faster and more efficient.
- Caching: Skycastle caches the results of actions, so that if an action has already been executed with the same inputs, it can be reused instead of re-executed.

- Integration with Meta tools: Skycastle integrates seamlessly with other Meta tools, such as Phabricator, Sandcastle, and Buck.

For more information on building with Skycastle, you can check out the official documentation at https://www.internalfb.com/intern/wiki/Skycastle/.

RBE Builds

Remote Build Execution (RBE) for AOSP is a system that allows you to offload Android builds to a remote cluster of powerful machines. This can significantly speed up the build process, especially for large projects like Android.

To build with RSB, you can follow these general steps:

1. **Create an RSB configuration file**: Write an RSB configuration file (.rsb file) that defines the actions and dependencies for your build process. This file specifies the commands to execute, the required inputs and outputs, and any additional metadata.
2. **Define actions**: In your RSB configuration file, define the individual actions that make up your build process. These actions can include running scripts, compiling code, or executing tests.
3. **Specify dependencies**: Define the dependencies between actions, so that RSB knows which actions need to be executed before others can start.
4. **Schedule the build**: Use the rsb command-line tool to schedule a build with RSB. This will trigger the execution of your build process on remote workers.
5. **Monitor the build**: You can monitor the progress of your build on the RSB UI, where you can see the status of each action and any logs or error messages.
6. **Debug and iterate**: If there are issues with your build, you can use the RSB UI to debug and iterate on your RSB configuration file.

Some benefits of building with RSB include:

- **Scalability**: RSB allows you to scale out your build process across multiple machines, making it faster and more efficient.
- **Parallelism**: RSB can execute multiple actions in parallel, taking advantage of the distributed nature of the system.

- **Caching**: RSB caches the results of actions, so that if an action has already been executed with the same inputs, it can be reused instead of re-executed.
- **Integration with Meta tools**: RSB integrates seamlessly with other Meta tools, such as Buck and Sandcastle.

For more information on building with RSB, you can check out the official documentation at https://www.internalfb.com/intern/wiki/Remote_Execution/Service_Bus/.

Note: RSB is currently being deprecated in favor of Skycastle, which provides similar functionality but with a more modern and flexible architecture. If you're starting a new project, it's recommended to use Skycastle instead of RSB.

Chesterfield Builds

Chesterfield is a CI/CD build service that schedules or fetches required builds, feeds build artifacts into Test Infra for validation, and into Signal Aggregator for monitoring. Chesterfield is part of a larger Reality Labs CI/CD redesign to migrate off of Translator.

To build with Chesterfield, you can follow these general steps:

1. **Create a Chesterfield configuration file**: Write a Chesterfield configuration file (`.chesterfield` file) that defines the actions and dependencies for your build process. This file specifies the commands to execute, the required inputs and outputs, and any additional metadata.
2. **Define actions**: In your Chesterfield configuration file, define the individual actions that make up your build process. These actions can include running scripts, compiling code, or executing tests.
3. **Specify dependencies**: Define the dependencies between actions, so that Chesterfield knows which actions need to be executed before others can start.
4. **Schedule the build**: Use the `chesterfield` command-line tool to schedule a build with Chesterfield. This will trigger the execution of your build process on remote workers.
5. **Monitor the build**: You can monitor the progress of your build on the Chesterfield UI, where you can see the status of each action and any logs or error messages.

6. **Debug and iterate**: If there are issues with your build, you can use the Chesterfield UI to debug and iterate on your Chesterfield configuration file.

Some benefits of building with Chesterfield include:

- **Scalability**: Chesterfield allows you to scale out your build process across multiple machines, making it faster and more efficient.
- **Parallelism**: Chesterfield can execute multiple actions in parallel, taking advantage of the distributed nature of the system.
- **Caching**: Chesterfield caches the results of actions, so that if an action has already been executed with the same inputs, it can be reused instead of re-executed.
- **Integration with Meta tools**: Chesterfield integrates seamlessly with other Meta tools, such as Buck and Sandcastle.

For more information on building with Chesterfield, you can check out the official documentation at https://www.internalfb.com/intern/wiki/Chesterfield/.

Choosing a Build System

Chesterfield is currently being deprecated in favor of Skycastle, which provides similar functionality but with a more modern and flexible architecture. If you're starting a new project, it's recommended to use Skycastle or RSB instead of Chesterfield. But which build system should you use? It depends on your specific use case and requirements. Both Skycastle and RSB are powerful tools for building and testing code, but they have different strengths and weaknesses.

Skycastle is a platform for defining and executing CI/CD workflows. It allows you to define a series of actions that should be executed in a specific order, and provides features such as caching, parallelism, and error reporting. Skycastle is particularly useful when you need to perform complex builds or tests that involve multiple steps, or when you need to integrate with other tools and services.

RSB, on the other hand, is a distributed build system that allows you to execute builds remotely on a cluster of machines. It is designed to speed up builds by distributing the

workload across multiple machines, and can be particularly useful when you have large codebases or complex builds that take a long time to complete.

In general, if you have a simple build process that involves compiling a few files and running some tests, RSB may be a good choice. However, if you have a more complex build process that involves multiple steps, dependencies, or integrations with other tools and services, Skycastle may be a better fit.

Ultimately, the choice between Skycastle and RSB will depend on your specific needs and requirements. You may want to experiment with both tools and see which one works best for your particular use case.

Testing Stages

This section describes the testing stages when developing for continuous integration.

Unit Testing

Unit testing is explained in Testing and Debugging.

Integration Testing

Integration testing is explained in Testing and Debugging.

End-to-End Testing

As described in Testing and Debugging, end-to-end testing (E2E testing) refers to a software development technique where an application’s workflow is validated from start to finish.

RLDI supports Jest-E2E, Unified CI, and Breakpad for end-to-end testing, as described in the following sections:

Jest-E2E

Jest-E2E is a testing framework developed by Meta that allows you to write end-to-end tests in JavaScript. It is used to test the functionality of web applications, mobile apps, and other software systems. For detailed information on Jest-E2E, refer to Jest-E2E section of the Testing and Debugging wiki.

Unified CI

Unified CI (Continuous Integration) testing is a framework used at Meta to automate the testing of code changes across multiple platforms and applications. It provides a unified way to run tests, collect test results, and provide feedback to developers.

Some benefits of using Unified CI testing include:

- Cross-platform testing: Unified CI allows you to test your code on multiple platforms, ensuring that it works as expected across different environments.
- Automated testing: Unified CI automates the testing process, saving time and effort for developers.
- Integration with Meta tools: Unified CI integrates seamlessly with other Meta tools, such as Sandcastle and Phabricator.

To use Unified CI testing, you can follow these general steps:

1. **Create a test configuration file**: Write a test configuration file (`.testconfig` file) that defines the tests to be run, the platforms to be tested on, and any additional metadata.
2. **Write test cases**: In your test configuration file, define the individual test cases that exercise specific parts of the code being tested. Each test case should have a clear description of what it is testing and should include assertions that verify the expected behavior.
3. **Schedule the tests**: Use the `unified_ci` command-line tool to schedule the tests to be run. This will trigger the execution of the tests on the specified platforms.
4. **Monitor the tests**: You can monitor the progress of the tests on the Unified CI UI, where you can see the status of each test and any logs or error messages.
5. **Debug and iterate**: If there are issues with your tests, you can use the Unified CI UI to debug and iterate on your test cases.

For more information on using Unified CI testing, you can check out the official documentation at https://www.internalfb.com/intern/wiki/Unified_CI/.

Note: Unified CI is currently being deprecated in favor of Skycastle, which provides similar functionality but with a more modern and flexible architecture. If you're starting a new project, it's recommended to use Skycastle instead of Unified CI.

Breakpad

Breakpad is a library and tool suite that allows you to distribute an application to users with compiler-provided debugging information removed. Crash records are compacted into “minidump” files, send them back to your server, and then produce C/C++/Objective C stack traces from these minidumps.

To use Breakpad in your application, you will need to:

1. Create a Breakpad server and configure it to receive and process crash reports.
2. Integrate the Breakpad client library into your application.
3. Configure the client library to send crash reports to your Breakpad server.

Here are some resources to help you get started with Breakpad:

- Breakpad wiki
- Breakpad user guide
- Breakpad server setup guide

Choosing an E2E Testing Option

Based on your specific needs and requirements, if you are looking for a mature and widely-used E2E testing framework that supports multiple platforms, Jest-E2E may be the best choice. If you are working on RL devices and want a unified testing experience, you may want to consider using Unified CI once it is ready for production use. If you need detailed crash reporting and analysis capabilities, Breakpad may be a good addition to your testing toolkit.

End-2-end testing is explained in Testing and Debugging.

Logging

Logging is the act of keeping a record of events such as problems, errors, and other operations that occur when running a computer system. RLDI supports three logging systems: Logview, Log Insights, and RL Crash Task Controls, which are described below.

Logview

Logview is a powerful tool for consolidating and analyzing log data from various sources, such as web servers and mobile devices. Operating as a centralized platform, Logview enables seamless log management, rapid issue identification, and diagnosis for both developers and system administrators.

To use Logview, you can follow these steps:

1. Go to the Logview homepage by typing "logview" in your browser's address bar.
2. Select the Logview context that you want to view from the list of available contexts.
3. You will be taken to the Overview page for the selected context, which displays a list of MIDs (Message IDs) that have been logged.
4. Click on a MID to view more details about the error, including the stack trace and other metadata.
5. Use the filters and search bar at the top of the page to find specific errors or to narrow down the list of errors.
6. Click on the "Trends/Triage" tab to view graphs and other visualizations of the error data.
7. Click on the "Tasks" tab to view any tasks that have been associated with the error.

By using Logview, you can quickly identify and diagnose errors in your code, which can help you improve the reliability and performance of your application.

For information on Logview, refer to the Logview wiki.

Log Insights

The **Log Insights** tool allows you to quickly look at potential causes of an issue without downloading the actual log file. In most cases you can take a quick look, get to know the root cause and quickly take action. The goal of this tool is that not everyone has to remember all the relevant keywords to lookup in logs but can rather directly go to relevant sections for the type of issues you are debugging.

To use Log Insights, you will need to have access to the logs you want to analyze. This typically involves having the appropriate permissions to view the logs, as well as knowing where the logs are stored.

Once you have access to the logs, you can use Log Insights to search for specific keywords or patterns within the logs. You can also use it to filter the logs based on certain criteria, such as the date and time of the log entry, the source of the log, or the severity level of the log.

In addition to searching and filtering logs, Log Insights also provides some advanced features, such as the ability to create custom dashboards and alerts, which can help you monitor your logs more effectively.

Overall, Log Insights is a powerful tool that can help you make sense of your logs and troubleshoot issues more efficiently.

For information on Log Insights, refer to the Log Insights Tool wiki.

RL Crash Task Controls

RL Crash Task Controls allows management of RL CrashBot detectors through a web UI, without having to modify detector configuration code files.

To use Crash Task Controls, you will need to follow these steps:

1. Go to the Crash Task Controls dashboard at `bunnylol ctc`.
2. Click on the "Create New Rule" button in the top right corner of the page.
3. Fill out the form with the following information:

    - Rule name: a descriptive name for your rule
    - Description: a brief description of what your rule does
    - Owner: the person or team responsible for managing tasks created by this rule
    - Priority: the priority level for tasks created by this rule
    - Tags: any tags you want to add to tasks created by this rule
4. Under "Conditions", specify the criteria that will trigger task creation. You can choose from a variety of conditions, such as:
    - App ID: tasks will be created for crashes affecting a specific app
    - Logview Category Key: tasks will be created for crashes with a specific logview category key
    - Sad Users: tasks will be created if a certain number of users experience the crash
5. Click "Save" to create your rule.

Once your rule is created, Crash Task Controls will automatically create tasks for crashes that meet the specified criteria. The tasks will be assigned to the owner you specified and will have the priority and tags you specified.

You can view and manage your rules on the Crash Task Controls dashboard. From there, you can edit or delete existing rules, or create new ones.

For information on Crash Task Controls, refer to the Crash Task Controls wiki.

Choosing a Logging System

The choice of logging tool depends on your specific needs and use case. Here's a brief overview of each tool to help you decide:

1. **Logview**: Provides a centralized location for viewing, searching, and analyzing logs, making it easier to identify and diagnose issues. Logview is particularly useful for debugging errors and exceptions in production environments.
2. **Log Insights**: Provides a more detailed view of log data than Logview. It allows you to search and filter logs based on specific criteria, such as time range, log level, and message content. Log Insights also provides additional features like

log parsing and analysis, which can help you extract valuable information from your logs.

3. **RL Crash Task Controls**: Specifically designed for managing crash tasks in the Reality Labs (RL) organization. It provides a centralized location for tracking and managing crash tasks, allowing you to prioritize and assign tasks to team members. RL Crash Task Controls also integrates with other tools like Logview and Crashbot, making it easier to identify and resolve issues.

Based on your requirements, if you need a tool for general-purpose logging and debugging, Logview or Log Insights might be a better fit. If you're part of the Reality Labs organization and need a tool specifically designed for managing crash tasks, RL Crash Task Controls may be the best choice.

Landing Code

When it comes to landing code, your only choice within the RLDI AOSP Supported Workflow is **Picknic**, which is an internal system that allows developers to ask for revisions to be cherry picked into a product`s release.

To land a code with Picknic, you can follow these steps:

1. Make sure your code is ready to be landed:
    - Ensure that all tests have passed and there are no critical failures in the Signals section of your diff.
    - If there are any issues, address them before attempting to land your code.
2. Click on the "Ship It" button:
    - Once you've confirmed that your code is ready to be landed, click on the "Ship It" button in the top right corner of your diff page.
3. Address any issues:
    - If there are any issues with your diff, such as failing tests or lint errors, you will need to address them before you can land your code.
    - You may need to make changes to your code, update your test plan, or provide additional information to address the issues.
4. Start the landing process:

    - Once you've addressed any issues, click on the "Ship It" button again to start the landing process.
5. Wait for the landing process to complete:
    - The landing process may take some time, depending on the size of your diff and the number of other diffs that are also being landed at the same time.
    - You can monitor the progress of the landing process in the Signals section of your diff page.
6. Receive email notification:
    - Once your diff has successfully landed, you will receive an email notification.

It's important to note that landing a diff at Meta is a collaborative process, and it's important to work with your team and other stakeholders to ensure that your code is ready to be landed and that it meets the necessary standards for quality and reliability.

For information on landing code with Picknic, refer to the Release Workflow wiki.