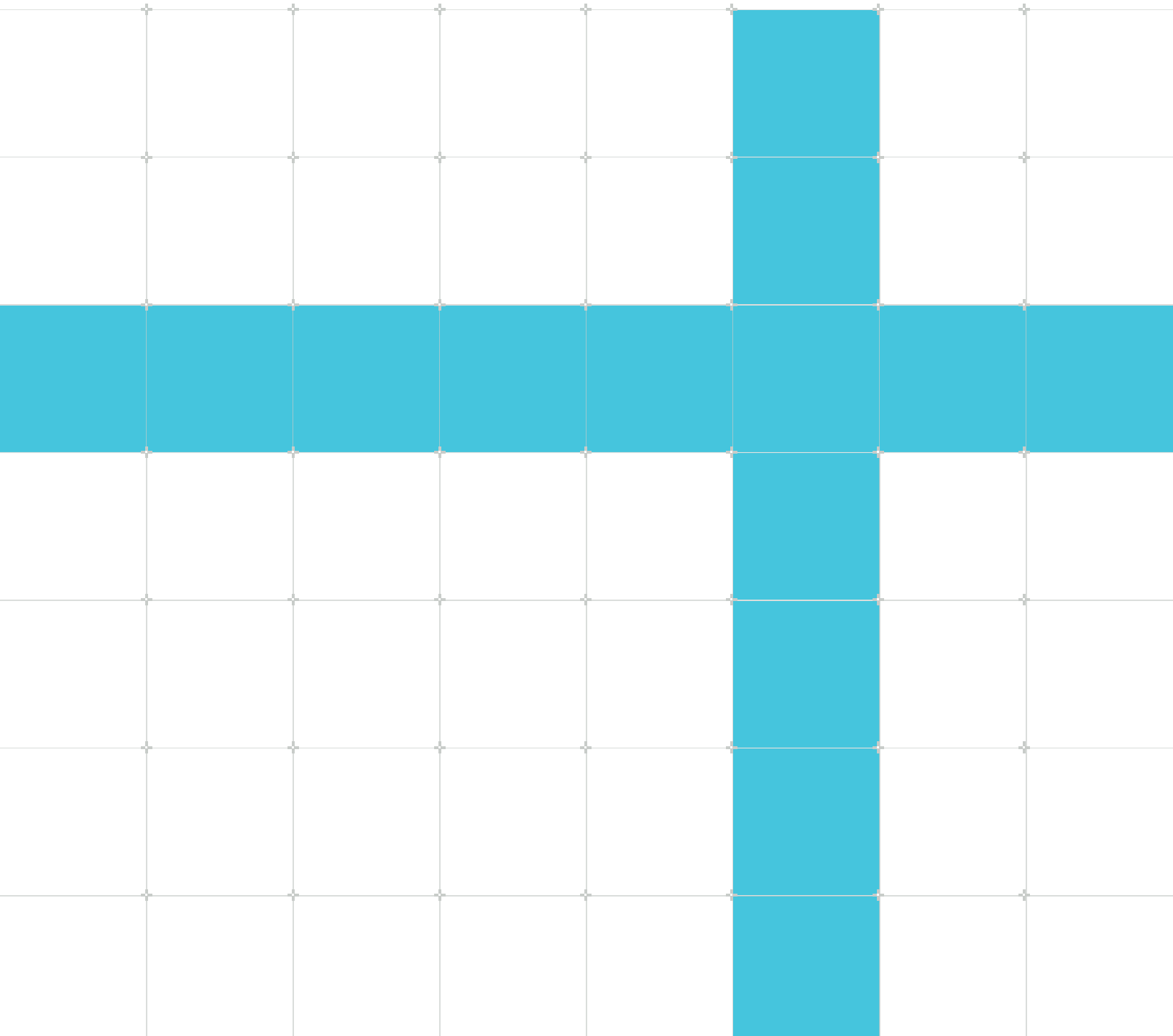
Integrate Arm Mobile Studio into a CI workflow

Version 1.0

Issue 01
102543_0100_01_en



Integrate Arm Mobile Studio into a CI workflow

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	25 November 2022	Non-Confidential	First release

Proprietary Notice

or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview 6

2. Configure your device farm 7

3. Export a configuration file 8

4. Configure CI commands 9

5. Helper script 11

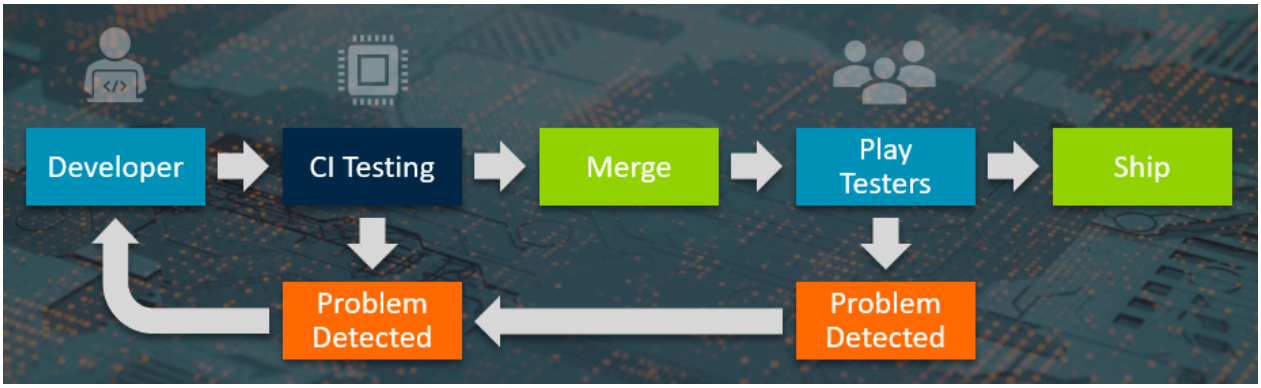
6. Create a dashboard 12

1. Overview

If your development team uses a CI (continuous integration) system to merge daily code changes, you can run nightly automated on-device performance testing across multiple devices, with Arm Mobile Studio. Automatically generate easy-to-read summary reports in HTML for your team to analyze each morning, and export machine-readable JSON reports, so you can build your own performance dashboards with any JSON-compatible database and visualization tools such as ELK stack.

1. Regularly pull your latest code and assets from your project repository and build debuggable APKs of the application test cases you want to run.
2. Add the Arm Mobile Studio capture and report generation commands to an appropriate stage of your CI pipeline, to export performance data in HTML and JSON format.
3. Push your JSON files to a database where you can collect and analyze the data over time.

Figure 1-1: CI testing



2. Configure your device farm

If you have access to a device farm for testing, ensure that you do the following:

1. Install Arm Mobile Studio on the host machine(s) that your devices connect to.
2. Each device must be in developer mode and have USB debugging enabled.
3. Each device must be connected to a host machine using USB.
4. If you are testing devices running Android 9 or earlier, you must include the Arm lightweight interceptor library (LWI) in your application. Refer to Prepare your application for detailed instructions on how to do this.
5. The testcase APKs you install must be debuggable, and ideally should be set to exit when the testcase completes. This simplifies the CI workflow, by removing the need to manually stop the application when the testcase finishes.
6. As a one-off setup task, you will need to generate a configuration file for each device, that defines which CPU and GPU activity counters Streamline should collect data from during the capture.

If you have a large number of devices, you may find it useful to categorize them by performance tier. The latest high-end smartphones will generally perform better than mass-market mid-range or low-end devices, and so you might want to set different performance targets for each. When you export data in JSON format, you can use the `targetInfo.device` field to select data from specific devices. Alternatively, you could push data from each device tier to a unique database index.

3. Export a configuration file

To generate performance data headlessly as part of your CI workflow, you need to generate a counter configuration file for each device, that defines which CPU and GPU activity counters you want to monitor during the capture. As a one-off setup task, you will need to create a configuration file for each device in your device farm. Streamline provides templates, that select an appropriate range counters for different GPUs, or you can build your own custom configuration.

1. Run the device connection script to connect Streamline to your device.
2. Click Counter configuration  and choose the counters you require, or select a template with  .
3. Click Export to save the configuration file to a location that your CI tool can access.
4. Repeat these steps for every device in your device farm.

You will need to provide this file to the CI command that you run to take a headless capture.

4. Configure CI commands

Here are the continuous integration workflow steps for performance analysis using Arm Mobile Studio. Use a CI tool such as Jenkins, TeamCity, or Buildbot to send the following instructions to the host machine(s) for each device in your device farm.

To simplify these steps, we've provided a helper script that generates the headless Streamline capture and handles starting and stopping of the application (steps 2 to 6).

1. Install your debuggable APK on each device, using Android Debug Bridge (ADB):

```
adb -s <device_serial_number> install <app.package.name>
```

For devices running Android 9 or earlier, your APK must include the Arm interceptor library. Refer to Prepare your application for instructions on how to do this.

2. Generate a headless Streamline capture and supply the configuration file you generated for this device:

```
python3 lwi_me.py --lwi-gles-layer-lib-path <path_to_layer_lib> --daemon
 <path_to_gatord> --package <app.package.name> --headless <capture_filename.apc>
 --headless-timeout <secs> --config <path_to_config_file>
```

   You can use `--overwrite` to overwrite an earlier headless output.

   For the full list of available command-line options refer to The lwi_me.py script options in the Performance Advisor user guide.

3. Add a wait period of at least 1 minute, to allow the script time to run.
4. Start the app on the device:

```
adb -s <device_serial_number> shell am start <app.package.name>
```

   If your app was built with Unity, you'll need to include the Unity player activity in `<app.package.name>`, for example: `com.arm.mygame/com.unity3d.player.UnityPlayerActivity`

5. Add an appropriate wait period to allow time for your testcase to run.
6. Stop the script and exit the app:

```
adb -s <device_serial_number> shell am force-stop <app.package.name>
```

7. Generate Performance Advisor reports in HTML and JSON formats.

```
pa <capture_filename.apc> -p <app.package.name> -d <output_directory> -t
 html:<file_name.html>,json:<file_name>.json
```

   For the full list of available command-line options refer to The pa command in the Performance Advisor user guide.

8. Push the HTML reports to a centrally visible location for your team to analyze each day.
9. Push the JSON files to your chosen database and visualization tool such as ELK stack.

```
curl -X POST "<Elasticsearch_location>/indexname/_bulk?pretty" -H 'Content-Type:
 application/x-ndjson' --data-binary @<file_name>.json
```


Note

If you are planning to export your JSON files to Elasticsearch, and you're using Arm Mobile Studio version 2020.1 or earlier, you need to convert the file to the expected format first. Refer to our FAQ for instructions on how to do this.

5. Helper script

To help simplify your CI steps, we've created an example Python script that you can modify and run as part of your CI workflow, to capture performance data automatically from an Android device, with Arm Mobile Studio. This script generates the headless Streamline capture and handles starting and stopping of the application (steps 2 to 6 under Configure CI commands).

Download the helper script

Modify this script to set the following values:

1. Set `DEVICE_NAME` to the ID of the current device. You can discover this using the `adb devices` command in a terminal.
2. Modify the command-line options for `GATORME_COMMAND` to:
   a. Specify the name of the testcase application with `--package`
   b. Specify the name and location of the resulting capture file with `--headless`
   c. The script assumes that `lwi_me.py, gator_me.py, gatord, libGLES_layer_lwi.so` and the configuration file for this device, `configuration.xml` are in the current working directory. If they are not, modify the filenames to add the full paths.
   d. To use the correct Python command on macOS or Linux, comment out "py" and "-3" and uncomment "python3":

```
# "py", "-3",
 "python3",
```

   e. Add any other command-line options you require for this run, such as the build name, timestamp or any frame capture options. For the full list of available command-line options refer to The lwi_me.py script options in the Performance Advisor user guide.
3. Modify the ADB command-line options for `APPLICATION_ADB_COMMAND`, to specify the application to run.
4. Set `APPLICATION_EXIT_PATTERN` to the logcat pattern emitted by the application when it finishes running the testcase. This is used to stop the application. Alternatively, if you've configured your application to exit at the end of the scenario, set this to `None`.

6. Create a dashboard

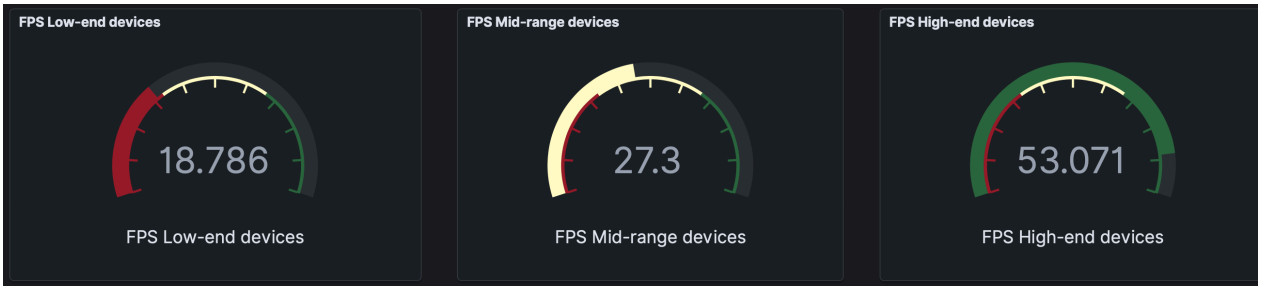
Decide which metrics your team needs to monitor over time. If you have organised the devices in your device farm into performance tiers, you can set performance targets and build separate charts to show whether devices in each tier are meeting those targets.

We've used Elasticsearch and Kibana to store and visualize JSON data exported from Arm Mobile Studio. Here are some example charts you could build to monitor performance.

Current average FPS

This chart shows a snapshot of the average FPS broken down by device performance tier.

Figure 6-1: FPS diagram.

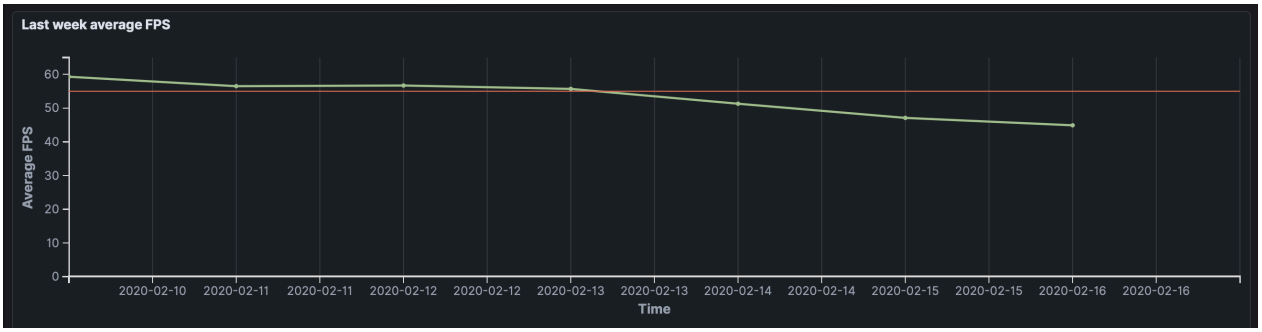


1. Build an average FPS chart for each device tier using the `allCapture.averageFrameRateFps` field.
2. Filter each chart to show devices that belong to each tier. If you've stored data for each device in separate indices, you can filter by index. Alternatively, use the `targetInfo.device` field to choose device names.
3. Adjust the colors in each chart to reflect your target FPS.
4. Filter the chart to the date range you require. You could show today's date, the last 7 days, or any date range that makes sense to your team.

Average FPS over time

This chart shows how the average FPS across all devices changes over time.

Figure 6-2: Average FPS diagram



Build a chart for each device tier. Plot the `allCapture.averageFrameRateFps` field for the devices in each tier over time. Select the required devices from different indices containing your tiers, or with the `targetInfo.device` field.

GPU budgeting

If you know the top frequency achievable by the GPU in your device, and you have a target frame rate, you can calculate a maximum GPU cycles per frame budget, and measure your content against it. If your content breaks this budget, it might cause frame rate to drop. Plot the `gpuCycles.max` field to monitor this value over time. You could also set a query to alert you when your budget is broken.

Region analysis

If you've used annotations or a regions file to divide your testcase into different sections, you can monitor data for each section in different charts.