10

Monitoring and optimizing

the application

Application monitoring is a simple process of keeping track of various aspects of an application and how they are performing. It is very important for consistent quality check and improvement and important to find out problems in the application before it gets to users.  
Application monitoring will not only let us know the performance of the application and issues within the application but also keep in check the status of its related databases and APIs.

In this chapter, we’ll be discussing about various ways of application monitoring. High level topics that we’ll be covering in this chapter:

* API Level Monitoring and Various Tools for API Monitoring
* Monitoring Application with Test Cloud
* Monitoring Application using Android Monitoring Tools

**API Level Monitoring**APIs (Application Programming Interface), are an integral part of today’s integrated development environments. It can also be understood with client-server relationship. Where Mobile app is the client requesting for a resource and API is at the server side that has a URL for any application that wants to make a request.  
Most of the applications share common APIs between web and mobile. APIs are a great way of providing a consistent operation behavior across different platforms. It also helps share the same business and data layer operations between different mobile application platforms like you can use the same APIs for iOS, Android and web applications.

Because API is so important in the development of a mobile application, it becomes equally critical to monitor APIs to ensure high availability. API goes down, entire application can stop working and user might not be able to do any operation that requires API to be available, that usually is any server operation and not offline.

**Why API Monitoring is Critical**

When we talk about continuous integration, it should include a continuous cycle of development, testing, deployment and monitoring. As engineering teams like using Agile practices and other techniques for software development, testing and deployment, it is critical that monitoring becomes a must have part of that continuous integration cycle and we update our monitoring methods and tools with the correct and latest versions to ensure high availability for our users and other consumers like mobile and web applications that might be using our APIs.

As we discussed earlier in this chapter, when APIs fail, your applications fail. If you are using APIs to provide services for your mobile app and they are either not available, functioning improperly or not being responsive, it impacts the performance of your app directly and the end user experience gets compromised.

Also, it is very important to monitor APIs not only that you are developing but also, third party ones that you might be using in the application.

**Important factors in API monitoring**

When monitoring API there are some key points or areas that need to be covered to ensure better availability.

* **API Availability**  
  We need to make sure if the API is available or not, sometimes the server might be down for some reason or the connection can be interrupted based on location and server.
* **Quality of Response**  
  When called an API, how is the quality of response returned from the API. Is it according to the agreement or not.
* **Response Time**   
  What is the response time to get a result when called the API.

**Developer’s Role in handling API unavailability**

It is also a good idea for an application developer to keep in mind that APIs might not be available some time and write code in a way that handles these kinds of situations gracefully. Even when application is not able to perform some API operations, if it is informed to the user in a nice way, it can help the user experience much more than facing runtime exceptions. Writing code to handle API exceptions and to handle scenarios where API does not respond.

**Various Tools for API Monitoring**

There are many tools available in the market for API monitoring and testing. What to choose totally depends on what you want to achieve through those tools. Some tools provide great support for performance monitoring and other tools are better suited for quality testing and recognizing erroneous data.

**Postman**

Postman is a rest client that started off as a Chrome browser plugin but recently came out with native versions for both Mac and Windows.

At a high level, you can use it to send a post request to your web server and it gives you the response back. It allows you to set up all the headers and cookies your API expects, and then check the response when it comes back.

* Can be used for both automated and exploratory testing
* Can be run on Mac, Windows, Linux &Chrome Apps
* Has a bunch of integrations like support for Swagger & RAML formats
* Has Run, Test, Document and Monitoring Features
* Doesn’t require learning a new language

**Karate DSL**

Karate allows you to create a test that can sequence calls to any kind of web-service and assert that the responses are as expected.

* Build on top of Cucumber-JVM
* Can run test and generate reports like any standard Java project
* Test can be written without any Java knowledge required
* Tests are easy to write even for non-programmers
* Check out a quick example on how to get started using Karate with BDD.

**SoapUI**

SoapUI is a headless functional testing tool from SmartBear software. It comes in two flavors: Free open source version and Pro Version. Since the free version is open-source, you can actually gain access to the full source code and modify as needed. The pro version is user-friendlier, and has additional functionality including a form editor, an assertion wizard for Xpath, and SQL query builder. The free version lets you:

* Can easily create custom code using Groovy
* Drag and Drop Test Creating
* Can create complex scenarios
* Asynchronous Testing
* SoapUI’s Mock Service lets you mimic web services before they are implemented

**HttpMaster Express**

HttpMaster describes itself as a web development and test tool to automate testing of web sites and services. It can be used to test RESTful web services and API applications. HttpMaster also allows you to and monitor API responses.

* HttpsMaster project offers global options to customize your API request
* Parameter capabilities enable you to include dynamic data with your request
* You can use request chaining to leverage request items to include some data from previous request with the next request

**Rest- Assured**

Rest-Assured is an open-source Java Domain-specific language (DSL) that makes testing REST service simple. It simplifies things by eliminating the need to use boiler-plate code to test and validate complex responses. It also supports XML and JSON Request/Responses.

* Removes need to create boilerplate code required to interact with a rest service
* Support BDD Given/When/Then syntax
* Integrated seamlessly with Java projects

**RestSharp**

RestSharp is a simple REST and HTTP API Client for .NET

* Supports .NET 3.5+, Silverlight 5, Windows Phone 8, Mono, MonoTouch, Mono for Android
* Easy installation using NuGet for most .NET flavors
* GET, POST, PUT, PATCH, HEAD, OPTIONS, DELETE supported

**Rest Console Console**

HTTP Client and Request Visualizer and Constructor tool, helps developers build, debug and test RESTful APIs. Rest Console is a HTTP Request Visualizer and Constructor tool, helps developers build, debug and test RESTful APIs.

* Easy query parameters creation
* Syntax highlighting
* Authentication support: Plain, Basic, OAuth + Custom

**Hippie-Swagger**

hippie-swagger is a tool for testing RESTful APIs. It’s also an API testing tool with automatic swagger assertions. In addition to validating API behavior, it will fail tests when swagger documentation is missing or inaccurate.

* Can validate All aspects of swagger file validated; parameters, request/response body, paths, etc.
* Accurate, human readable assertion messages

**Airborne**

Airborne is an open source Ruby based RSpec driven API testing framework.

* Works with Rack application like Sinatra and Grape
* Works with APIs written in Rails

**Mockbin**

Mockbin was recommended by Augusto Marietti. Mockbin allows you to generate custom endpoints to test, mock, and track HTTP requests & responses between libraries, sockets and APIs.

* Mock Custom Endpoints
* Create Custom HTTP Methods
* Log and inspect incoming calls to your custom endpoints

**Using Test Cloud for Monitoring**

You learned about Xamarin Test Cloud in previous chapters and how to use it for continuous testing in the continuous integration life cycle. Here we will discuss more about how to use Xamarin Test Cloud and the analytics it provides after running application on different set of devices.

We will be using 2 different applications here to see the monitoring analytics and compare them to get a better understanding of how this helps us identify various performance and functionality related issues in our application.

Applications we will be using for the walkthrough:

1. PhoneCallApp (The application we developed in the previous chapters)
2. Xamarin Store (Sample Android application provided by Test Cloud)

Xamarin Test Cloud can help us identify application functionality related issues on real devices.  
It is a great source of application monitoring in terms of testing on different mobile devices and with different versions of operating systems.

Getting a detailed analysis of various applications functions is very important to make sure our application is running as expected on our target devices.  
With that being said, it is also critical to the application to be able to run on different operating system versions and on to analyze how it performs and how much memory usage does it have.

**Benefits of Monitoring with Test Cloud**

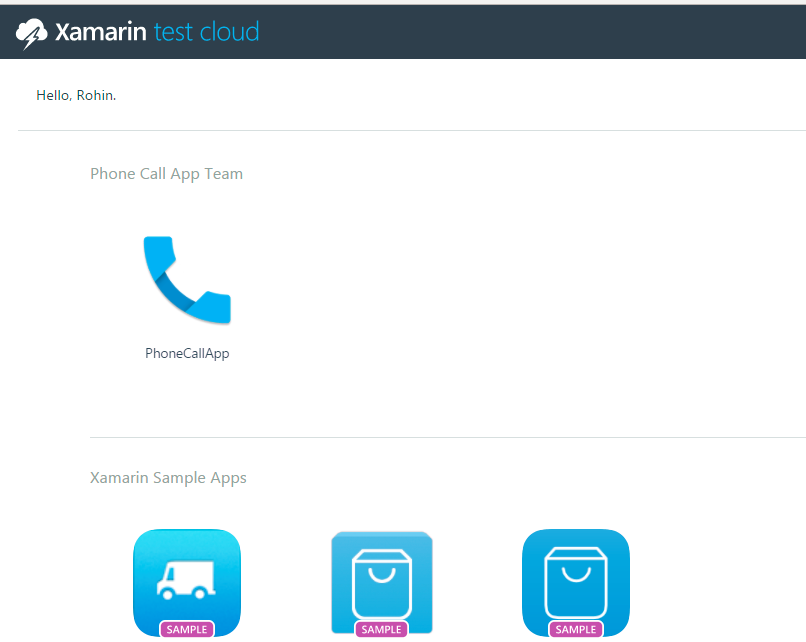
With Test cloud it not only provides monitoring capabilities but also relieves us from testing the same application functionality on different devices manually and thus giving a true continuous integration process.

* Provides continuous testing capabilities to our CI process with automated test runs and detailed reports with notifications.
* Testing application on different OS versions is critical to the success of a mobile application and Test Cloud serves that purpose very well.
* Testing application on different devices from its huge device list available on cloud
* Analyzing performance of the application
* Analyzing memory usage on different devices with different hardware configurations

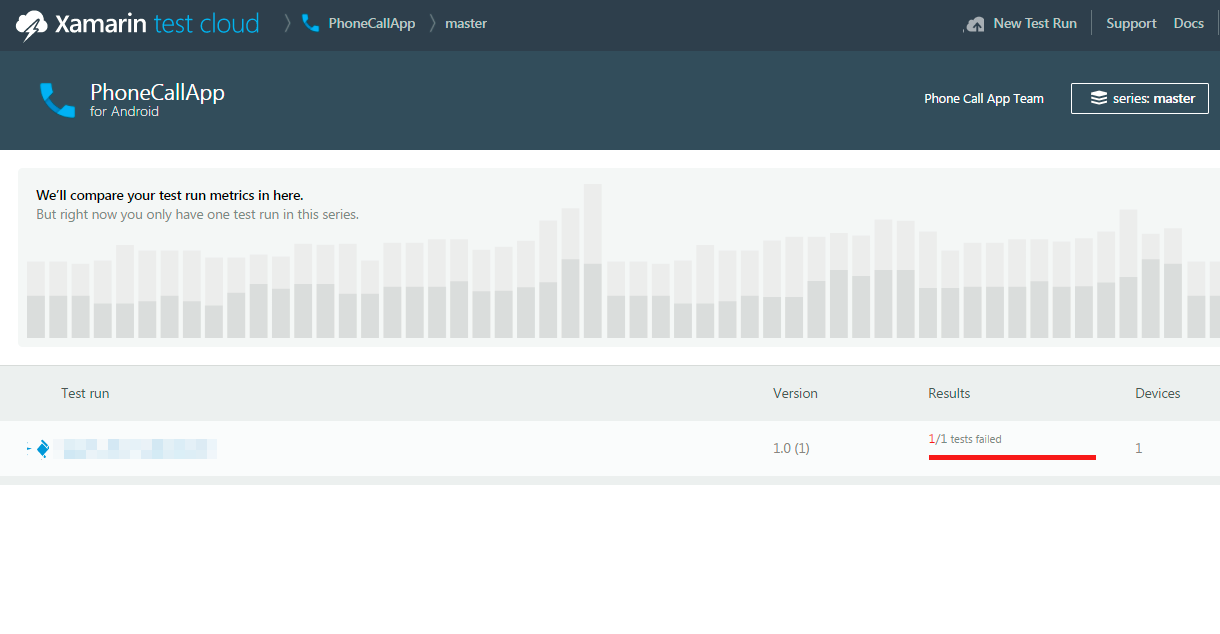
**PhoneCallApp**

Let’s go through some steps to see how to monitor our PhoneCallApp.

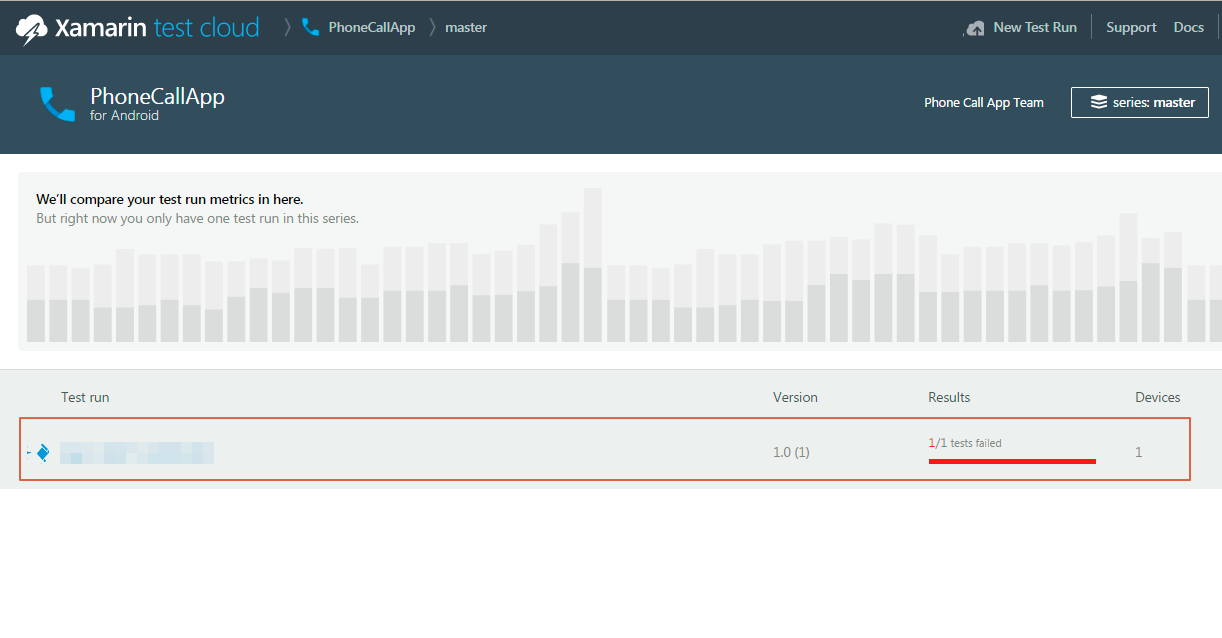
1. Go to <https://testcloud.xamarin.com/>
2. Click on the PhoneCallApp icon to get to the details of “Test Runs”.



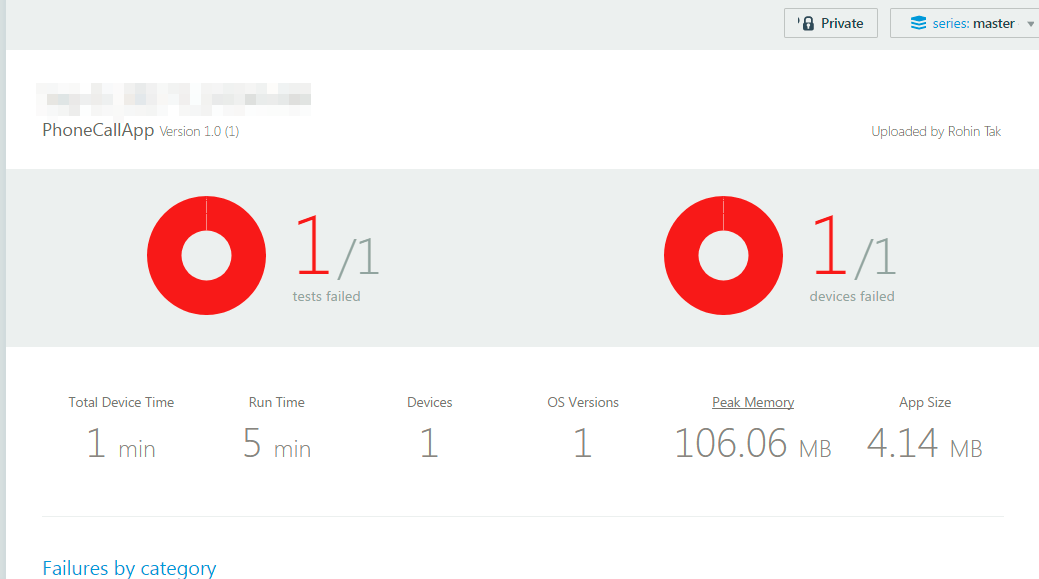
1. On the next page you’ll see a list of tests run for the application.



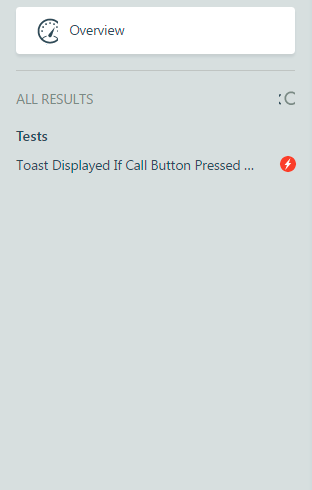
1. Now because we have only one test run as of now, Test Cloud does not provide us with the graphical metrices above. In other examples we see next, you’ll be able to see more detailed comparison of different test runs.
2. Click on the test run from the list to see it’s results.



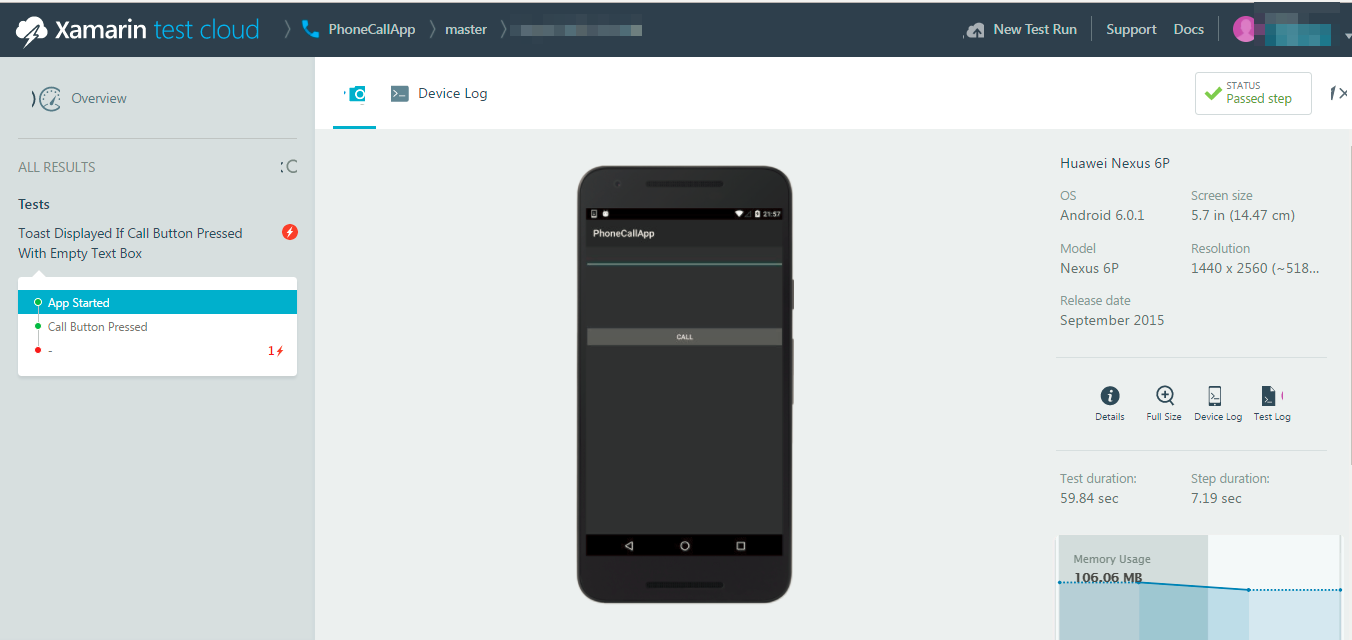
1. The test run listed is the one we ran earlier in previous chapters and uploaded from our machine to Xamarin Test Cloud using the command line.
2. To get an idea of this interface, let’s have a look at different part of Xamarin Test Cloud’s interface.
3. Now this is an overview screen that shows a summary of all the tests run for this application.



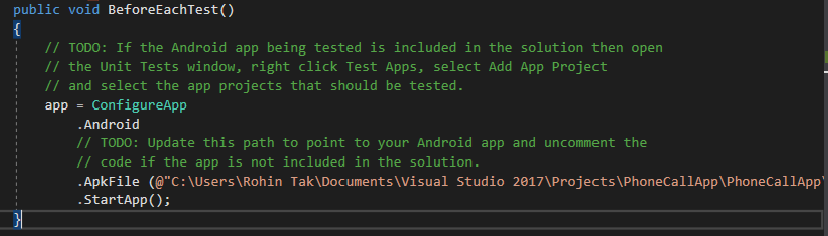
1. This screen shows summary details like, how many tests were failed from total tests run, How much time the app run on a devices, how many devices these tests were run on etc.
2. This screen is very useful to get a brief idea when you want to get a report on how your application is doing on different devices and OS versions.
3. Next thing you’ll see if at the left pane, that is the list of UI tests included in the test run.



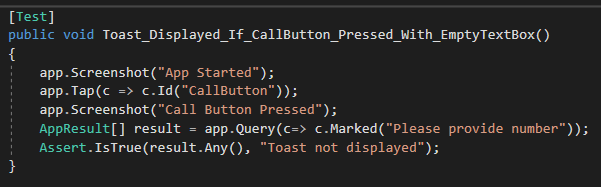
1. This screen basically has the list of all your Xamarin.UITests that you include in your project. You can click on these different tests to see their respective results on the right pane of the screen.
2. Let’s click on the test from the list in above screen.
3. This will take us to the next screen that has detailed reports of the test run.



1. Have a close look at the left pane on this screen.
2. It gives us some steps of the test run on the device.
3. These steps are nothing but what we had written previously in the code to take screen shots of every activity the test does.
4. The Steps are as mentioned below (We are using the screens of the test code written in previous chapters here).
   1. **App Started:** Take a screenshot when the app start happens, this was written in the BeforeEachTest() method in the Tests.cs file.

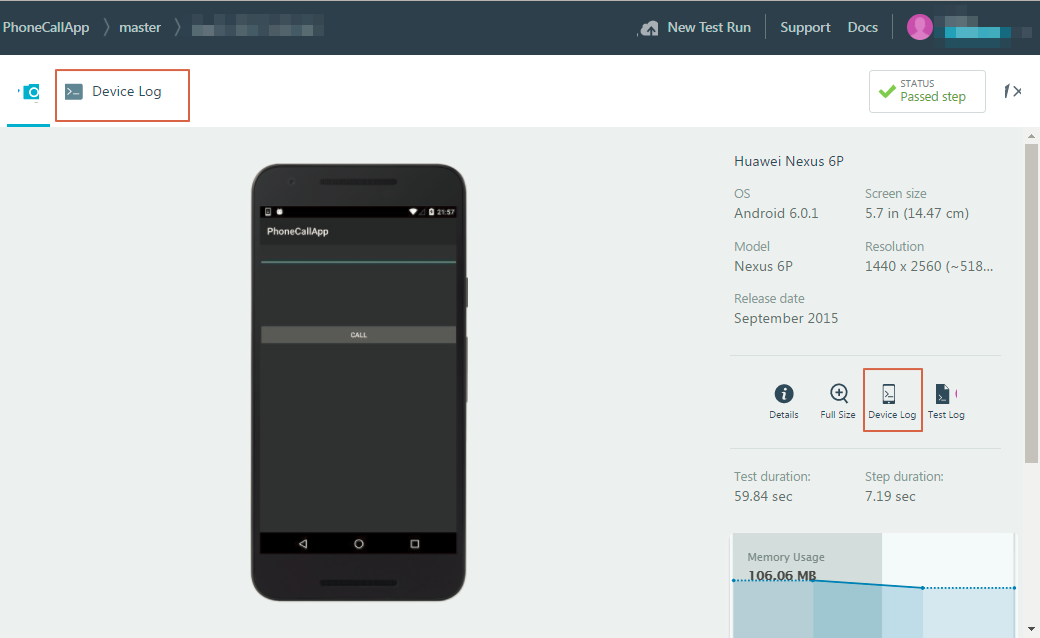


* 1. **Call Button Pressed:** This step is when the Xamarin.UITest presses the Call Button to make a call.

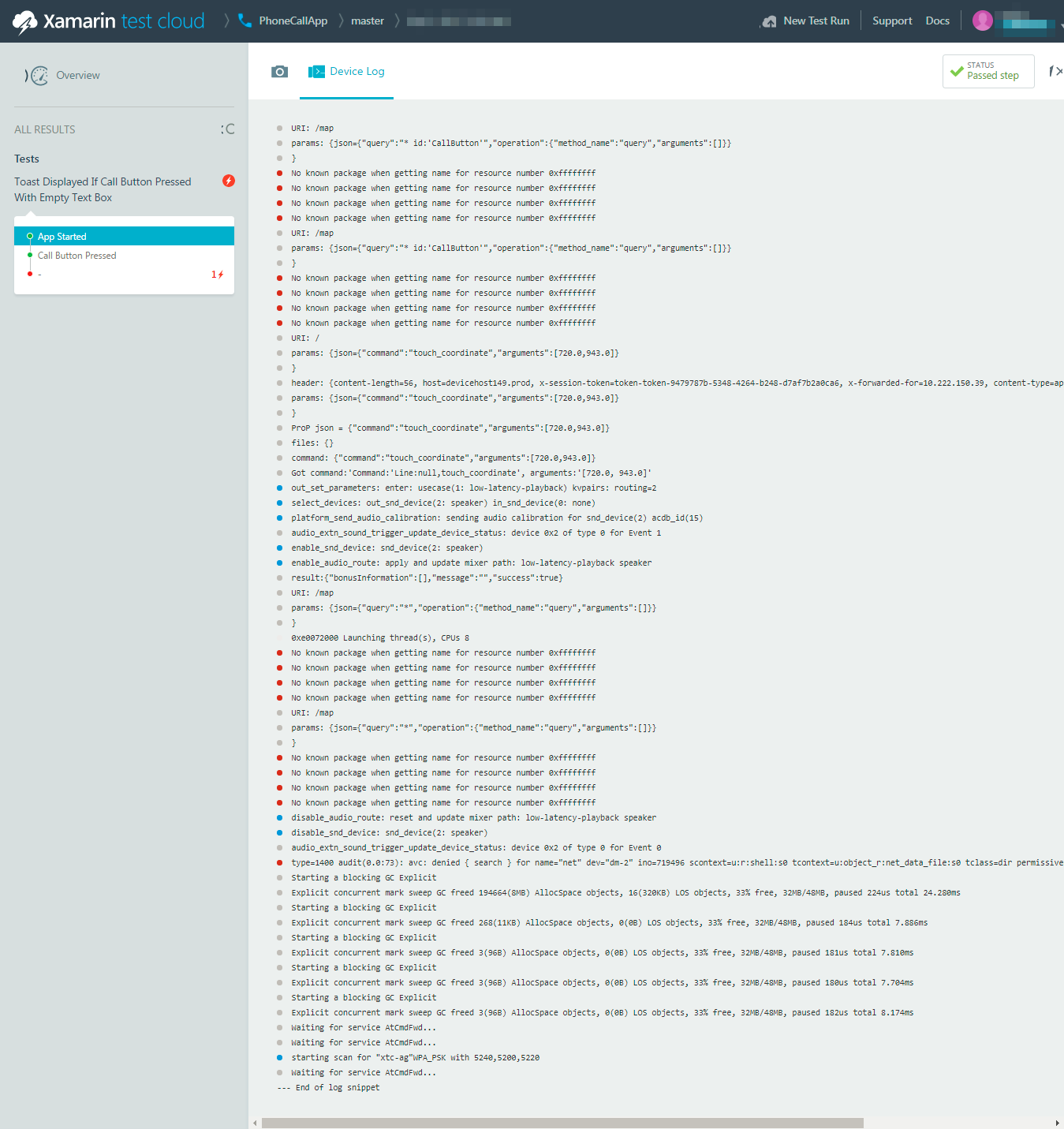


* 1. **Failed Step (The Assert):** This is the last step and is shown to have a proof of the failed step, so you can see the outcome that we had received and compare with what was expected. This was the final assert that decides if the test passes or not based on the outcome in the Assert.IsTrue() condition.

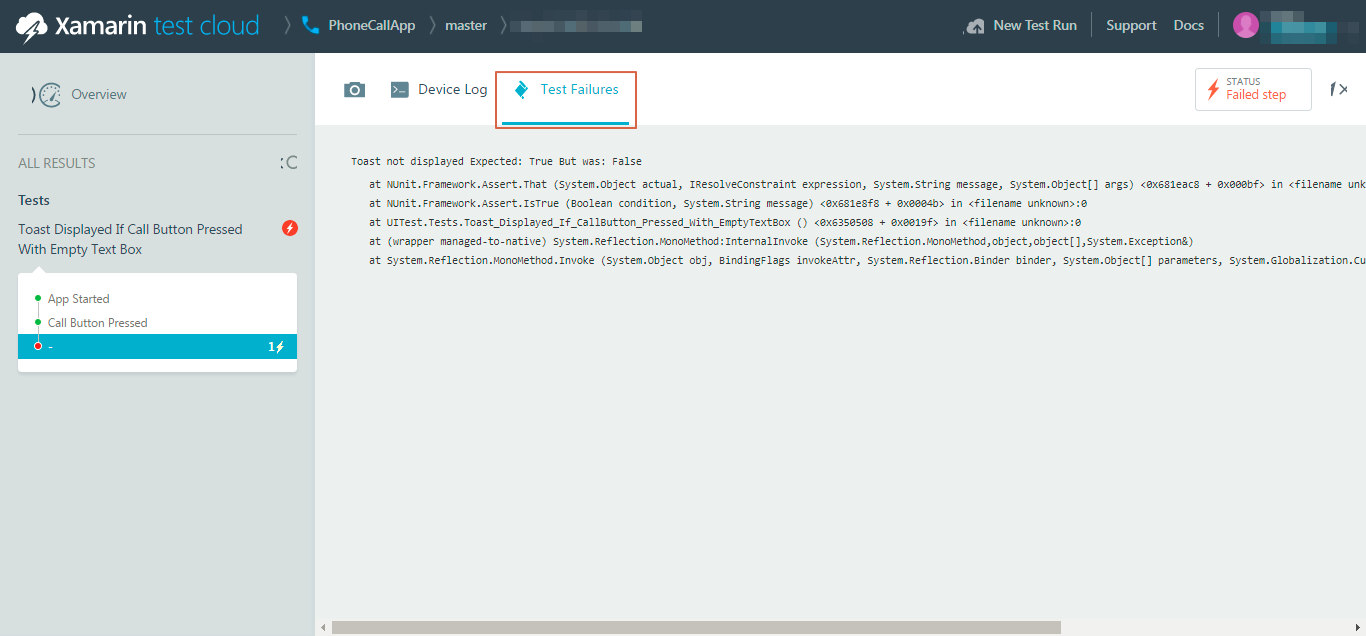
1. You can click on each of these steps on the left pane and analyze the screenshots taken to see exactly what went during the test. This is a great way to see exactly what went wrong when the test failed.
2. Now, sometimes, the screenshots are not enough to identify the issue, for more detailed analysis, Test Cloud also provides us with Device Logs.



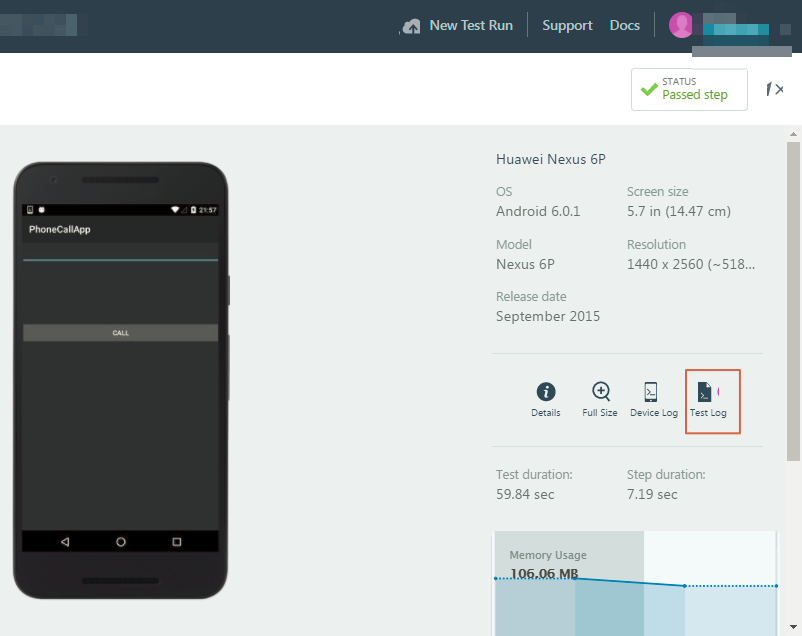
1. Device logs are great way to see what’s going on under the hood and see a more detailed information about the application’s behavior and how the device itself behaves when the application is run on it.
2. This can help pin point the issues when a test fails on the device, logs are always a savior in that sort of scenarios.
3. Click on the device log and you can see step by step logs for each screenshot on the same screen.



1. When a test fails, Test Cloud provides us with one more option to see the Test Failures.



1. It’s very useful for automated test developers to see the exception information when a test fails.
2. Last but not least, there is also a Test Log option, that can be used to get a consolidated log of the entire test run.

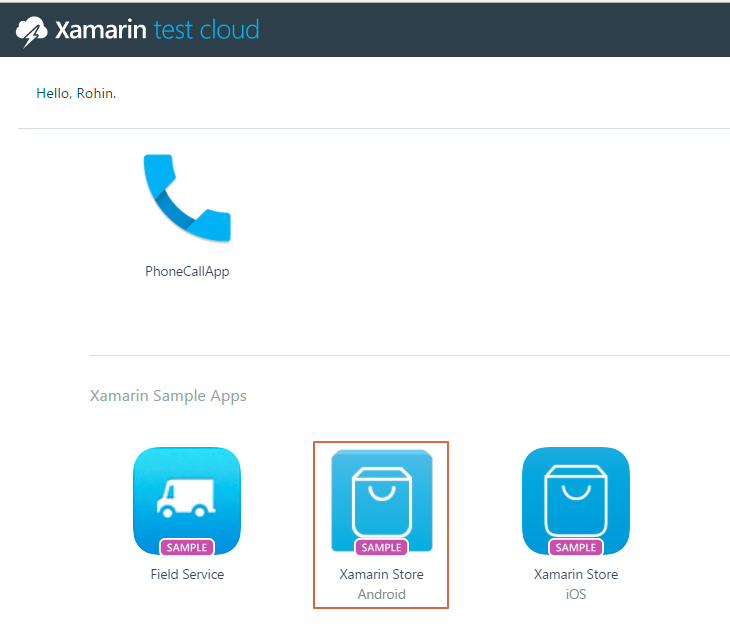


**Xamarin Store App**

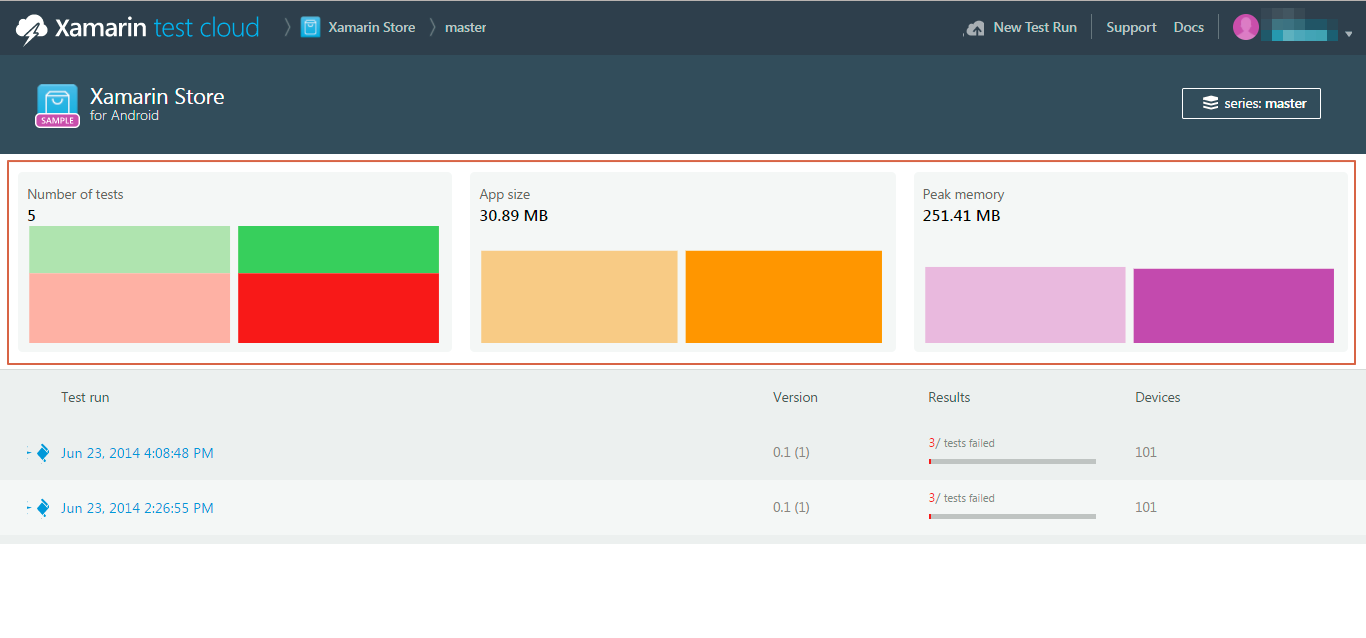
Now that we have seen different options provided by Test Cloud to monitor our application and its functionality using test runs, let’s see how does the dashboard and tests look when we have multiple test runs on various physical devices with different OS versions.  
This will give us a better idea of how a comparative monitoring can be done on Test Cloud to analyze application’s behavior on different devices and compare them with one another.

Xamarin Store application is a sample application provided by Test Cloud on its platform to help understand the platform and get an idea of the dashboard. Let’s go through the steps to understand how to monitor your application running on multiple devices and how to compare different test runs.

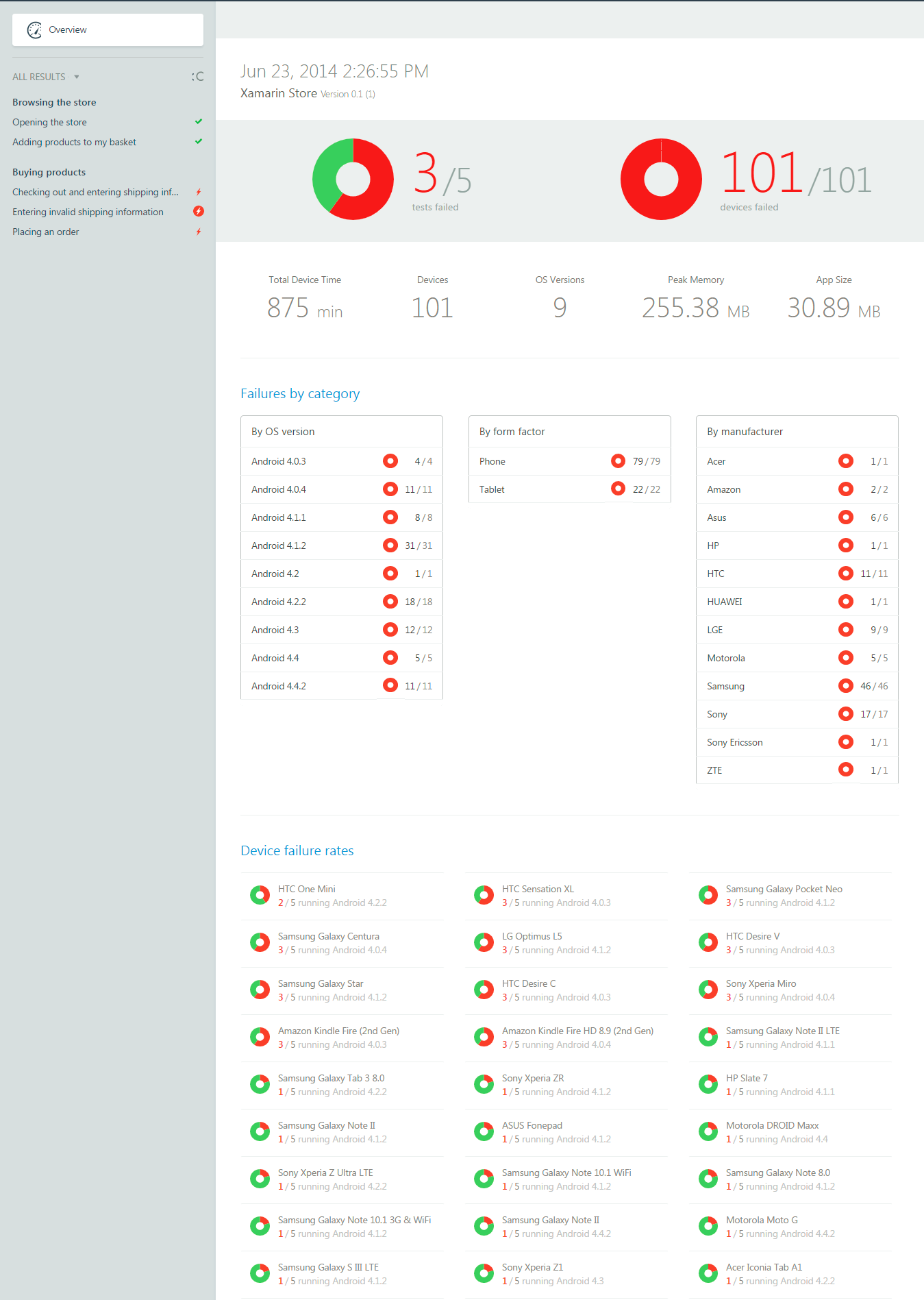
1. Go to the Test Cloud home page just like in previous example and click on the Xamarin Store icon.



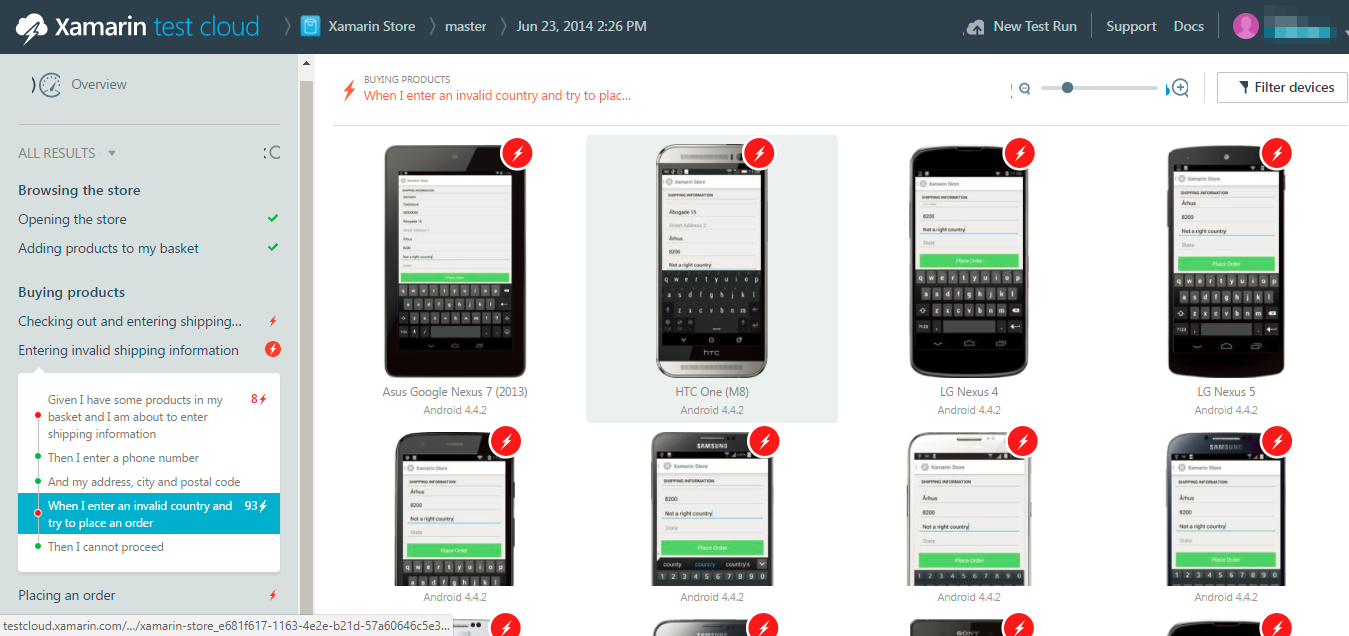
1. On the next screen, you’ll see a graphical representation of different test runs and a brief information about how many tests failed for total tests, what’s the application size and its peak memory usage information during different test runs.



1. This gives us a nice comparative look at how our application is performing on different set of runs. It is possible that application was performing fine during 1st run and then some code changes made some functionality to get failed. So this graph is very useful to monitor a timeline of changes that affected application functionality .
2. You can further click on of the graph or the test run to see it’s overview.
3. Now this screen gives us a great view at how running application on different devices can be monitored. It’s a very nice way to keep track of application on different devices and OS versions.



1. Let’s click on of the steps to see the results of the step on multiple devices.



1. The red icon shows failed tests. This page shows all the list of devices you chose to run the test on, it shows all the devices the test was passed and show red flag on failed devices.
2. You can further click on each device to get device specific screens and logs.

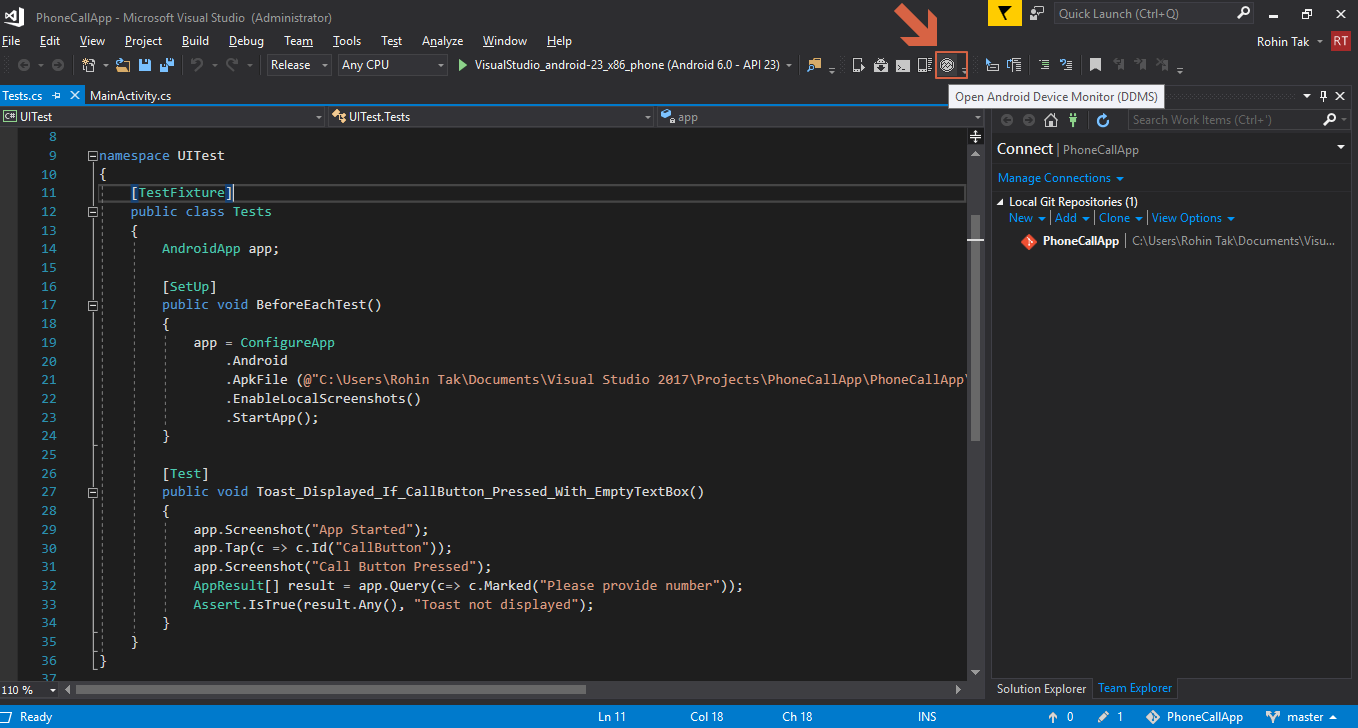
**Using Android Monitor tool for Monitoring**

Application performances are important. When application doesn't perform fast enough or crashes often, users are unhappy and application is doomed to fail. This is especially true in mobile world, where everything should be fast and responsive.

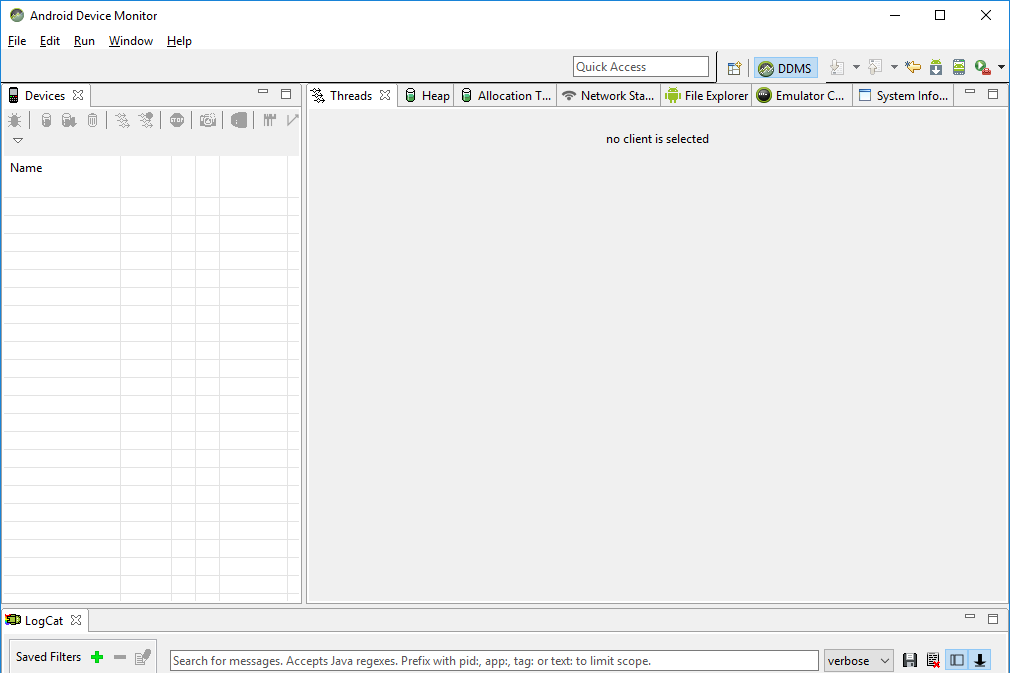
If you are developing cross-platform application using Xamarin and you run into performance issues, you may be in a bigger trouble than initially expected. The reason is multiple levels of abstractions, different memory heaps on different levels, unmanaged code and memory… You need a good profiler.

When it comes to Android, you should try using great tool called Android Device Monitor. Although it represents Android level information and you can't see Xamarin specific objects, analyzing trace or memory dump can still be very helpful.

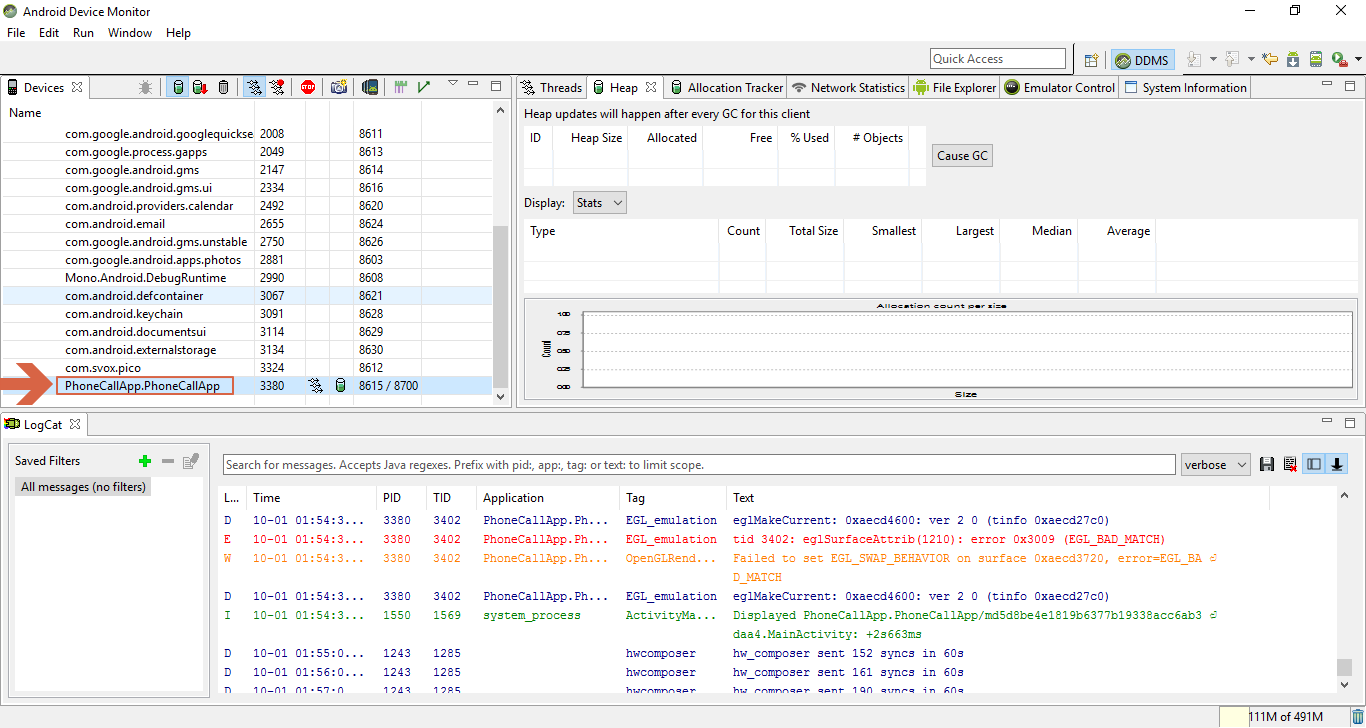
1. Go to visual studio, and from the toolbar, run Android Device Monitor.



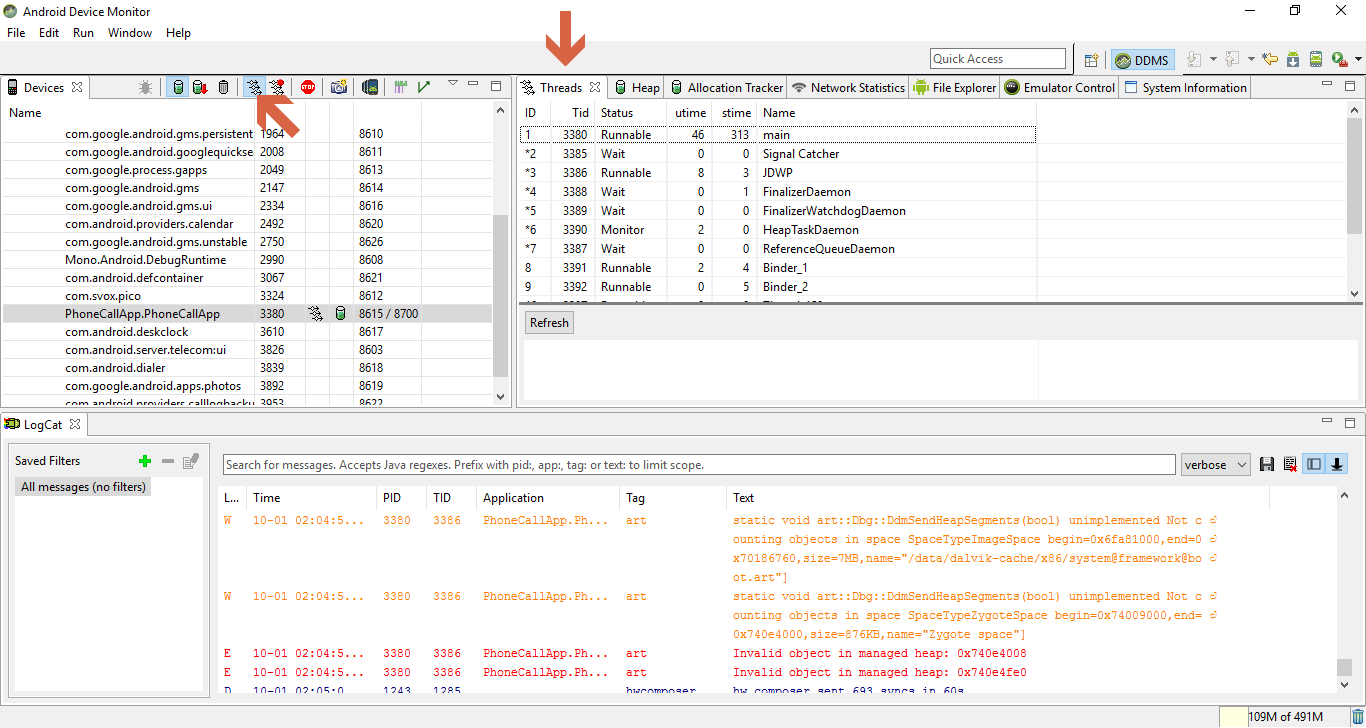
1. A new application Android Device Manager should open



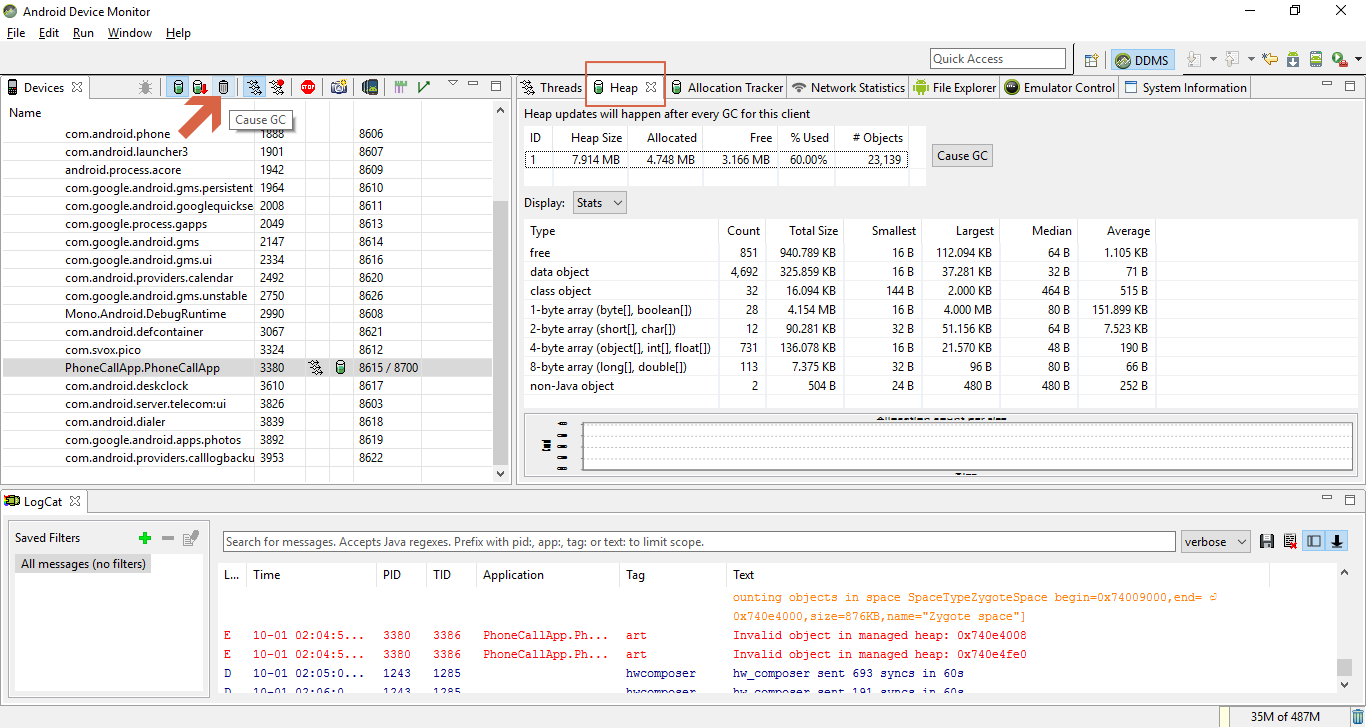
1. Go back to Visual Studio and run PhoneCallApp to get the device listed in Android Device Monitor.
2. You can run application on an emulator or a physical device connected to the computer.
3. Once the application is started, come back to android device monitor and you should be able to see the device running in the left pane.
4. Under the device name, you should be able to see all the processes running on the device.
5. In that list, select your application and you should be able to see related info



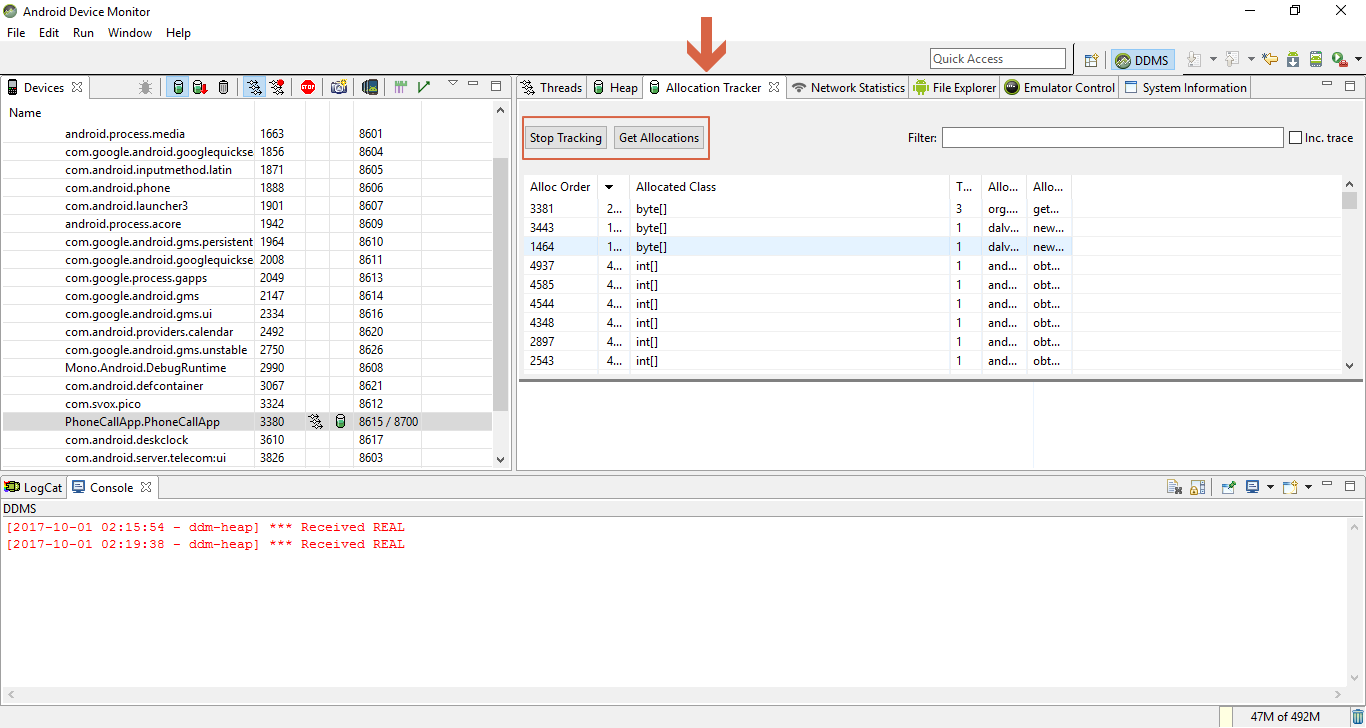
1. To monitor different threads running in your application, click on the update threads button on the left side pane’s toolbar. And then you should be able to see different threads running by your application on the right-hand side.



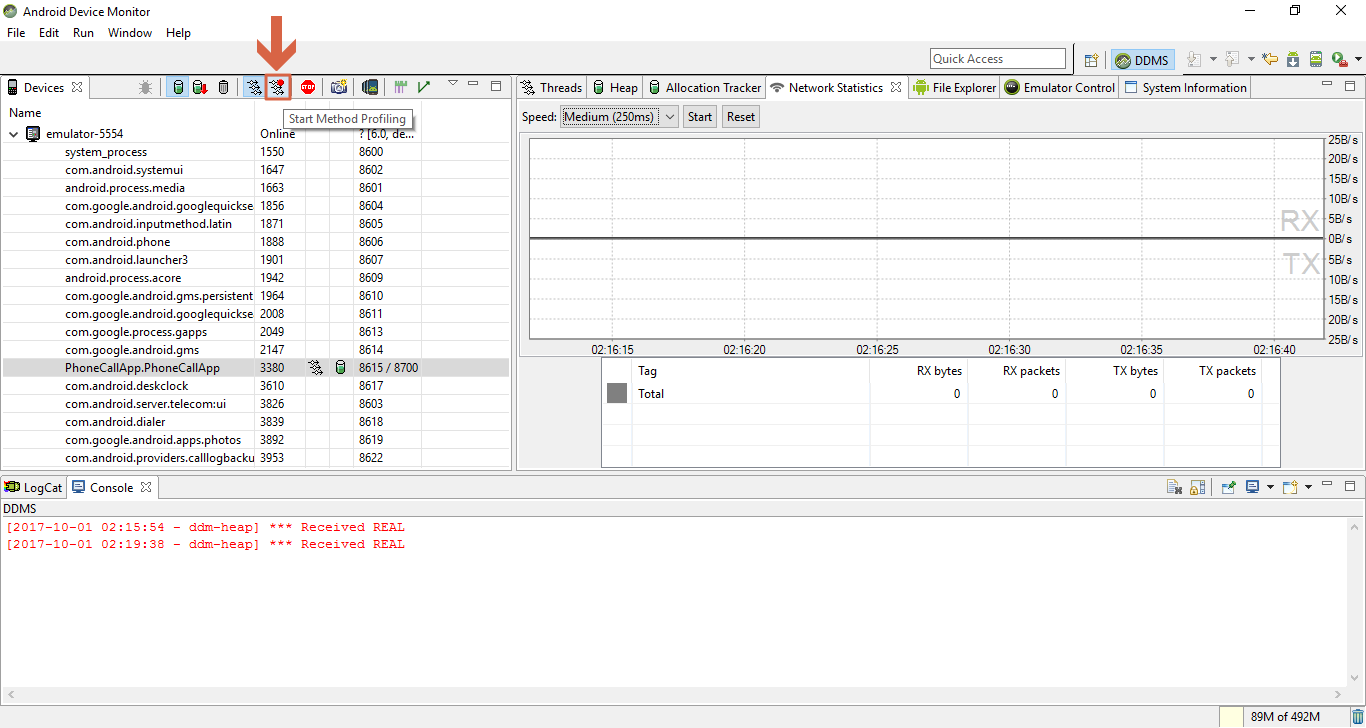
1. Being able to monitor threads run by your application can be very helpful in finding any unwanted background threads run by your application that might be causing extra battery usage or slowing down your application.
2. Sometimes thread might get deadlocked and it is very difficult to identify issues in such scenarios. This feature of ADM helps in a big way.
3. Similarly, monitoring memory usage of your application is a great tool to optimize your app and support low memory devices, and sometimes improving the performance by reducing memory consumption.
4. Click on cause GC (garbage collection) from the toolbar in the left pane and then select Heap on the right side to see the Heap memory allocation details.



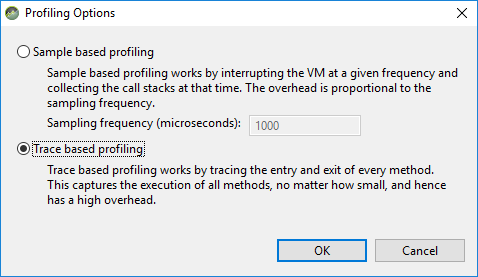
1. You can get more detailed Allocation monitoring in the Allocation tracker tab, click on start tracking and then get allocation button to get allocation details.



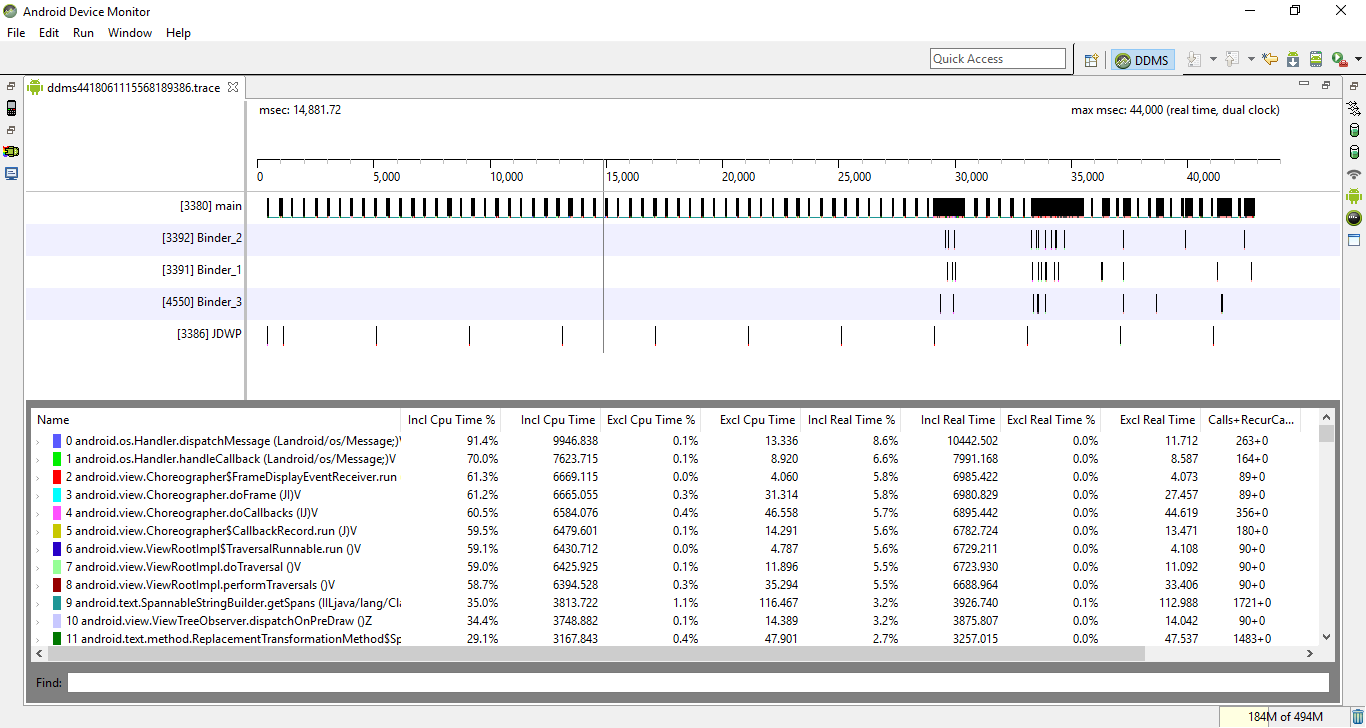
1. There are also options to monitor network related usage.
2. A very important feature of Android Device Monitor is profiling, which helps in profiling time taken and other details based on methods in your application code. It is great tool to identify method level performance and latency.  
   To do so, click on the start method profiling button on the tool bar in the left pane.



1. On the next step, select whether you want sample based or trace based profiling and click OK.



1. Do your tasks on the application and come back to ADM and click on stop method profiling.
2. Trace file is generated, containing the trace information you want to analyze.
   1. **A timeline panel**: describes when each thread and method started and stopped. We can go to specific time span and check what each thread did at that moment.
   2. **A profile panel**: provides a summary of what happened inside a method. We can see which one took the most of CPU time or how many calls it had.



1. Methods are on Android level, and you might need to compare with which method might have called these Android methods.

**Summary**

In this chapter, we learned about different types of monitoring techniques such as API monitoring, performance monitoring and functional monitoring. We also discussed about different tools for API level monitoring. Learned in detail about functional monitoring on multiple devices using Xamarin Test Cloud and performance monitoring using Android Device Monitor. In the next chapter, we’ll discuss about debugging and troubleshooting during different phases of development.