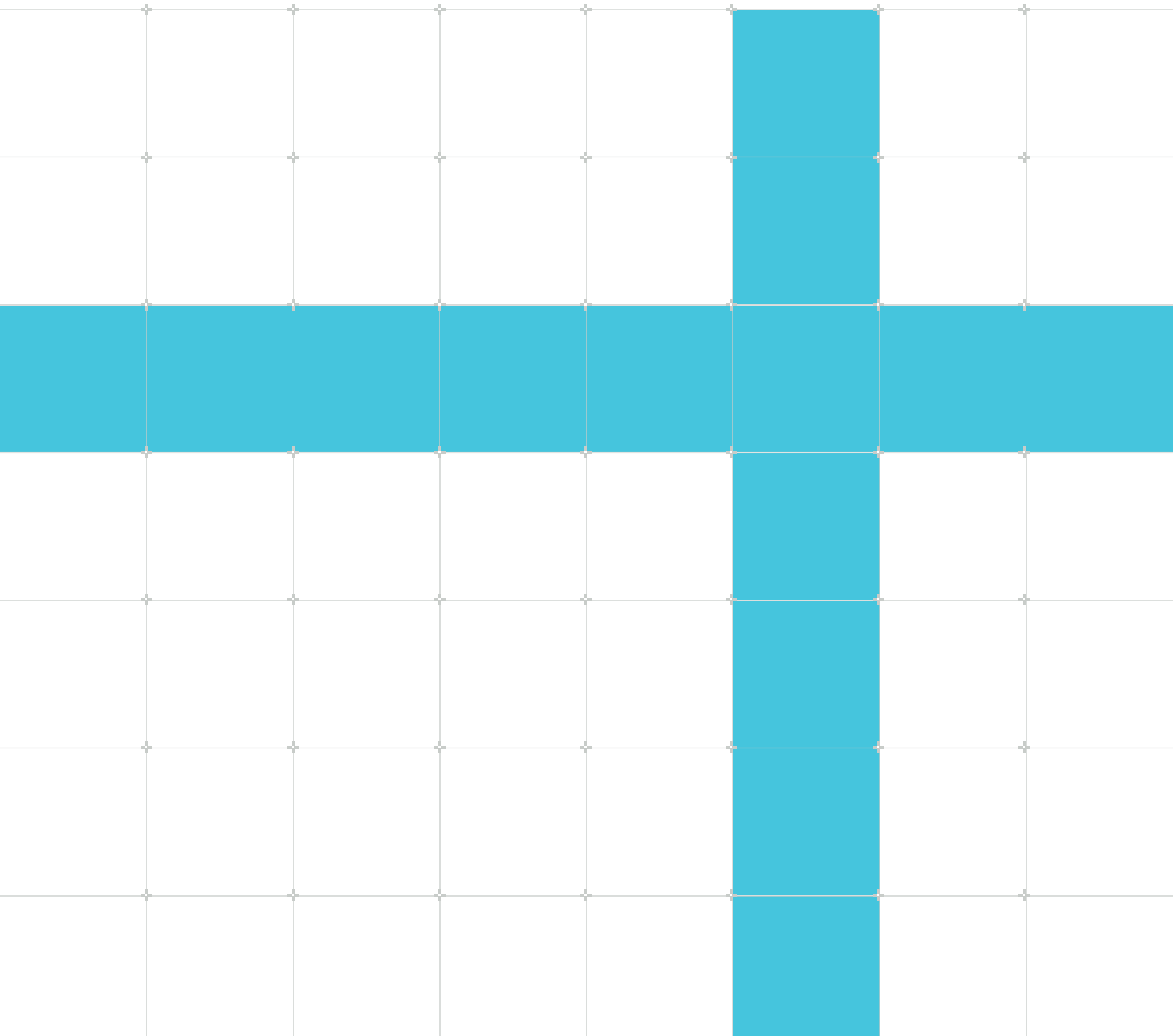
Integrate Arm Performance Studio into a CI workflow

Version 1.3

Issue 00
102543_0103_00_en



Integrate Arm Performance Studio into a CI workflow

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	25 November 2022	Non-Confidential	First release
0101-00	21 April 2023	Non-Confidential	Update to Performance Advisor report-generation command.
0102-00	18 July 2023	Non-Confidential	Update to script name and command-line options.
0103-00	23 February 2024	Non-Confidential	Arm Mobile Studio renamed to Arm Performance Studio. Updated screenshot and minor fixes.

Proprietary Notice

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.

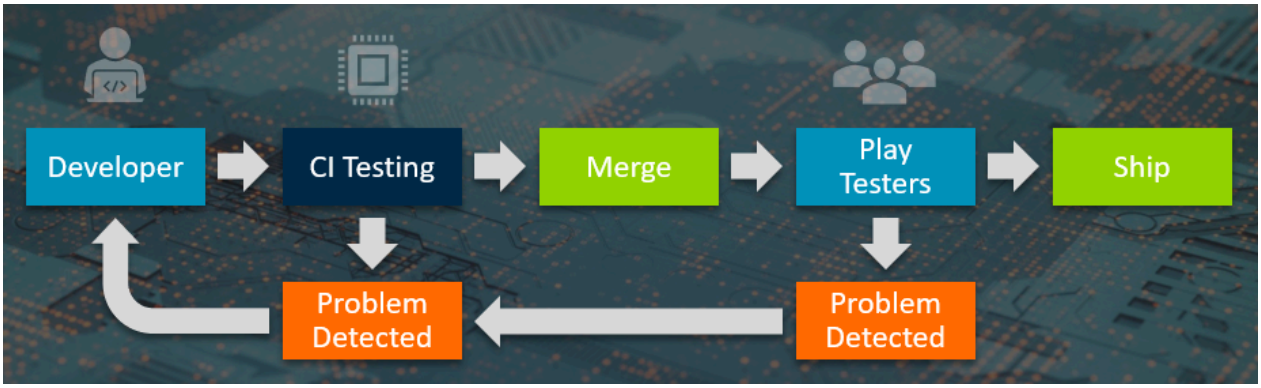
Contents

1. Overview

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 Streamline and Performance Advisor available as part of Arm Performance Studio for Mobile. Automatically generate 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 tool 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 Streamline 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



Note

If you are unfamiliar with Streamline and Performance Advisor, work through the Get started with Performance Advisor tutorial to learn how to take captures.

Next steps

Go to Configure your device farm to learn how to prepare the devices you want to test.

2. Configure your device farm

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

1. Install Arm Performance Studio on the host machine(s) that your devices connect to.
2. Complete the setup tasks as described in the Get Started with Performance Advisor tutorial.
3. If you are testing devices running Android 9 or earlier, you must include the Arm lightweight interceptor library (LWI) in your application.
4. 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.
5. As a one-off setup task, you will need to Export a configuration file for each device. This file 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.

Next steps

Export a configuration file for each device. This file defines which CPU and GPU activity counters Streamline should collect data from during the capture.

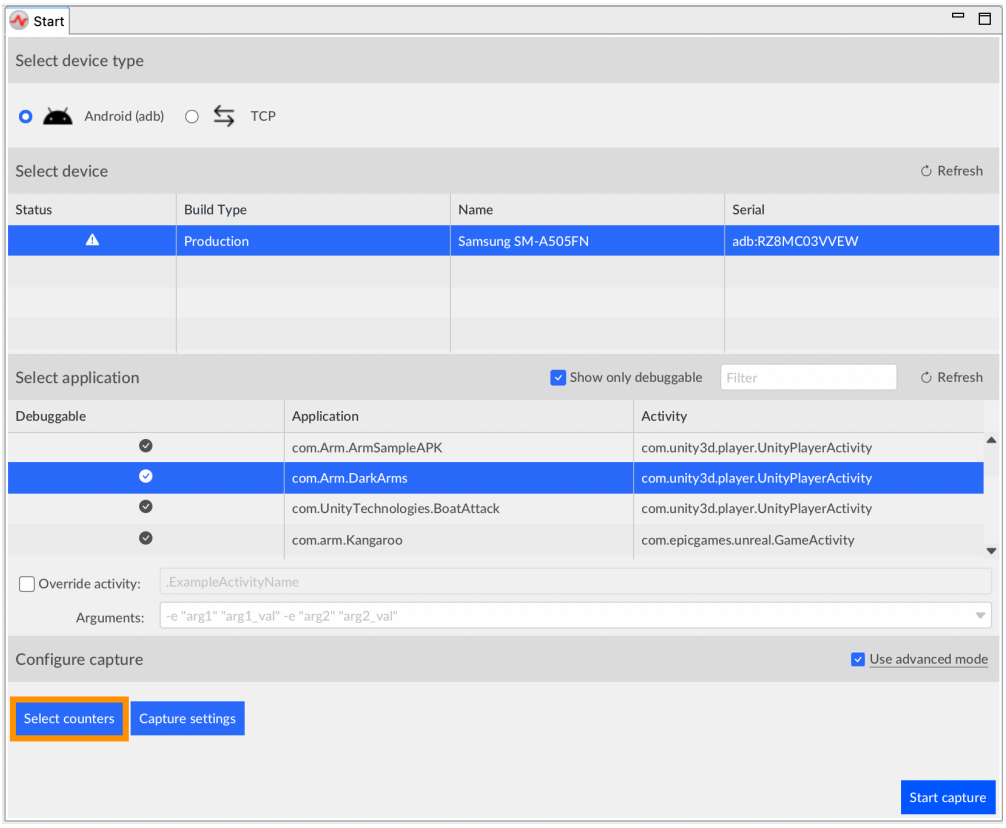
3. Export a configuration file

To generate performance data headlessly from Streamline 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.

Procedure

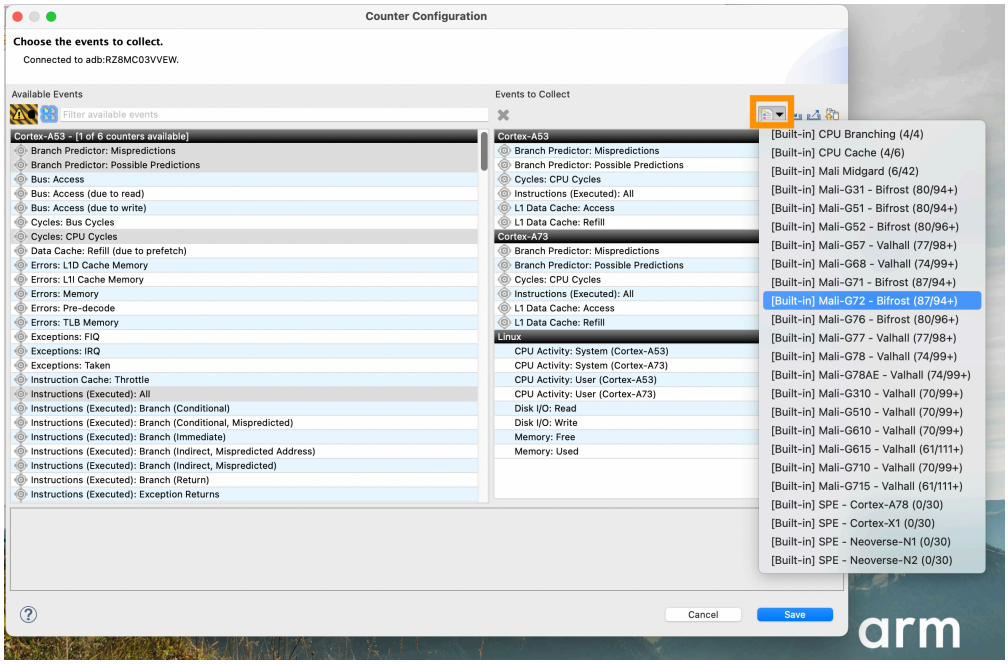
1. Open Streamline and ensure the device is connected via USB. You should see your device in the Start tab.
2. Select your device and the application you want to profile.
3. Enable the Use advanced mode option and choose Select counters.

Figure 3-1: Streamline Start Tab in advanced mode



4. Select a template for the GPU in the target device.

Figure 3-2: Adding counters from a template in Streamline



Alternatively, you can create a custom configuration by adding the required counters to the Events to Collect list.

5. Click Export to save the configuration file to a location that your CI tool can access.
6. Repeat these steps for every device in your device farm.

You will need to provide this configuration file in the CI command that runs the set up script `streamline_me.py` described in the Configure CI commands section.

Next steps

Once you have generated configuration files for all the devices in your device farm, you can Configure CI commands in your CI tool.

4. Configure CI commands

Here are the continuous integration workflow steps for performance analysis using Streamline. 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.

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

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

2. Change to the `<install_directory>/streamline/bin/android directory`, or copy the following files from that location to your working directory:

```
streamline_me.py
<arm|arm64>/gatord
<arm|arm64>/libGLESLayerLWI.so
<arm|arm64>/libVkLayerLWI.so
```

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

```
python3 streamline_me.py --lwi-mode counters --daemon <path_to_gatord> --package
 <app.package.name> --headless <capture_filename.apc> --headless-timeout <secs>
 --config <path_to_configuration.xml>
```

   Use the `--overwrite` option to overwrite an earlier headless output.

   For Vulkan applications, you also need to include the `lwi-api=vulkan` option.

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

4. Add a wait period of at least 1 minute, to allow enough time for the script to run.
5. 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 will need to include the Unity player activity in `<app.package.name>`, for example: `com.arm.mygame/com.unity3d.player.UnityPlayerActivity`

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

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

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

```
Streamline-cli -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 Streamline-cli -pa command in the Performance Advisor user guide.


Note

This command changed in Arm Mobile Studio 2023.1. If you are using a previous version, the command name was `pa`. For the command-line details, refer to the Performance Advisor user guide for your version of Arm Mobile Studio.

9. Push the HTML reports to a centrally visible location for your team to analyze each day.
10. 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
```

Next steps

Once your CI system is collecting data regularly, you can set up a dashboard in any database and visualisation tool. Learn how to Create a dashboard using ELK stack.

5. Create a dashboard

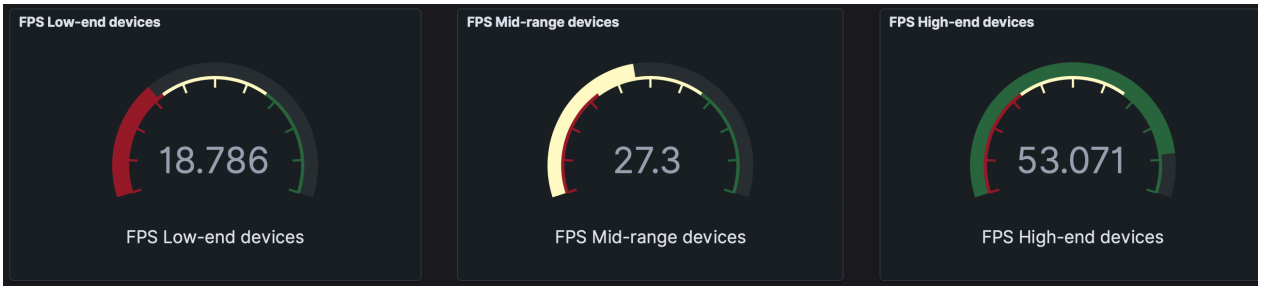
Decide which metrics your team needs to monitor over time. If you have organized the devices in your device farm into performance tiers, you can set performance targets and build separate charts to show whether devices in each tier are meeting those targets.

We have used Elasticsearch and Kibana to store and visualize JSON data exported from Arm Performance 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 5-1: FPS diagram

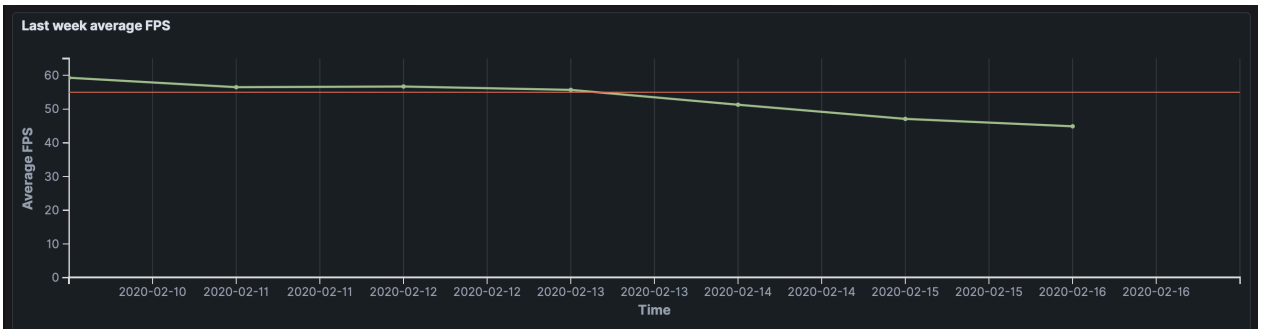


1. Build an average FPS chart for each device tier using the `allCapture.averageFrameRateFps` field.
2. Filter each chart to show devices that belong to each tier. If you've stored data for each device in separate indices, you can filter by index. Alternatively, use the `targetInfo.device` field to choose 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 5-2: 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.