

# Cross-Platform App Development with Xamarin and Visual Studio App Center

Robotics

---

This article presents how to build up an environment for the development of cross-platform apps step by step. By following the guidelines of Continuous Integration, the process of development will conform to the guide of software engineering, which reduces risks in development, reduces repetitive manual processes of building, testing as well as deployment and enables continuous delivery of the apps.

Cross-Platform here means the ability that one set of code can be compiled to three different object code and then deploy to devices of three different platforms, namely Android, iOS and Windows UWP, without additional customization for each platform individually. Xamarin is the chosen framework to achieve this target.

One Xamarin sample project with different tests is used for the illustration in this article. The project is a simple calculator with basic functions but in most case sufficient for the illustration. [Here](#) is the link to the project on GitHub.

## Content

### Cross-Platform App Development with Xamarin and Visual Studio App Center

#### Content

#### 0. Prerequisites

#### 1. Source Control

##### 1.1 Create Repository on GitHub

##### 1.2 Cost and alternative solutions

#### 2. Develop and manage projects with Visual Studio

#### 3. Build apps with Visual Studio App Center

##### 3.1 Create corresponding apps in App Center

##### 3.2 Setup code signing file for app distribution

- 3.3 One hidden problem in App Center while building
- 3.4 Cost and alternative solutions
- 4. Test and Test Automation
  - 4.1 Unit Test
  - 4.2 UI test with Xamarin.UITest
    - 4.2.1 Create test project with Xamarin.UITest
    - 4.2.2 Submit UI test to App Center
  - 4.3 Using customized scripts in App Center
  - 4.4 Start tests automatically after build process
  - 4.5 Cost and alternative solutions
- 5. Team Collaboration in App Center
- 6. Publish the App and manage App lifetime
  - 6.1 App publish
  - 6.2 App lifetime management
- 7. Summary

## 0. Prerequisites

As prerequisites, accounts from the following websites or organizations should be available:

- GitHub account
- Microsoft account, will be used in:
  - App Center, to locate the developer and the Apps
  - Visual Studio

To distribute the apps in App Store of each platform, special signature files are needed, this will be discussed in chapter 3.2.

## 1. Source Control

Many tools support source control and version management, Git is one of the most popular tools. Therefore it is chosen for source control in this CI-solution (CI:

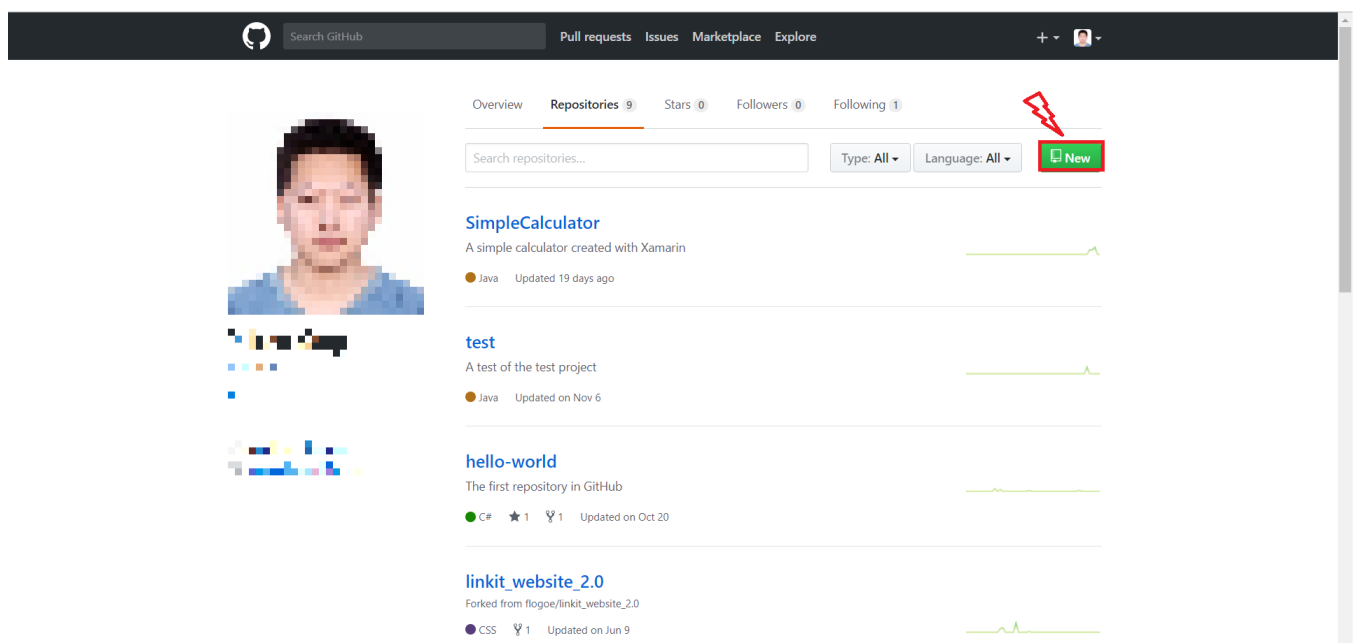
Continuous Integration). Since the solution is designed to use cloud resources as much as possible, GitHub is selected as Git server, so that no local Git server is needed.

If Git is not installed on your computer, please visit [Git official website](#) to download and install the suitable version.

## 1.1 Create Repository on GitHub

Below is a step-by-step specification for creating a public repository on GitHub, as mentioned in chapter 0, the prerequisite is a GitHub-account.

Firstly, log in with the account, redirect to the person homepage and then create new repository under the repository tab.



After given the repository name, an empty repository is created.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: yangpeiren / Repository name: MyRepositoryName ✓

Great repository names are short and memorable. Need inspiration? How about [bug-free-octo-doodle](#).

Description (optional): My project

☒ Public  
Anyone can see this repository. You choose who can commit.

☐ Private  
You choose who can see and commit to this repository.

☒ Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None Add a license: None ⓘ

Create repository

© 2017 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

Then the page will be redirected to the newly created repository, the address of the repository can be retrieved as shown in the picture below.

Code Issues Pull requests Projects Wiki Insights Settings

A simple calculator created with Xamarin

18 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

yangpeiren try to fix problem with build on test cloud

SimpleCalculator try to fix problem with build on test cloud

.gitattributes Add UI Test project and Unit Test project

.gitignore Project created

README.md Initial commit

README.md

SimpleCalculator

A simple calculator created with Xamarin

Clone with HTTPS ⓘ Use SSH

Use Git or checkout with SVN using the web URL.

https://github.com/yangpeiren/SimpleCalcu1

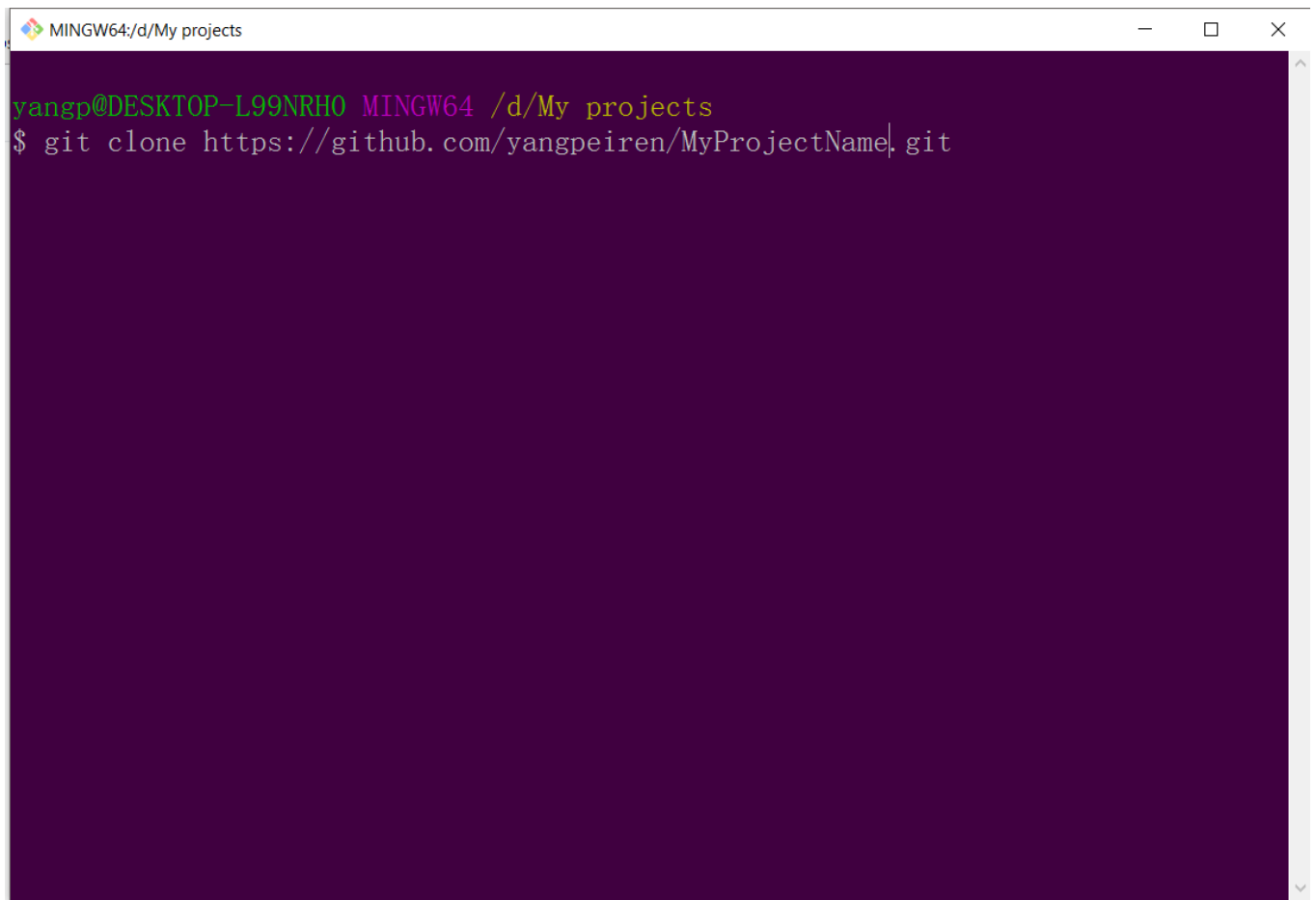
Copy to clipboard

Open in Desktop Open in Visual Studio

Download ZIP

After this step, there are many possibilities to clone the repository to the local directory. As an example, the standard tool **Git-Shell** is used here. Open the directory where the project is stored and then right click the mouse to activate Git-Shell, give the following command in and the remote repository will be then cloned to local.

```
1. $ git clone https://github.com/yangpeiren/MyProjectName.git
```

A screenshot of a terminal window titled 'MINGW64/d/My projects'. The prompt is 'yangp@DESKTOP-L99NRH0 MINGW64 /d/My projects'. The command entered is '\$ git clone https://github.com/yangpeiren/MyProjectName.git'. The terminal has a dark purple background and a light gray scrollbar on the right.

```
yangp@DESKTOP-L99NRH0 MINGW64 /d/My projects
$ git clone https://github.com/yangpeiren/MyProjectName.git
```

Until here, source control system for the CI-solution is built up.

## 1.2 Cost and alternative solutions

The cost for source control with Git and GitHub is **0**. Below is a list of alternative solutions for source control and their costs:

Source Control	Cost
GitHub with private repository	\$9
Bitbucket	free for up to 5 users, \$2 per user per month
Visual Studio Team Services	free for up to 5 users, \$30 per month for up to 10 users

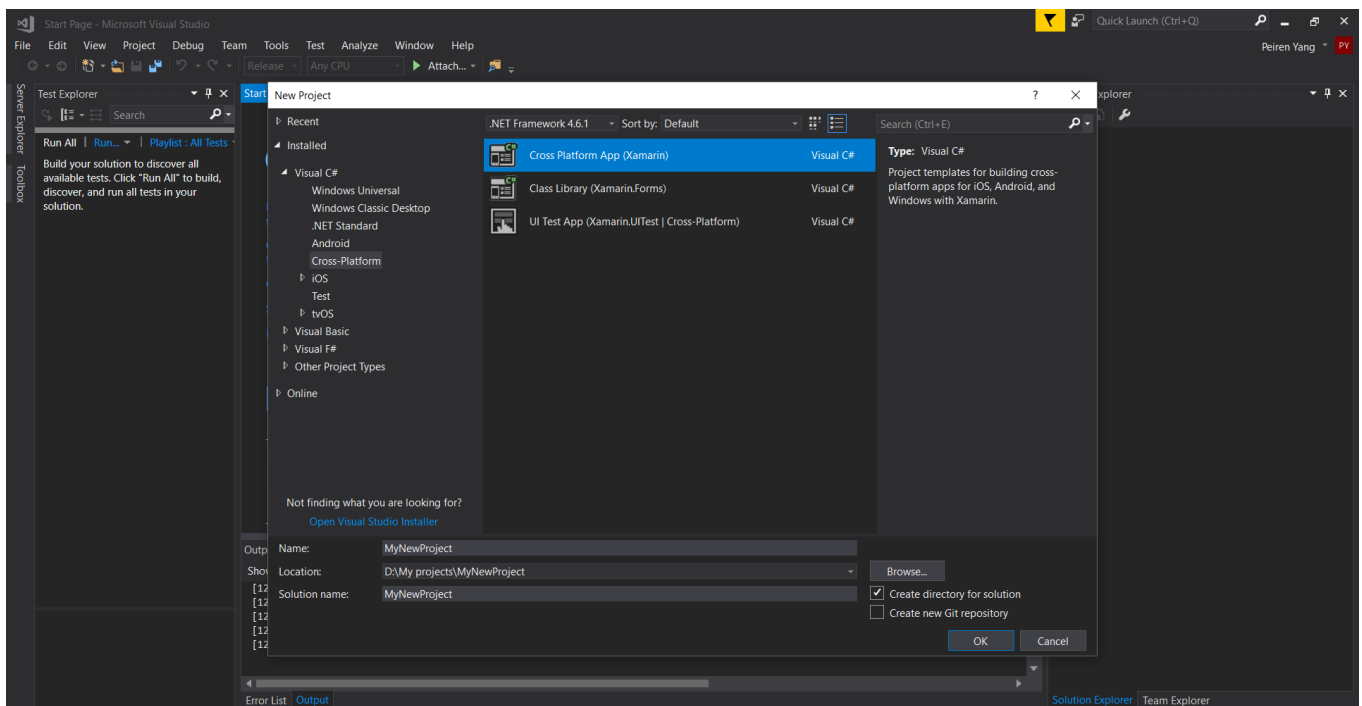
## 2. Develop and manage projects with Visual Studio

Visual Studio is an excellent IDE for C# programming. It is thus used in the CI-solution for following tasks:

- Programming
- Git management
- Tests management and test execution manually

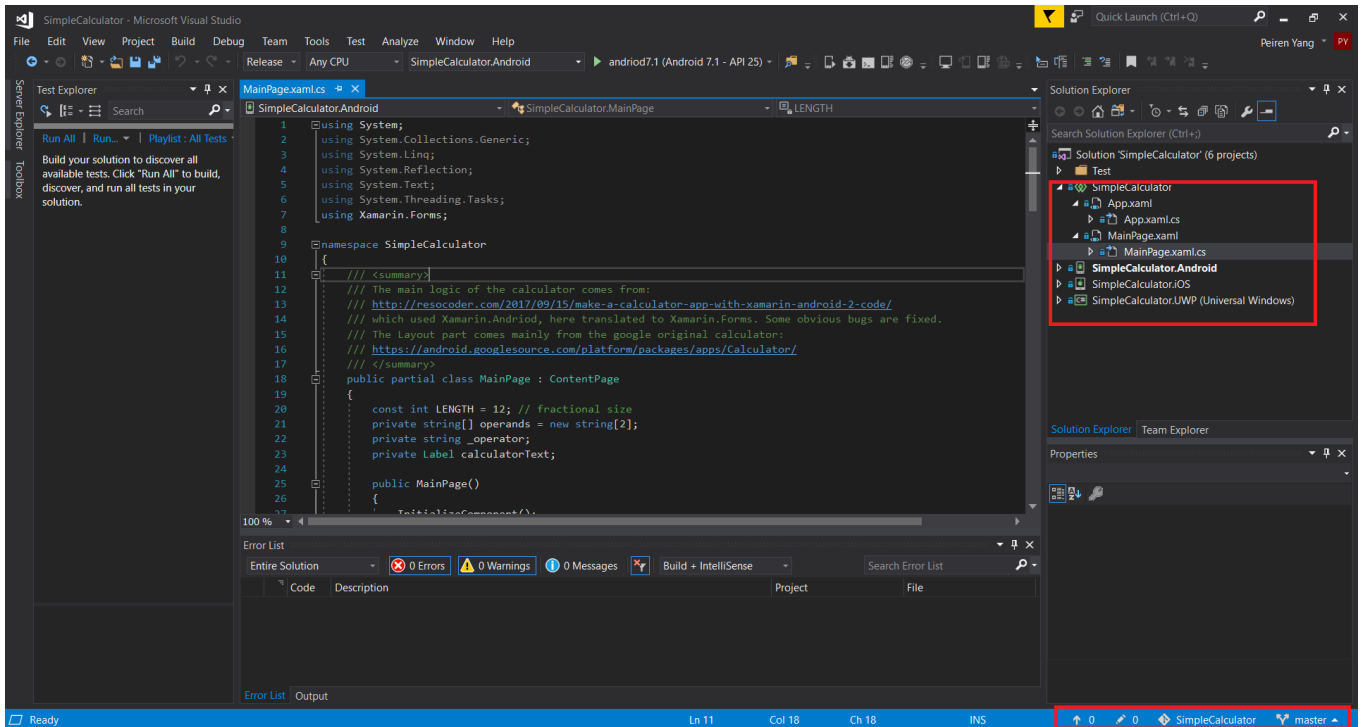
The build up process of the project with Git as source control is shown below.

First please make sure that the new project is created under the directory which was built up in chapter 1. Next, as the picture below shows, create a Xamarin cross-platform project.



The checkbox for creating new Git Repository should not be checked. Since the folder is already under the management of Git. After the project creation, two things are to be checked here:

- Four projects are contained in a complete solution, namely a root project, an Android project, an iOS project and a Windows UWP project.
- The Current directory is recognized as Git managed directory in Visual Studio.



To push the newly created projects to GitHub, the buttons on the bottom right corner of Visual Studio provide even more convenient user experience, although the Git-Shell can be used either.

### 3. Build apps with Visual Studio App Center

A lot of CI-tools support build of Xamarin projects. Visual Studio App Center is selected in this solution since no local server is needed in the CI-solution. The other tools will be listed at the end of this chapter.

**Visual Studio App Center** supports build, test, distribute and manage apps.

Furthermore, it supports not only Xamarin framework but also original Android, iOS projects.

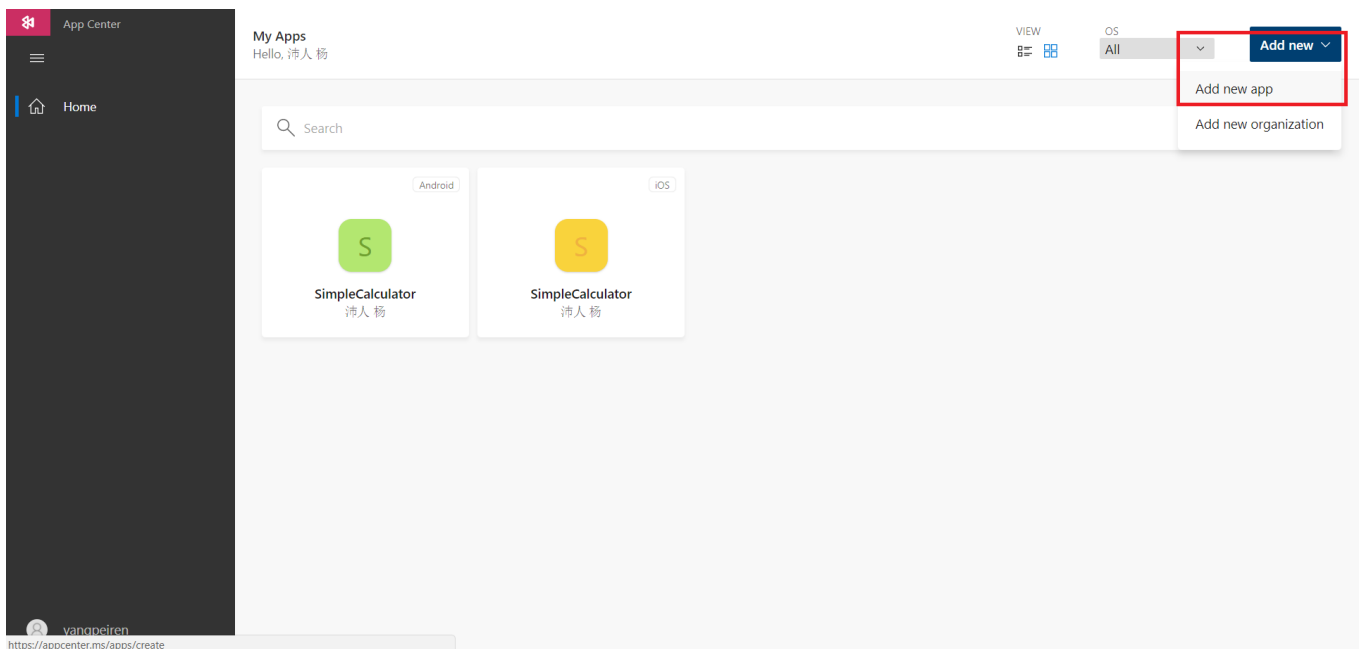
As a prerequisite a **Microsoft account** is needed, which is listed in chapter 0.

#### 3.1 Create corresponding apps in App Center

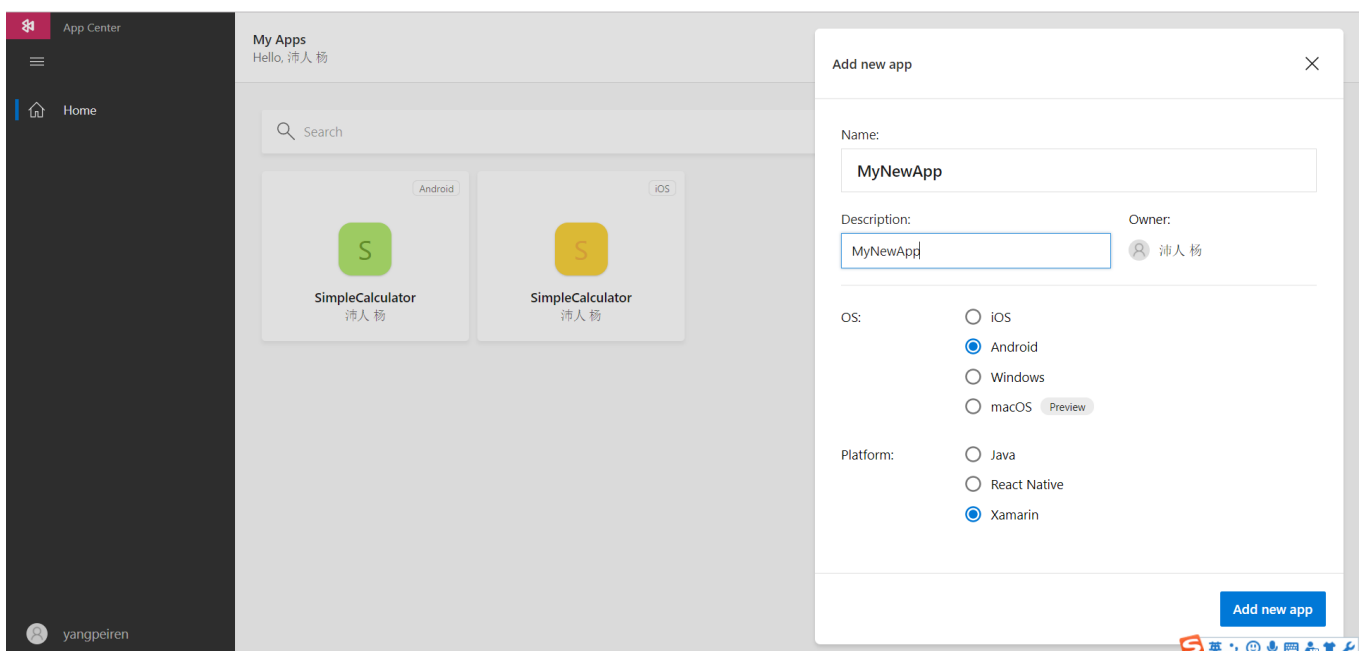
After logged into the App Center, new Apps can be added. As the projects created in

Visual Studio are cross-platform projects, and for each platform, there are different tools to build the target app, each of the three platforms should have a stub in App Center. Below is a process of how to create app stub.

In the main page of App Center, add new app can be triggered by the button at the top right corner.



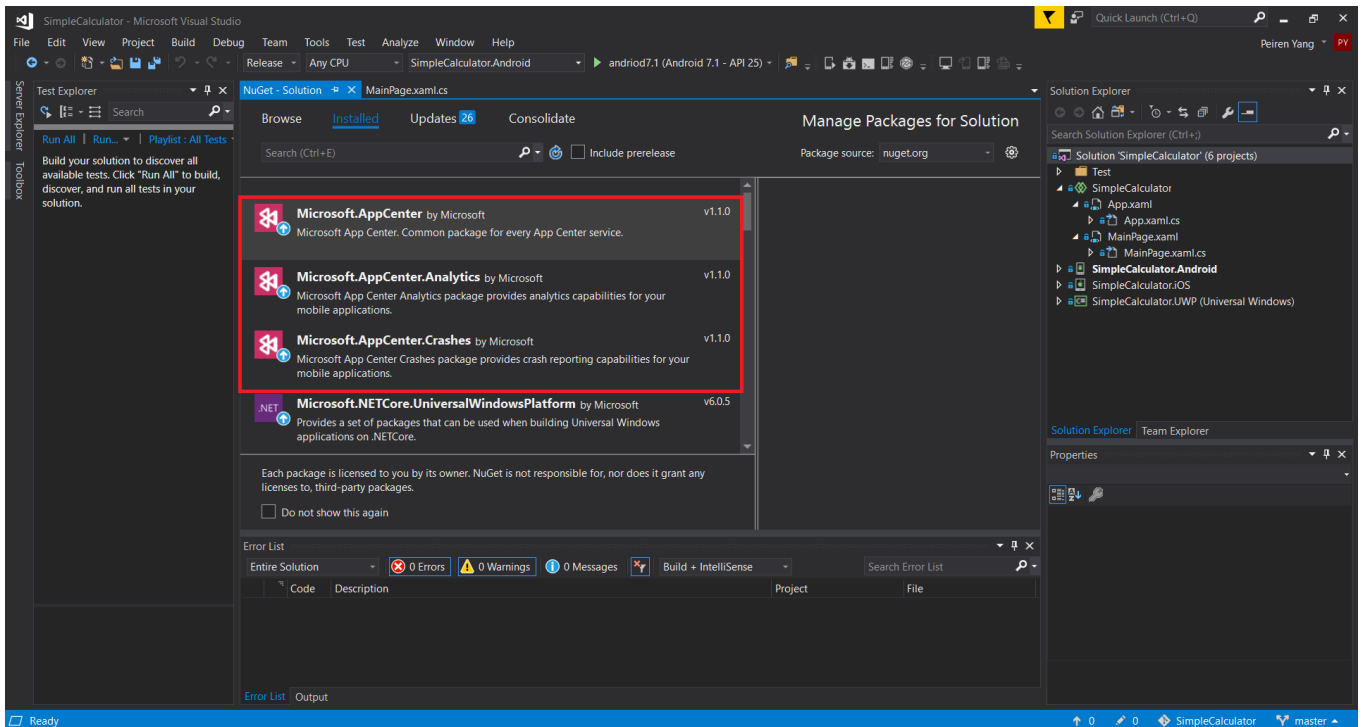
The name of the App and platform should be selected afterward.



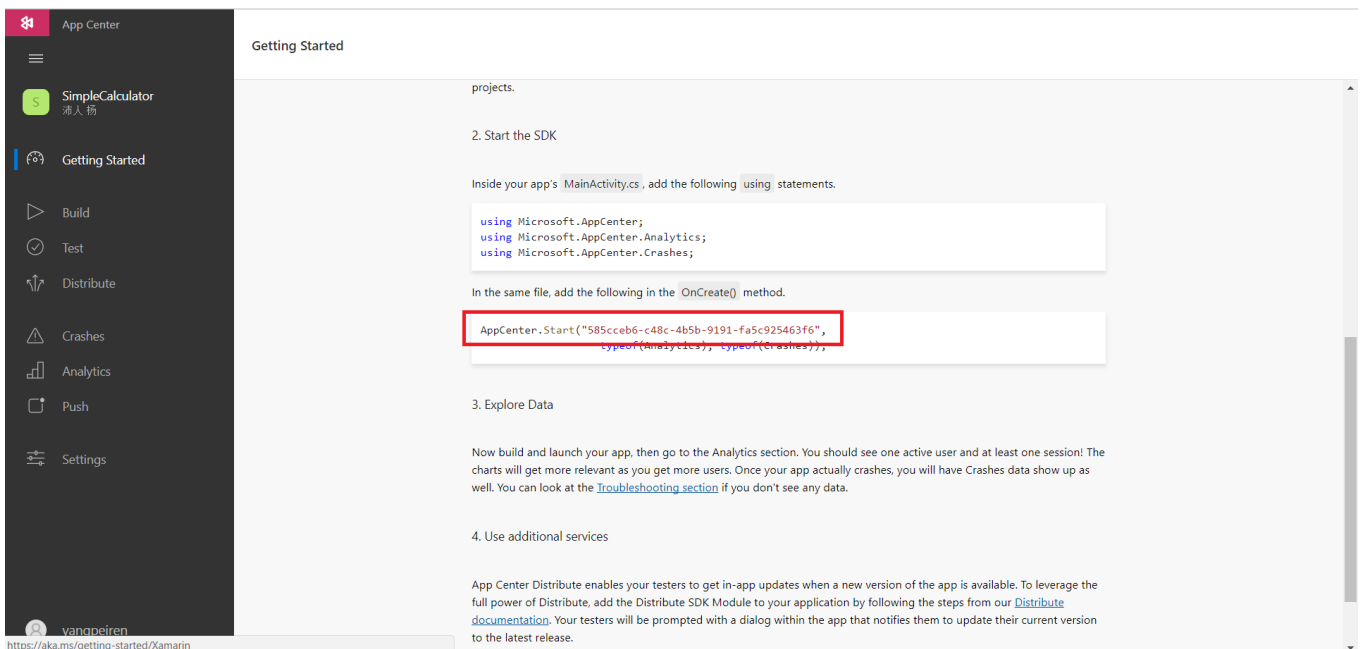
To use the build environment in App Center, the corresponding packages should be



installed in the project in Visual Studio, as shown in the picture below.

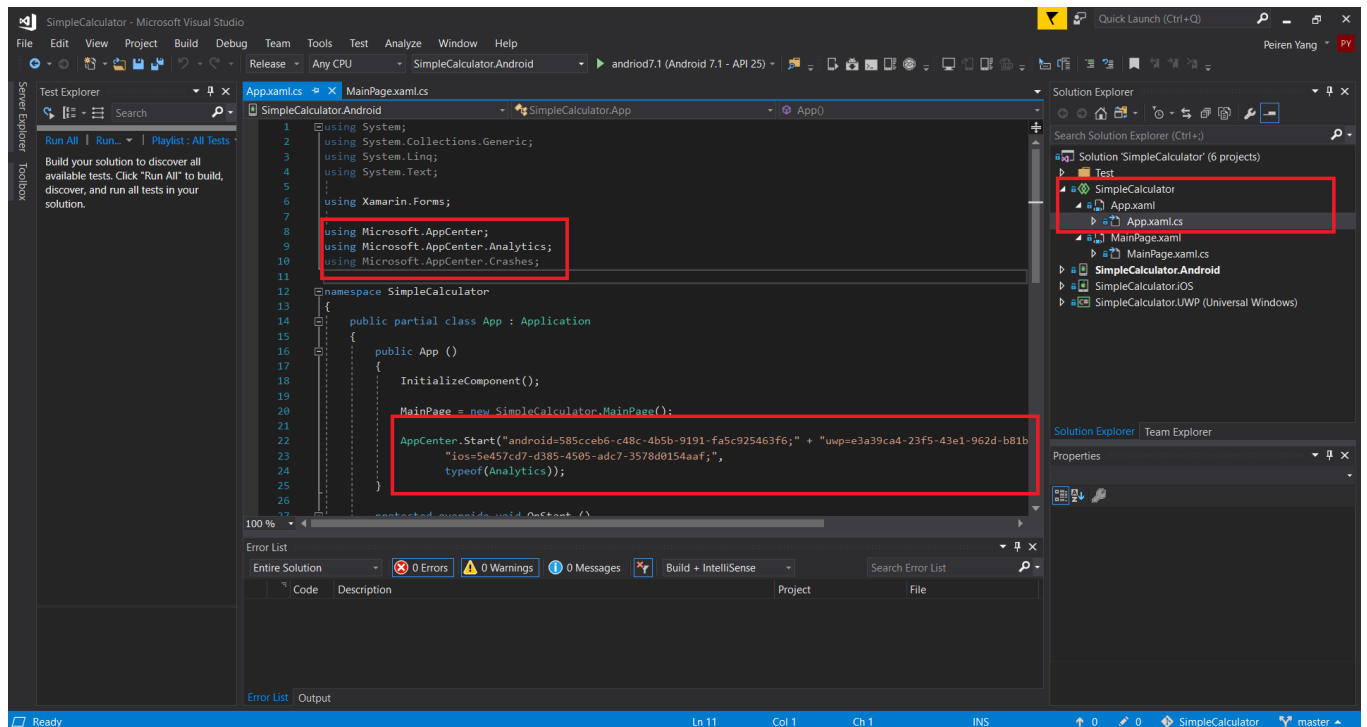


To distinguish each app in App Center, a string named app secret will be allocated after the app stub created, this can be retrieved in the app stub under the tab "Getting Started".



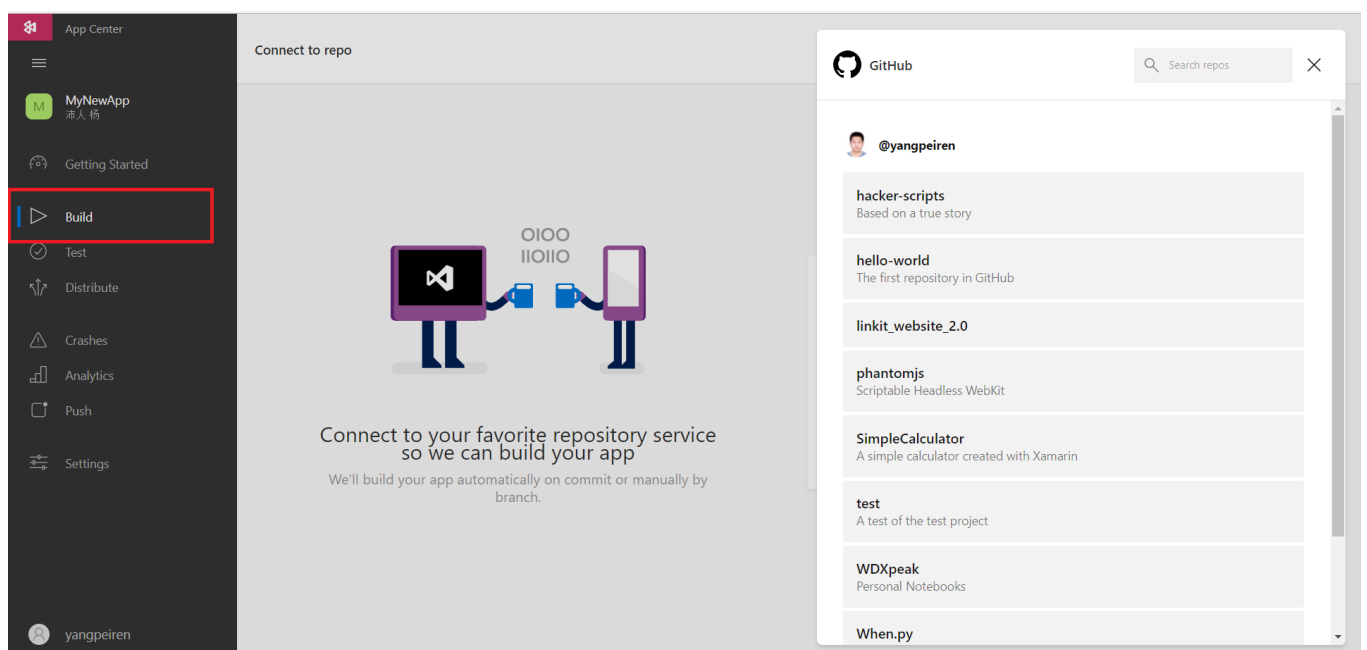
The retrieved app secret should be then added to the Xamarin project, more precisely in the main class of the main project (for example App.xaml.cs in the main project

SimpleCalculator).



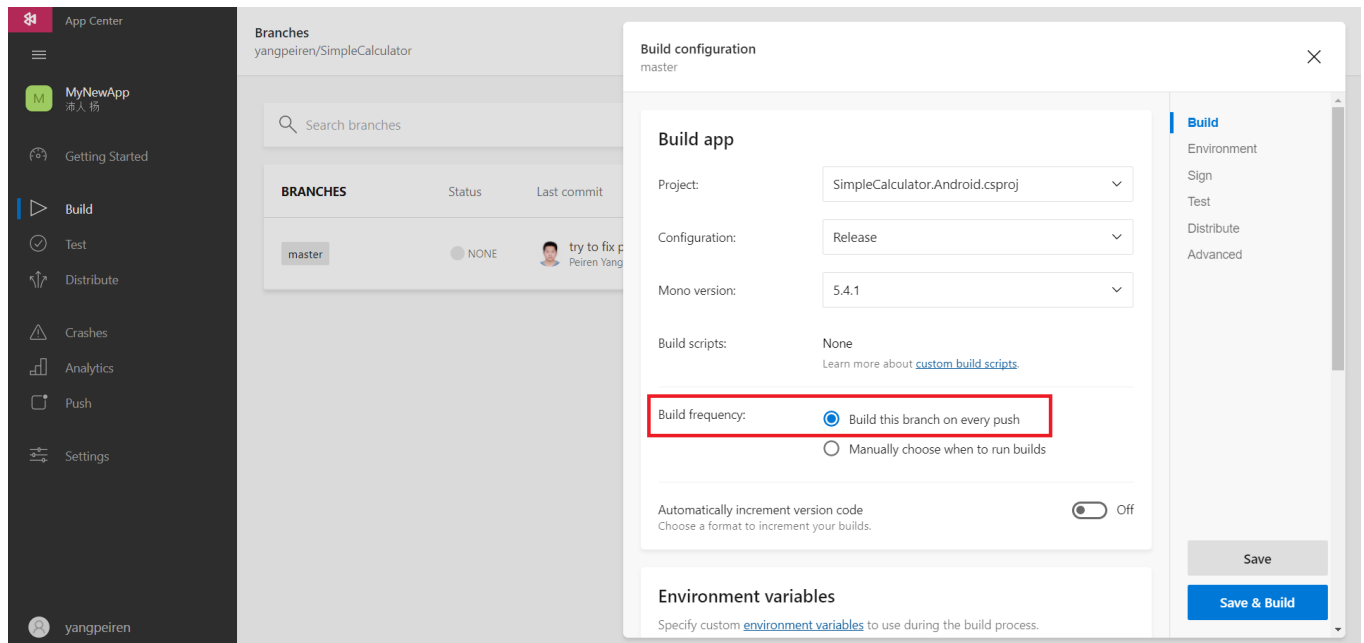
After this change, all preparations for app build were done, please do not forget to commit the changes and push them to the remote repository.


Next, the source code can be linked from GitHub to App Center for building the app. For the first time of link, the account and password of the repository owner is requested from App Center.



After successfully linked to the project repository, some parameters for the app build are to be specified:

- Setting the **build frequency** to "Build this branch on every push", the build can be directly triggered in Visual Studio by pushing changes to GitHub.

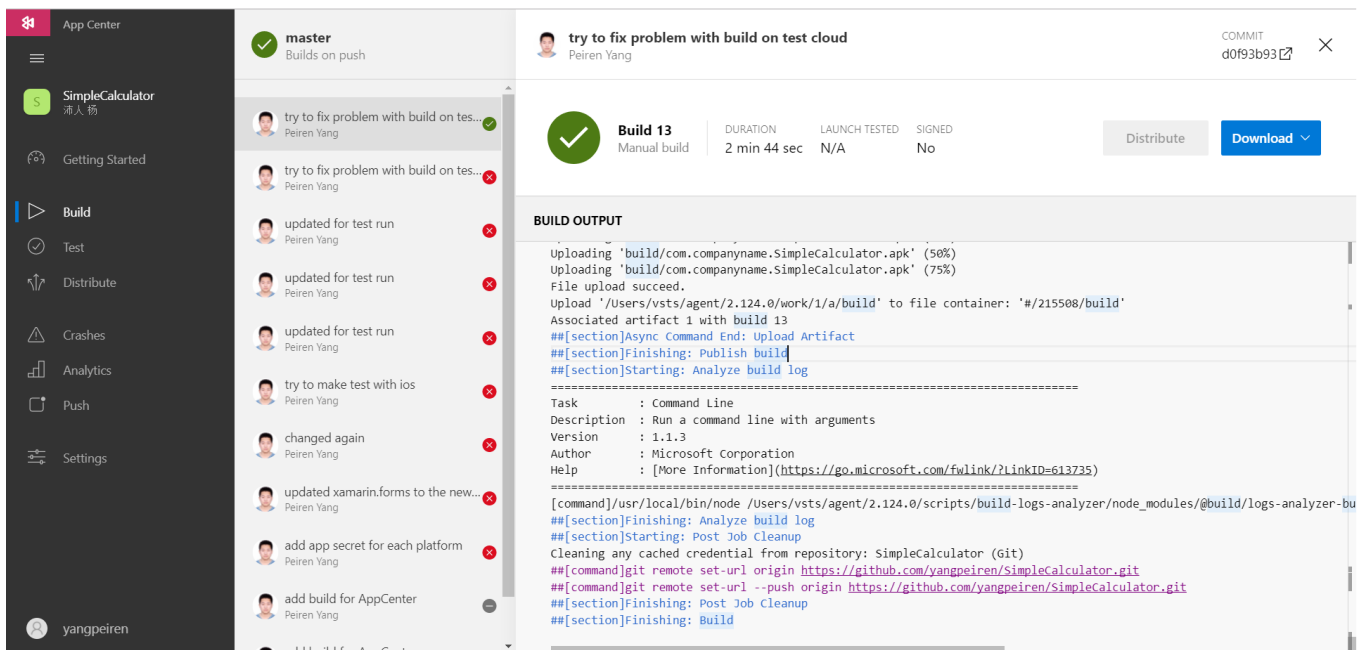


- **Build scripts** are the key components to automate the complete CI process, they will be discussed in chapter 4.3.
- If the option "**sign build**" is on, the built program can be then directly installed in mobile devices, this will be discussed in chapter 3.2.
- **Build badge** is another option, in some open source projects on GitHub a lot of badges can be seen in the readme file. App Center offers one badge either. The badge looks like this: .

The grammar to integrate the badge in the readme file:

```
1. [![Build status](https://build.appcenter.ms/v0.1/apps/7402eef4-95c4-473c-88e0-a8b2d3e4e369/branches/master/badge)](https://appcenter.ms)]
```

The readme file from sample project with the integrated badge can be found [here](#). Below is a screenshot of the App Center with a successful build, the built app can be then downloaded and published (build with a signature file).



## 3.2 Setup code signing file for app distribution

If an Android .apk file or an iOS .ipa file need to be installed on a physical device, the app must be signed. The signature file can be generated individually by the SDK of each platform.

Below is an example of generating a .keystore file with Java JDK:

Run a Command Prompt under administrator authority, redirect to the path where the Java JDK installed and type in the command below:

```
1. # create a keystore file which contains a single key and valid for 10000 days.
2. # generated file name is calculatorkey.keystore
3. # alias will be used afterward
4. keytool -genkey -v -keystore calculatorkey.keystore -alias calculator -keyalg RSA -keysize 2048 -validity 10000
```

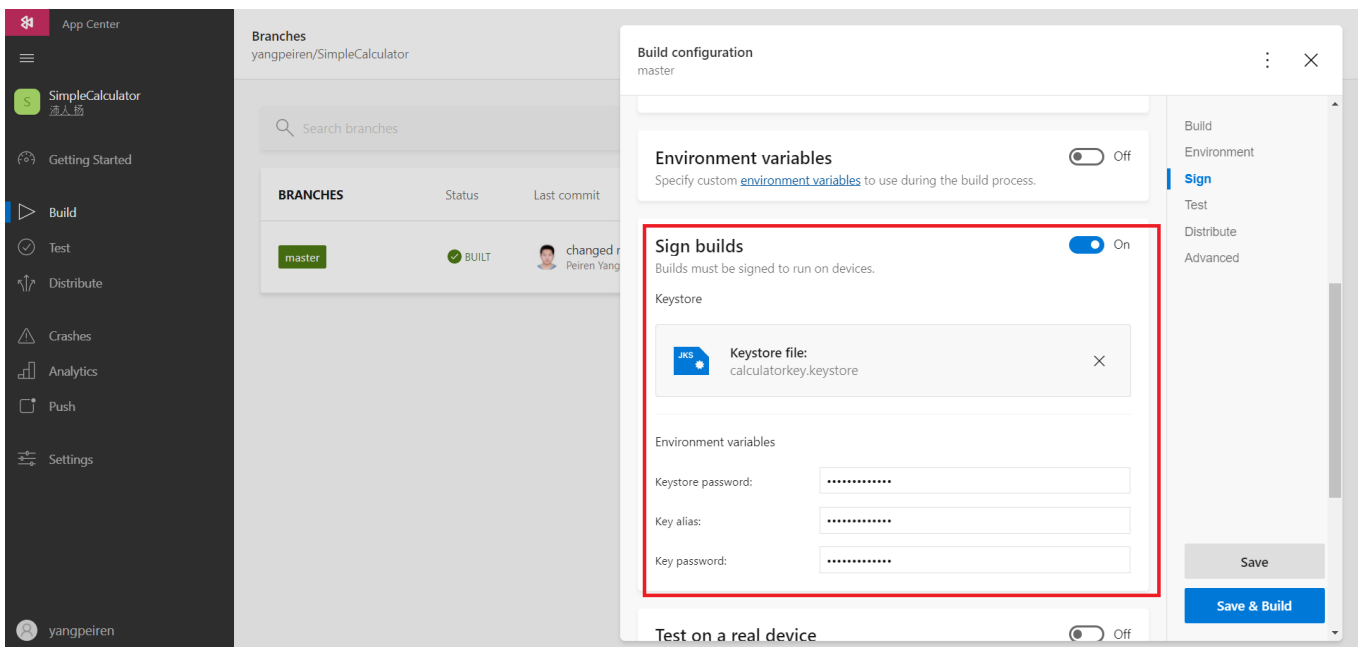
During the generation process, questions listed in the screenshot below should be answered and password for the signature file should be set up.

```
Administrator: Command Prompt
C:\WINDOWS\system32>cd C:\Program Files\Java\jdk1.8.0_131\jre\bin
C:\Program Files\Java\jdk1.8.0_131\jre\bin>keytool -genkey -v -keystore calculatorkey.keystore -alias calculator -keyalg
RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Peiren Yang
What is the name of your organizational unit?
[Unknown]: KIT
What is the name of your organization?
[Unknown]: KIT
What is the name of your City or Locality?
[Unknown]: Karlsruhe
What is the name of your State or Province?
[Unknown]: BW
What is the two-letter country code for this unit?
[Unknown]: EN
Is CN=Peiren Yang, OU=KIT, O=KIT, L=Karlsruhe, ST=BW, C=EN correct?
[no]: y

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
for: CN=Peiren Yang, OU=KIT, O=KIT, L=Karlsruhe, ST=BW, C=EN
Enter key password for <calculator>
(RETURN if same as keystore password):
Re-enter new password:
[Storing calculatorkey.keystore]

C:\Program Files\Java\jdk1.8.0_131\jre\bin>
```

After file generation, The .keystore file can be uploaded to the App Center to sign apps.



Please reference [here](#) for further information about code signing. Please reference [here](#) to see how to publish the built apps in App Store like Google Play.

### 3.3 One hidden problem in App Center while building

If the project to be built was once debugged in Visual Studio, the debug label will stay in the project file, and with this hidden label, the App Center can never make a successful build. Since App Center uses mono instead of the standard C# compiler to build apps, and mono does not support app building with embedded debug interfaces.

Solution to this problem is until now not officially given. There is a trick to fix the problem informally. Taken the project SimpleCalculator.Android as an example, the original **SimpleCalculator.Android.csproj** looks like this when opened with the text editor:

```
1. <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|A
   nyCPU' ">
2.     <DebugType>full</DebugType>
3.     <Optimize>true</Optimize>
4.     <OutputPath>bin\Release\</OutputPath>
5.     ...
6. </PropertyGroup>
```

The value in the property **DebugType** should be deleted, after deletion the file should look like this:

```
1. <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|A
   nyCPU' ">
2.     <DebugType></DebugType>
3.     <Optimize>true</Optimize>
4.     <OutputPath>bin\Release\</OutputPath>
5.     ...
6. </PropertyGroup>
```

With this modification, the build task in App Center then runs normally.

### 3.4 Cost and alternative solutions

For each user there are **4 free build hours monthly** with one build concurrency (not

possible to have parallel build tasks) available, building the sample project on Android platform costs about four minutes, this corresponds to **about 60 builds monthly**. The price for unlimited build time is **\$40 per month**, and for each further concurrency costs the same.

Below is a list of alternative solutions for app build and their costs:

Tool Name	Cost	Comment
Visual Studio App Center	\$40 per month	with unlimited build time
Jenkins	free	one server with Mac OS (special for compile iOS apps) is needed
TeamCity	\$1999	requirement similar to Jenkins, suitable for enterprise use

## 4. Test and Test Automation

Developers prefer CI-tools like Jenkins to make test automation in large projects since the jobs are easily configured with bash scripts and notifications can be customized to give feedback efficiently. However, physical server compared to cloud service has less flexibility in time and costs.

In this solution, test and test automation are supported by Visual Studio App Center, in which all tests are running on **real physical devices** in the "device farm" of App Center, **parallel running** is also feasible, which could save time for the developer. Other alternative tools will be discussed at the end of this chapter.

### 4.1 Unit Test

Both the unit test and UI test are designed to be done with the test framework NUnit here. A unit test can be easily written and tested locally. The design of unit test is omitted in the sample project.

As unit tests are usually very quick, they can be triggered after each build event with the post-build script, this will be discussed in chapter 4.4.

## 4.2 UI test with Xamarin.UITest

Below is a comparison between two popular test frameworks which supported by App Center:

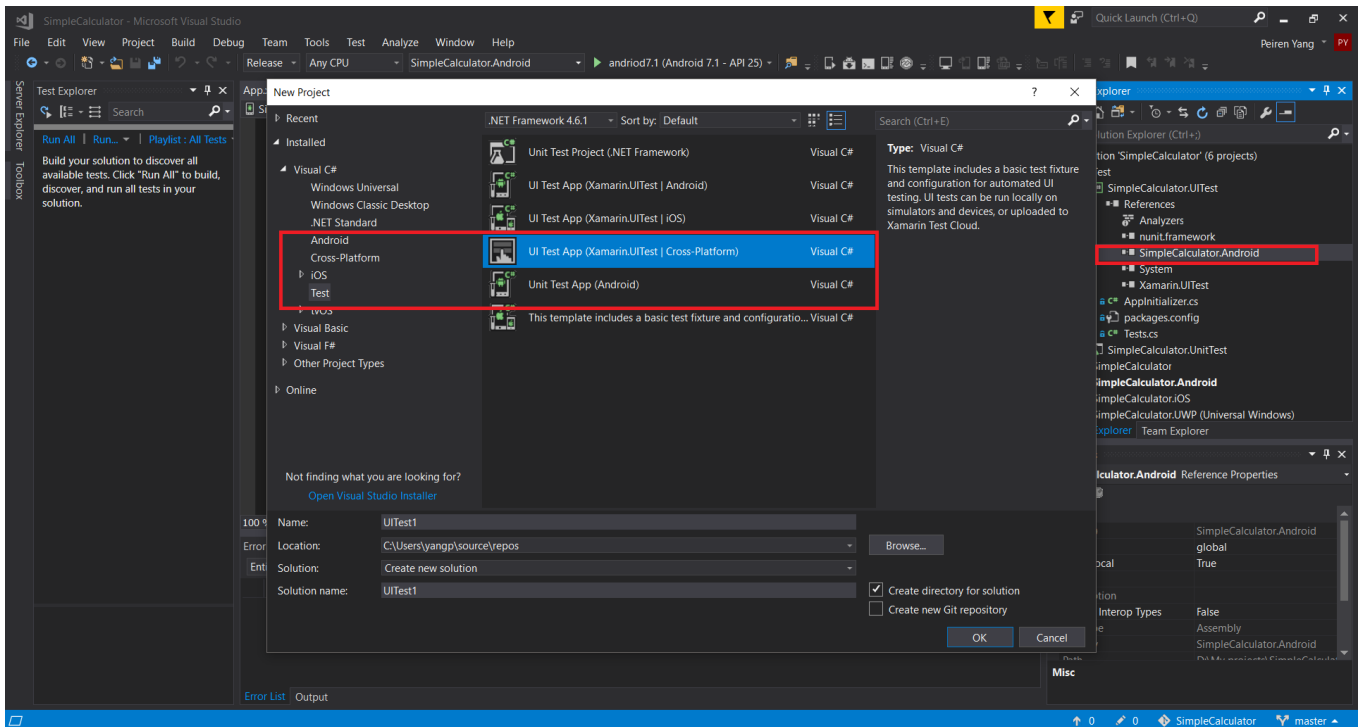
- Xamarin.UITest
  - Integrated into the solution of the project and managed by Visual Studio
  - Test cases written in C#
- Calabash
  - Test cases are very elegant, the readability is very good
  - Test cases written in Ruby

As the title reveals, the sample project uses Xamarin.UITest as the test framework, nevertheless, Calabash is recommendatory.

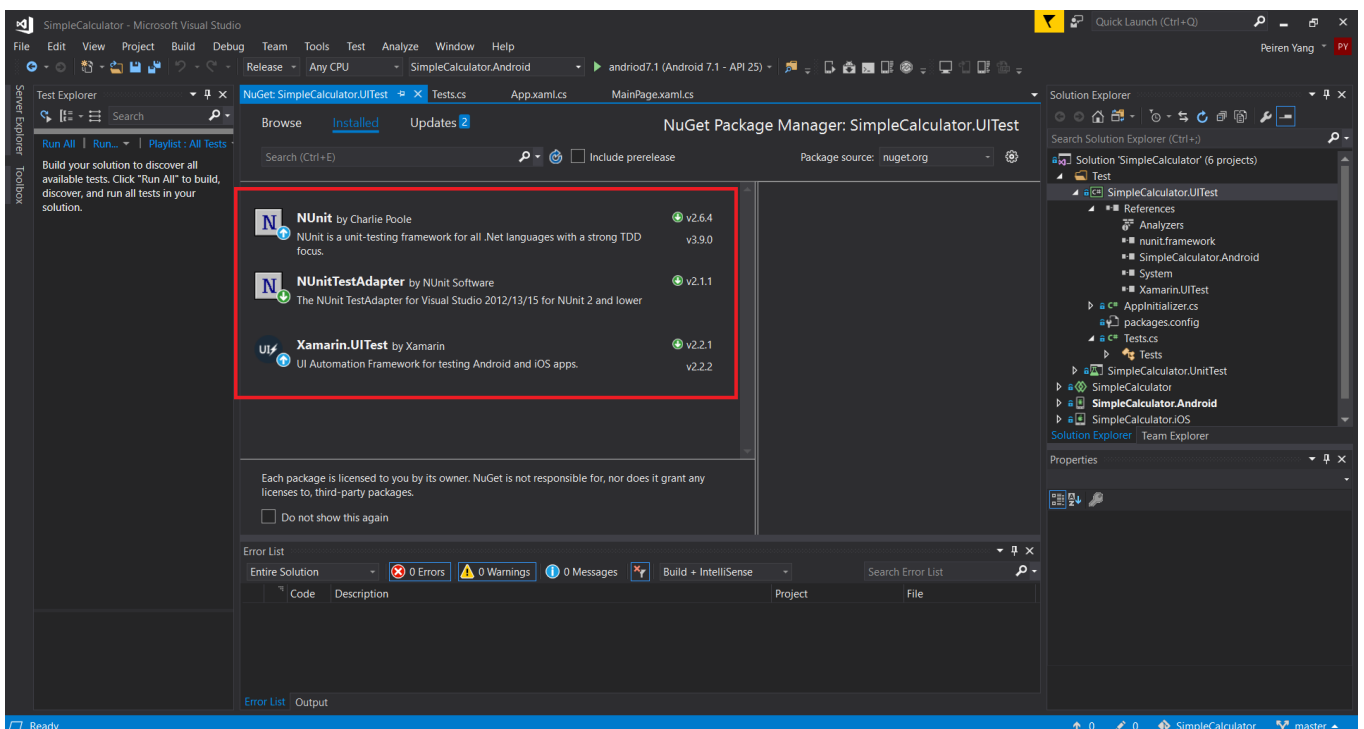
### 4.2.1 Create test project with Xamarin.UITest

New test project should be created within the frame of the current solution so that they can be managed together. After creation, references for the projects to be tested (the Android project and iOS project) should be added to the test project (as shown in the screenshot below with red highlights).





Next, three packages should also be installed in the project. Please note that the current Xamarin.UITest can only work with **JUnit Version v2.x** but not with NUnit Version v3.x.



After installation, test cases can be added to the project. As a demonstration, several button click tests and arithmetic operation tests were added in the sample project and

analyzed below.

In the UI test project, test cases should have access to the controls. However, the controls are a private attribute of the UI class and thus not accessible. Therefore the layout file of the original project should be modified, more precisely, all controls were extended with a new attribute **AutomationId**, for example:

Original:

```
1. <Button Text="7" Grid.Row="1" Grid.Column="0" Style="{StaticResource numberStyle}" Clicked="OnButtonClicked" />
```

With additional attribute "AutomationId":

```
1. <Button Text="7" Grid.Row="1" Grid.Column="0" AutomationId="button7" Style="{StaticResource numberStyle}" Clicked="OnButtonClicked" />
```

Below is a sample test case with explanations as comments.

```
1. ...
2. namespace SimpleCalculator.UITest
3. {
4.     [TestFixture(Platform.Android)]
5.     public class Tests
6.     {
7.         IApp app;
8.         Platform platform;
9.         public Tests(Platform platform)
10.        {
11.            this.platform = platform;
12.        }
13.        [SetUp]
14.        public void BeforeEachTest()
15.        {
16.            app = AppInitializer.StartApp(platform);
17.        }
18.        [Test]
19.        // categorize test
20.        [Category("AddTestUITest")]
21.        // one line is one test case, pass test parameters to the test function
```

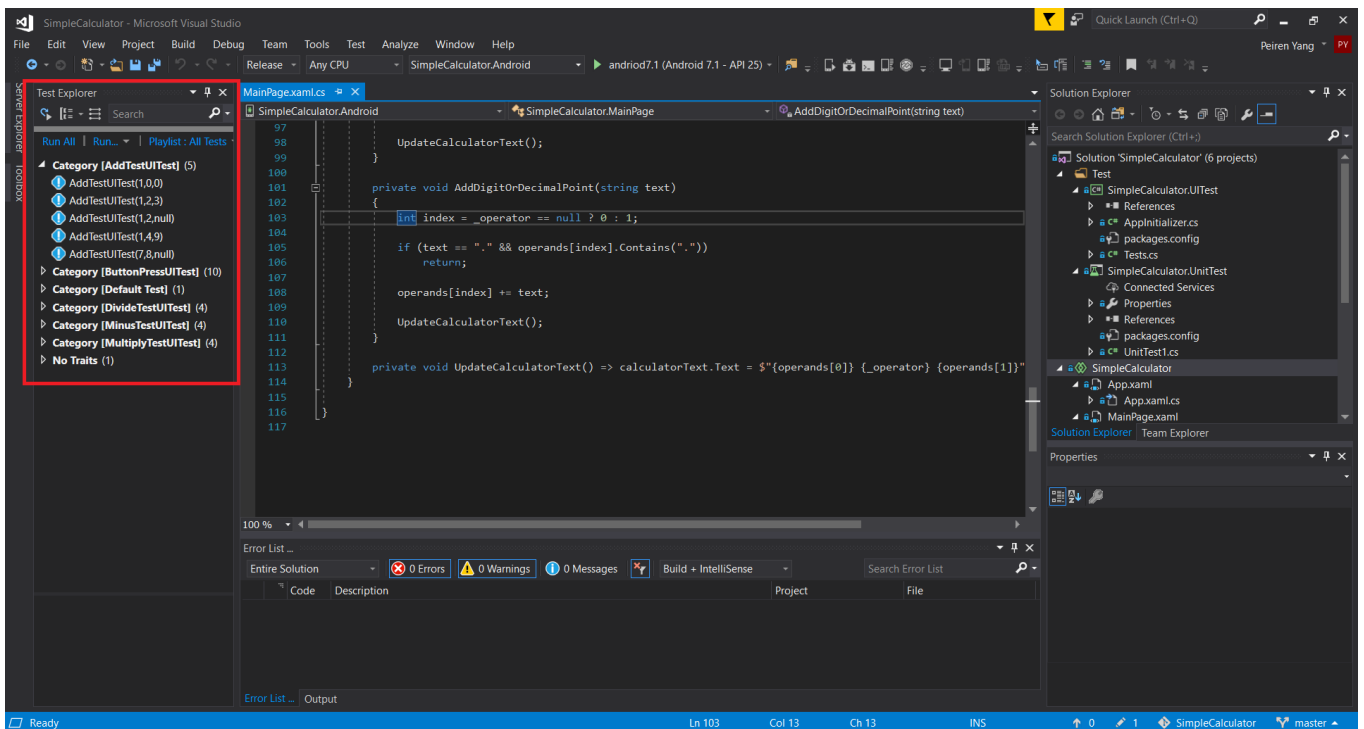
```

22.         [TestCase(1, 2, null)]
23.         [TestCase(1, 2, 3)]
24.         [TestCase(7, 8, null)]
25.         [TestCase(1, 4, 9)]
26.         [TestCase(1, 0, 0)]
27.     public void AddTestUITest(int number1, int number2, int? number3
28.     )
29.     {
30.         //Arrange
31.         // here uses AutomationId as handle of the controls
32.         app.Tap("button" + number1.ToString());
33.         app.Tap("buttonAdd");
34.         app.Tap("button" + number2.ToString());
35.         if (number3 != null)
36.         {
37.             app.Tap("buttonAdd");
38.             app.Tap("button" + number3.ToString());
39.         }
40.         //Act
41.         app.Tap("buttonEqual");
42.
43.         //Assert
44.         int value = number3 == null ? number1 + number2 : number1 +
45.         number2 + (int)number3;
46.         var appResult = app.Query("resultText").First(result =>
47.         result.Text.Trim() == value.ToString());
48.         Assert.IsTrue(appResult != null, "Result is not correct!");

```

Please visit [SimpleCalculator.UITest](#) to see full test project.

After one local build, all test cases will be recognized by the Visual Studio test explorer, and correspondently can be run locally on an Emulator of the mobile platforms or on windows (for UWP Apps). The test cases later can be locally validated or even debugged to meet the expectation of the developer.

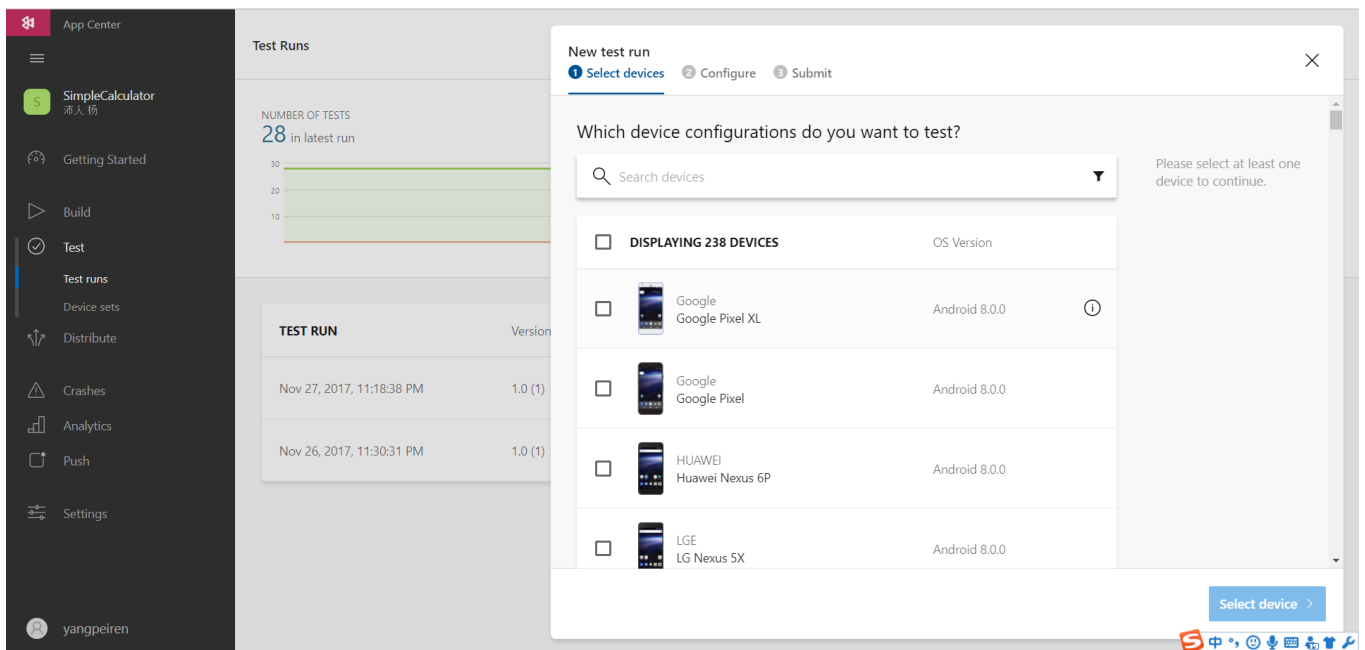


Thus the UI test project is set up, please don't forget to push the project to the remote repository.

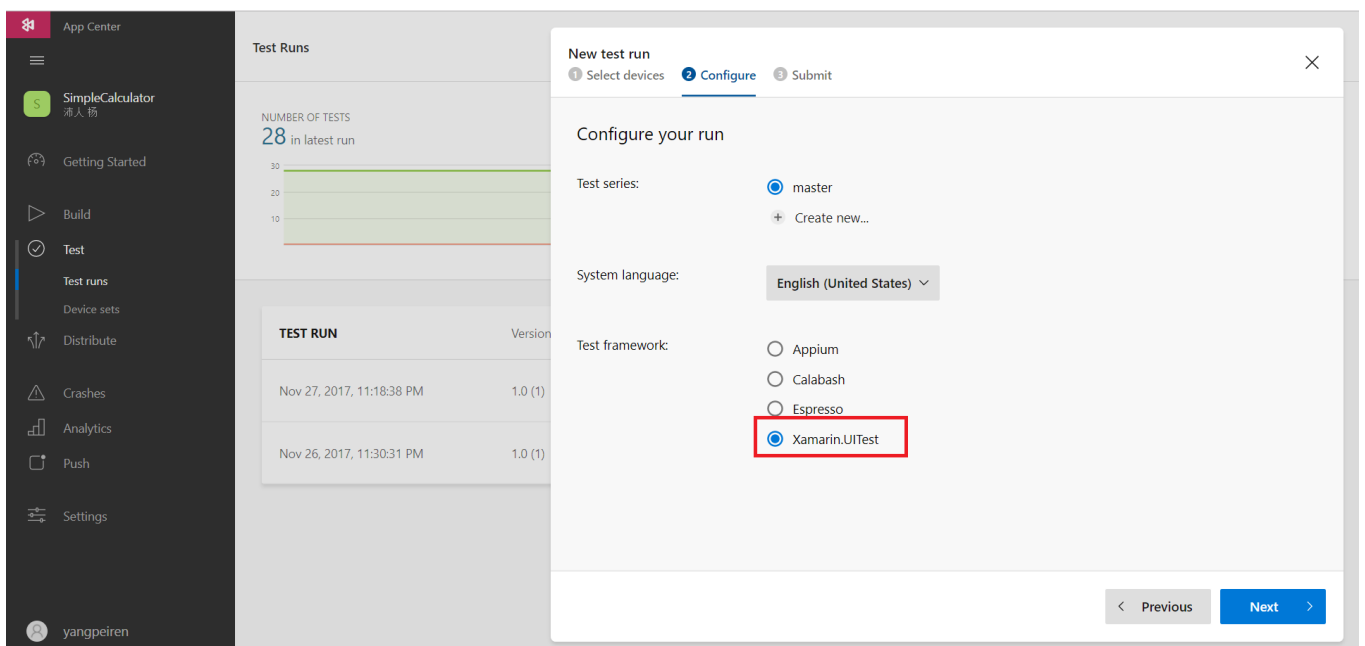
## 4.2.2 Submit UI test to App Center

Back to App Center, a new test can be configured under the tab test, test run.

Firstly, a list of devices with their OS versions should be figured out (App Center offers an additional function named "Device set", under which preferred sets of devices can be defined and referenced many times).



Select Xamarin.UITest as the test framework.



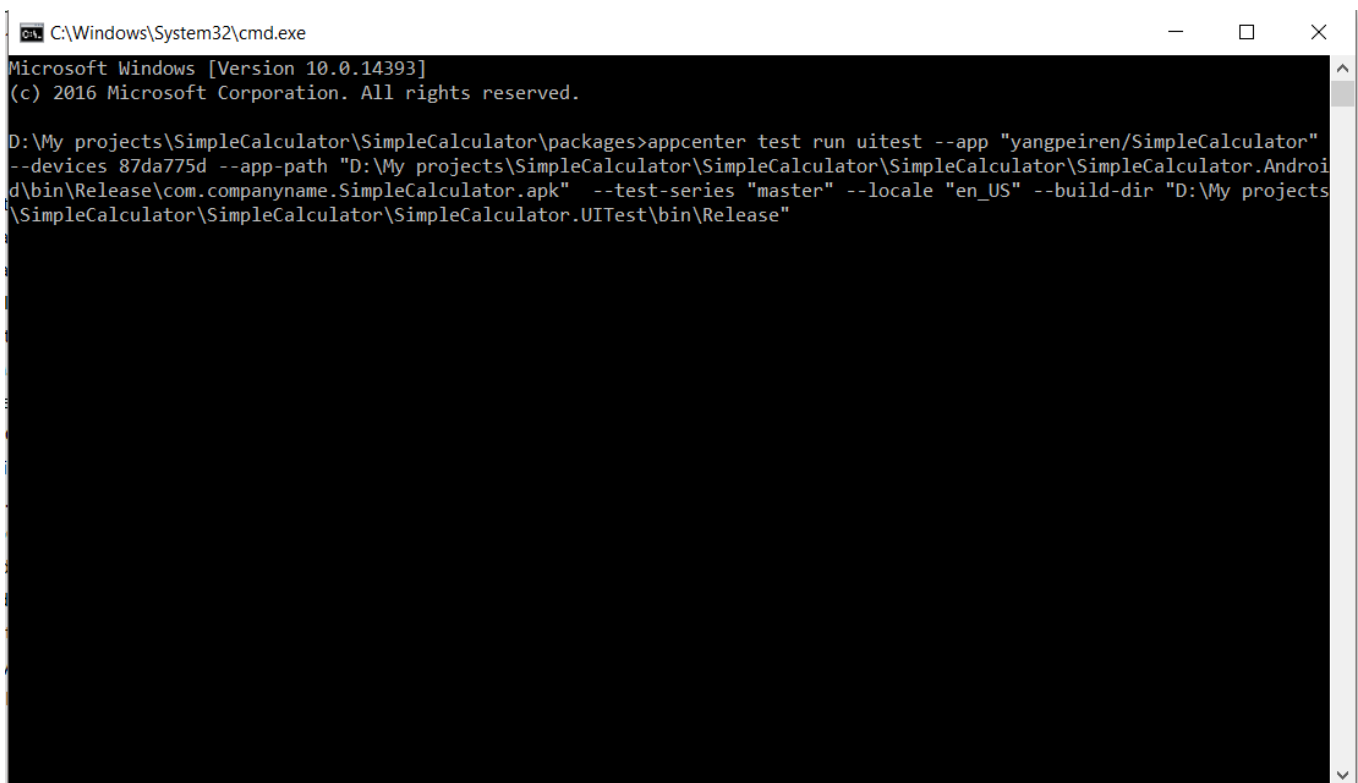
Then a guide command line for submitting the tests is generated, to trigger the test in App Center with Command Prompt, **Node.js** and **App Center Command Line Interface (CLI)** (an open source project from Microsoft, which is managed on GitHub, please visit [the readme file](#) on GitHub to see the full functions) should be installed in the computer, the first one can be downloaded from Internet, the second one can be installed by following command in Command Prompt:

```
1. $ npm install -g appcenter-cli
```

Notice: In the first time of the App Center in the command line, one time login is needed. The command can be found in the CLI readme file above.

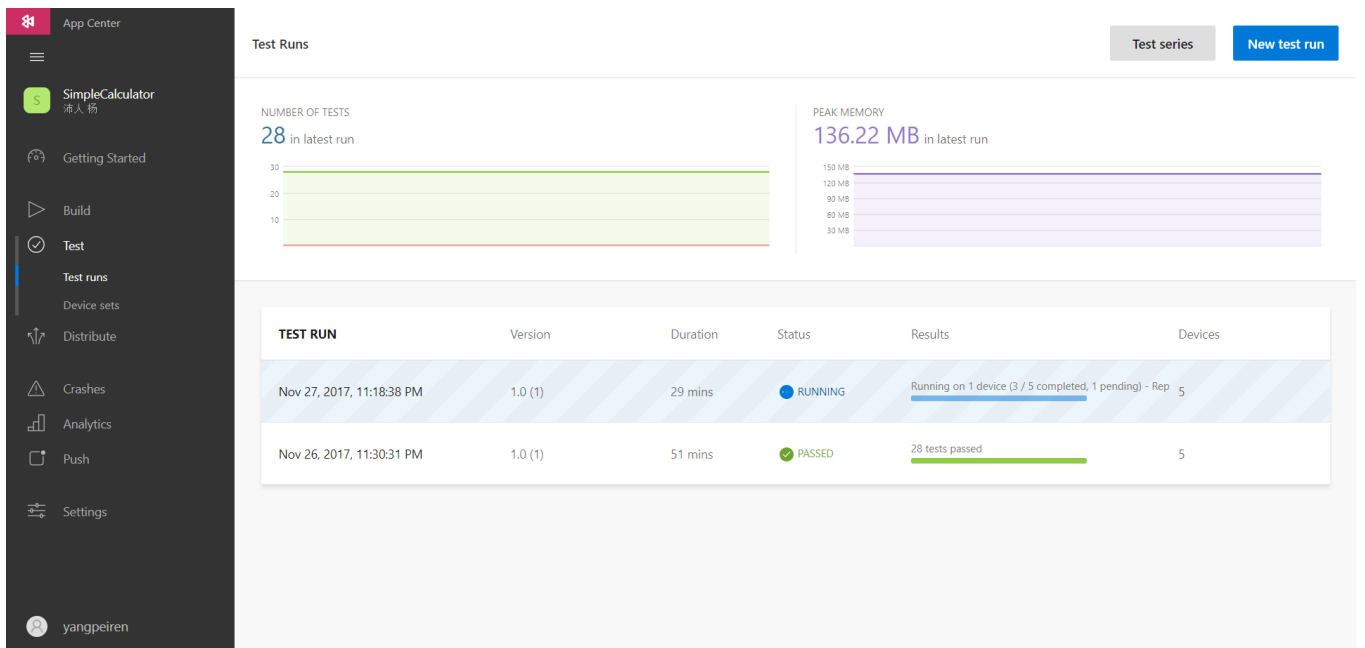
The UI test can be submitted with local command line to the App Center by the command below:

```
1. $ appcenter test run uitest --app "yangpeiren/SimpleCalculator" --devices 87da775d --app-path "D:\My projects\SimpleCalculator\SimpleCalculator\SimpleCalculator\SimpleCalculator.Android\bin\Release\com.companyname.SimpleCalculator.apk" --test-series "master" --locale "en_US" --build-dir "D:\My projects\SimpleCalculator\SimpleCalculator\SimpleCalculator.UITest\bin\Release"
```

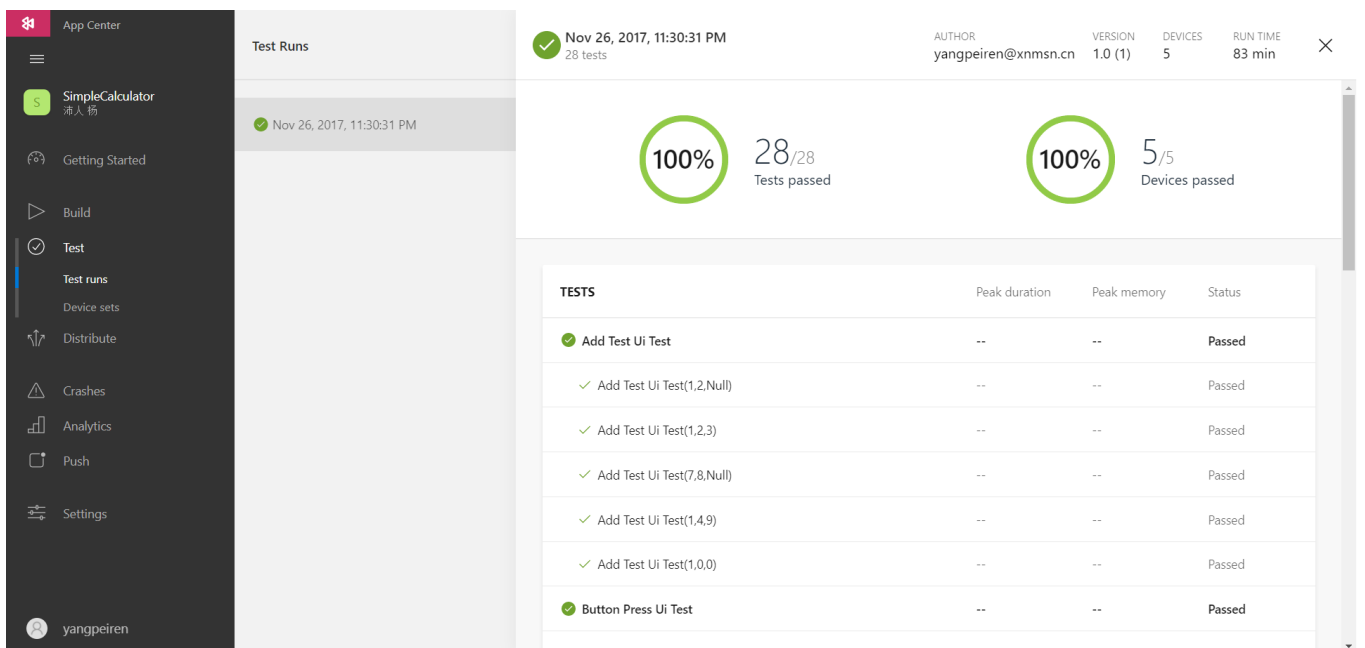


The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The window content includes the Microsoft Windows version (10.0.14393) and copyright information (© 2016 Microsoft Corporation). The command prompt shows the current directory as "D:\My projects\SimpleCalculator\SimpleCalculator\packages" and the command being executed: `appcenter test run uitest --app "yangpeiren/SimpleCalculator" --devices 87da775d --app-path "D:\My projects\SimpleCalculator\SimpleCalculator\SimpleCalculator\SimpleCalculator.Android\bin\Release\com.companyname.SimpleCalculator.apk" --test-series "master" --locale "en_US" --build-dir "D:\My projects\SimpleCalculator\SimpleCalculator\SimpleCalculator.UITest\bin\Release"`. The command is entered on a single line, wrapping around the width of the window.

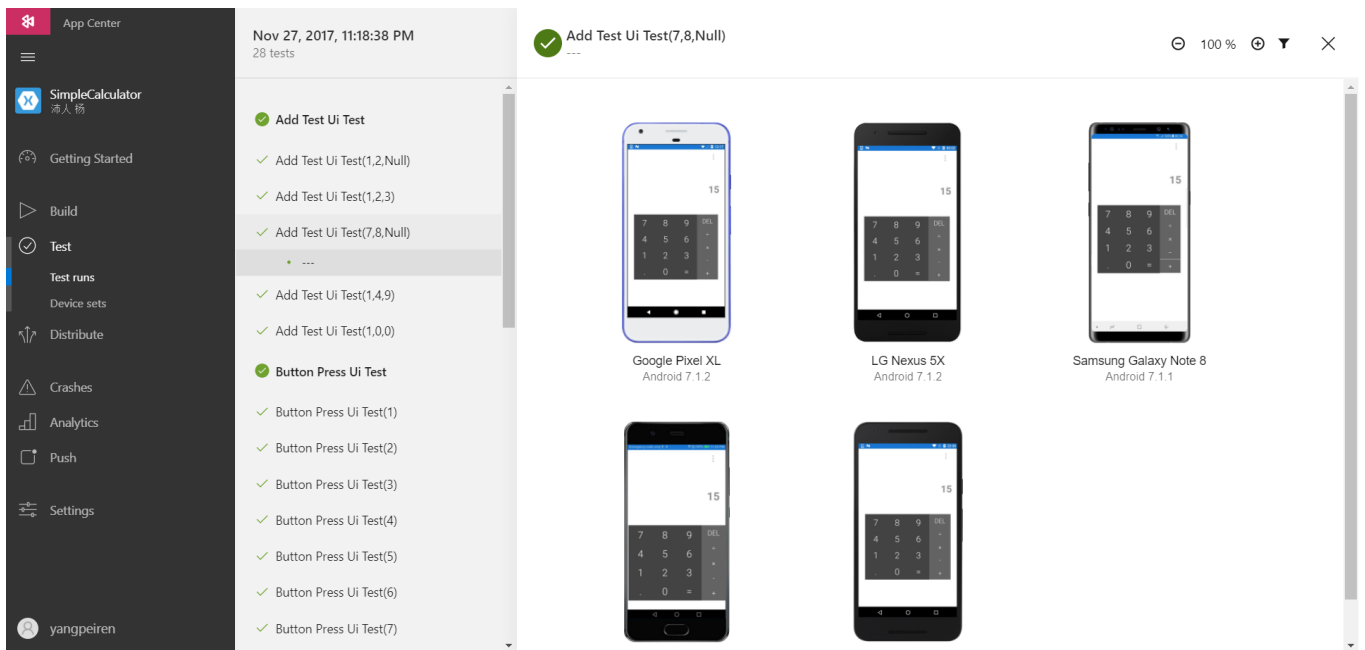
The running status of the submitted tests can be seen immediately in App Center under the test tab of the side panel.



When finished, the test result is given as shown below.



When clicked on every single test, screenshots of execution for each test device are already made for all test devices, which is very user-friendly.



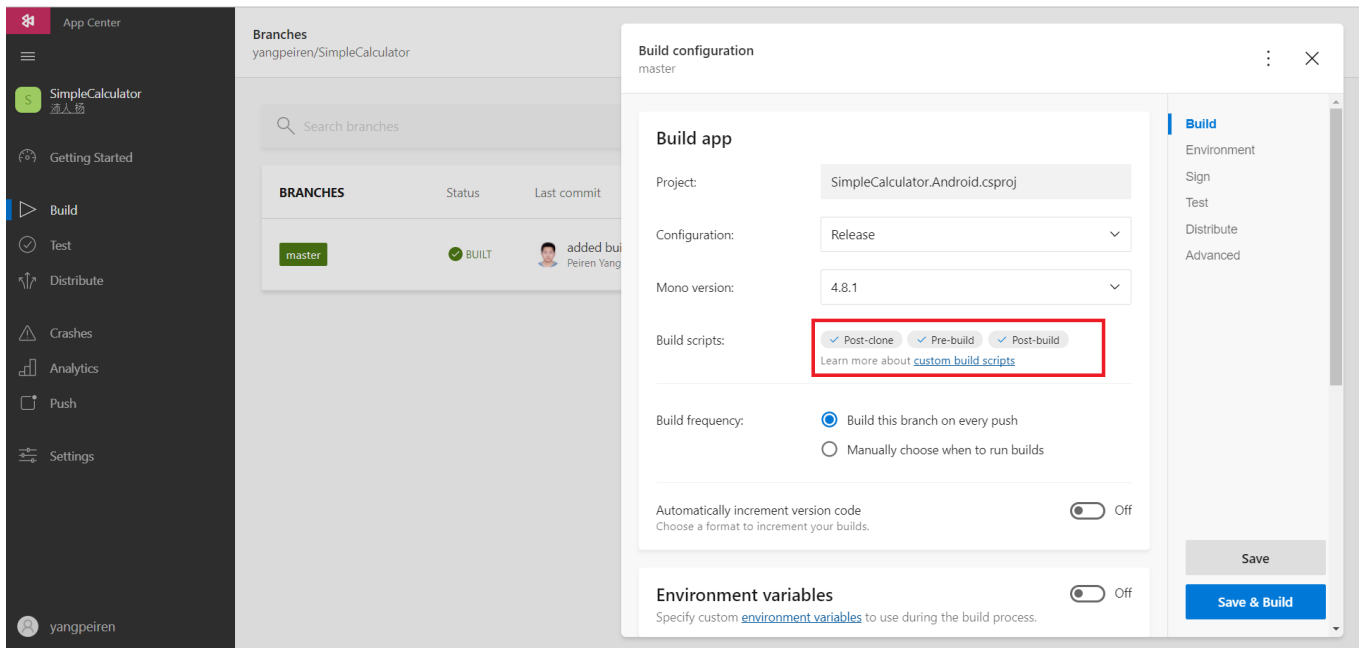
## 4.3 Using customized scripts in App Center

There are three different kinds of build scripts, namely post clone script, pre-build script and post-build scripts. They could be used for customization of the build process, test after build and many other operations.

To let the App Center recognize them, the scripts should have specified file names and be placed next to the project-level (.xcodeproj, .csproj, .sln). For Android and iOS platform, the scripts are written in Linux bash, for windows UWP they are written in PowerShell. Please reference [the sample Android project](#) to see the valid naming of the scripts and the storage position of them in regard to the project root file.

As shown below, all conditions mentioned above are fulfilled, the scripts is recognized by the App Center correctly.





To protect the background server against misuse, the absolute path is disabled in the scripts. For example, the root folder of the project can be addressed by `$APPCENTER_SOURCE_DIRECTORY\SimpleCalculator` instead of `\home\username\projects\SimpleCalculator`. Please change the commands in the scripts accordingly after the local test. For all variables inside App Center please reference [here](#).

## 4.4 Start tests automatically after build process

As discussed in chapter 4.1 and 4.2, unit tests and UI tests can be triggered manually. With the knowledge of chapter 4.3, they can also be triggered automatically in the post-build script in a comforting way.

To build the unit test projects and execute them, the following commands should be added into the post-build script:

```
1. # build unit test project
2. $ msbuild $APPCENTER_SOURCE_DIRECTORY\SimpleCalculator\SimpleCalculator
   .UnitTest\SimpleCalculator.UnitTest.csproj
3. # run the unit test
4. $ nunit3-console
   $APPCENTER_SOURCE_DIRECTORY\SimpleCalculator\SimpleCalculator.UnitTest\
   bin\Release\SimpleCalculator.UnitTest.dll
```

```
5. # display result
6. $ find . -name 'TestResult.xml' -exec cat {} \;
```

To trigger UI test directly after build (possible build of UI test project is needed), the following command is needed:

```
1. $ appcenter test run uitest --app "yangpeiren/SimpleCalculator" --device
es 87da775d --app-path
"$APPCENTER_OUTPUT_DIRECTORY\com.companyname.SimpleCalculator.apk" --t
est-series "master" --locale "en_US" --build-dir
"$APPCENTER_OUTPUT_DIRECTORY\SimpleCalculator.UITest\bin\Release" --tok
en $AppCenterLoginForAutomatedUITests
```

## 4.5 Cost and alternative solutions

Visual Studio App Center has a "device farm" in its background, with which tests can be done on various devices (more than 2000 devices as officially claimed) with various OS versions. The basic cost of this service is [\\$99 per month](#) for one concurrency.

One restriction of the UI test in App Center is that one submitted test can only run **maximal 360 minutes**, test cases should be broken up into batches for a longer test.

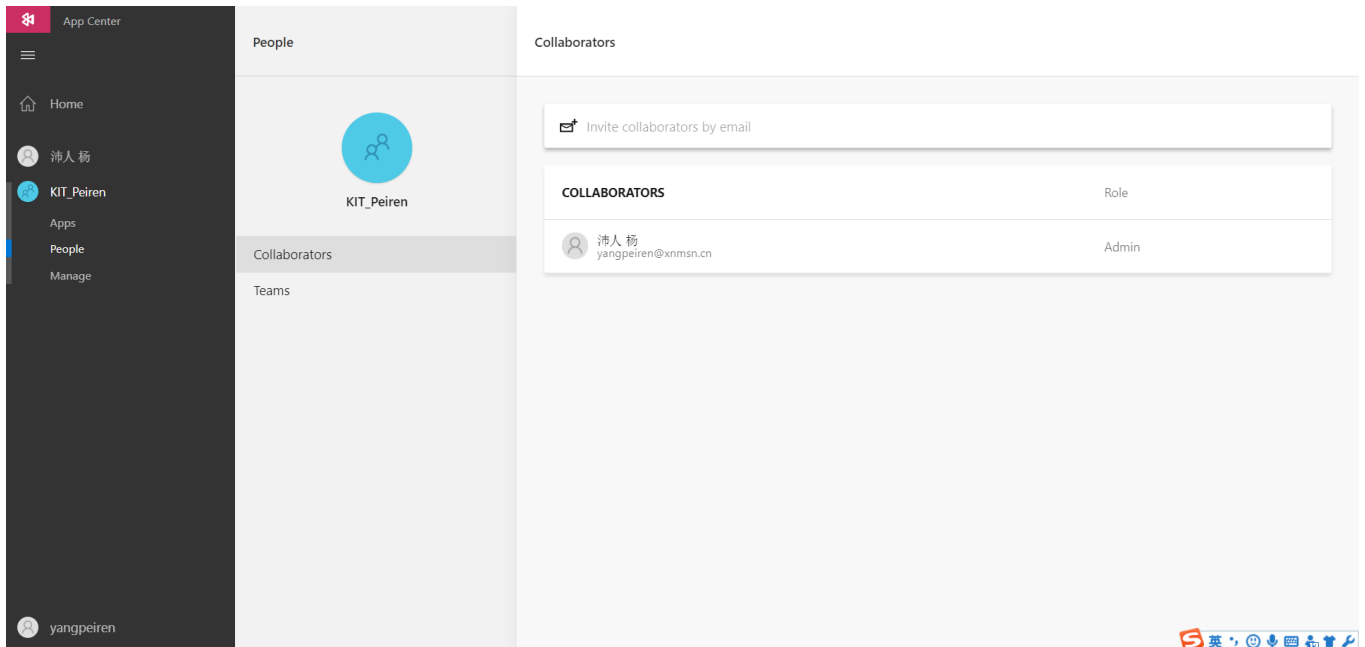
Amazon offers similar service named [AWS device farm](#), the price is [\\$250 per month](#) without test limits.

As an alternative, local CI tools like Jenkins with physical mobile devices connected or installed emulators support tests either. Compare to the cloud solutions, they have no cost advantage:

- Mobile devices are expensive, to upgrade or downgrade their OS version cost time, which is not flexible
- Emulators are often very large, one Android emulator costs more than 10G space to store and 1G RAM while running.

## 5. Team Collaboration in App Center

Collaboration is an important feature of Continuous Integration. Visual Studio App Center supports this feature like other local CI tools. Email as notification can be also configured. Below is a screenshot of this function.

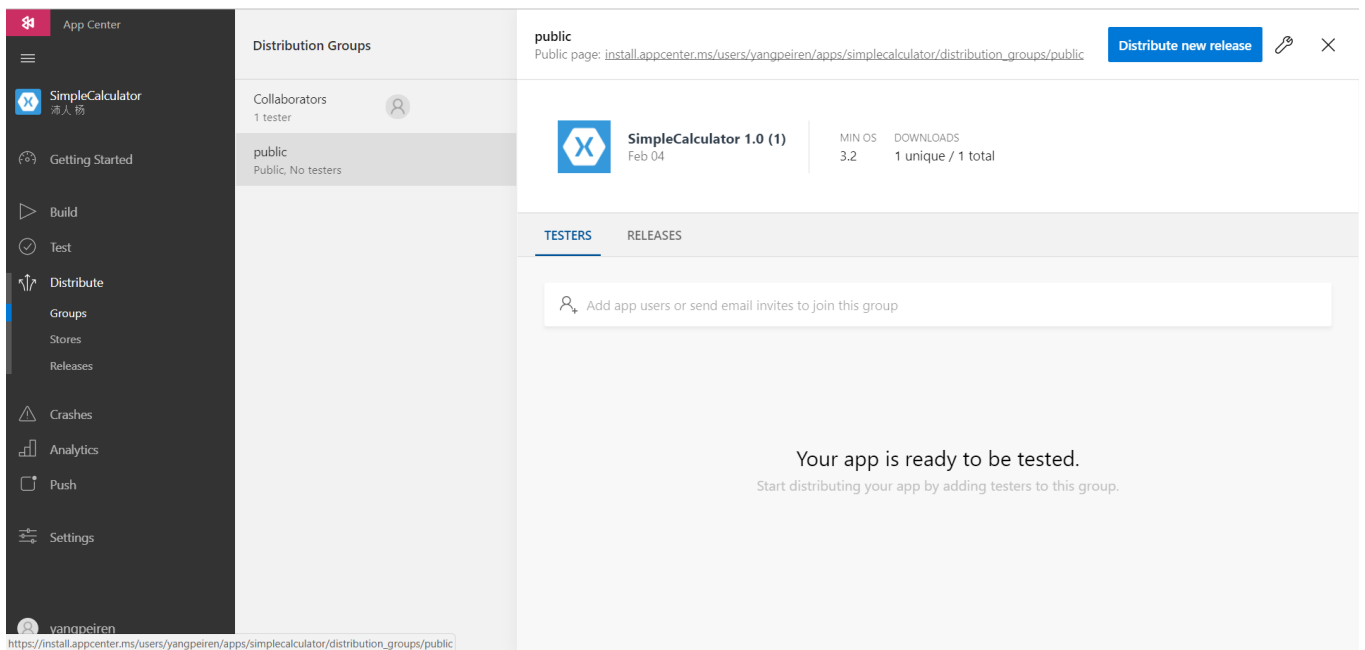


In this function, the 30 days of trial is not summed up with the personal account, which is very useful if the trial for individual use expired.

## 6. Publish the App and manage App lifetime

### 6.1 App publish

A signed successfully built app can be then published for download and install. Below is the publish management of the sample project in App Center.



Apps can be published to:

- Target group of users, such as beta testers or company internal use
- Public groups, the built sample project is downloadable [here](#) as an example
- App Stores such as Google Play and Apple's App Store

## 6.2 App lifetime management

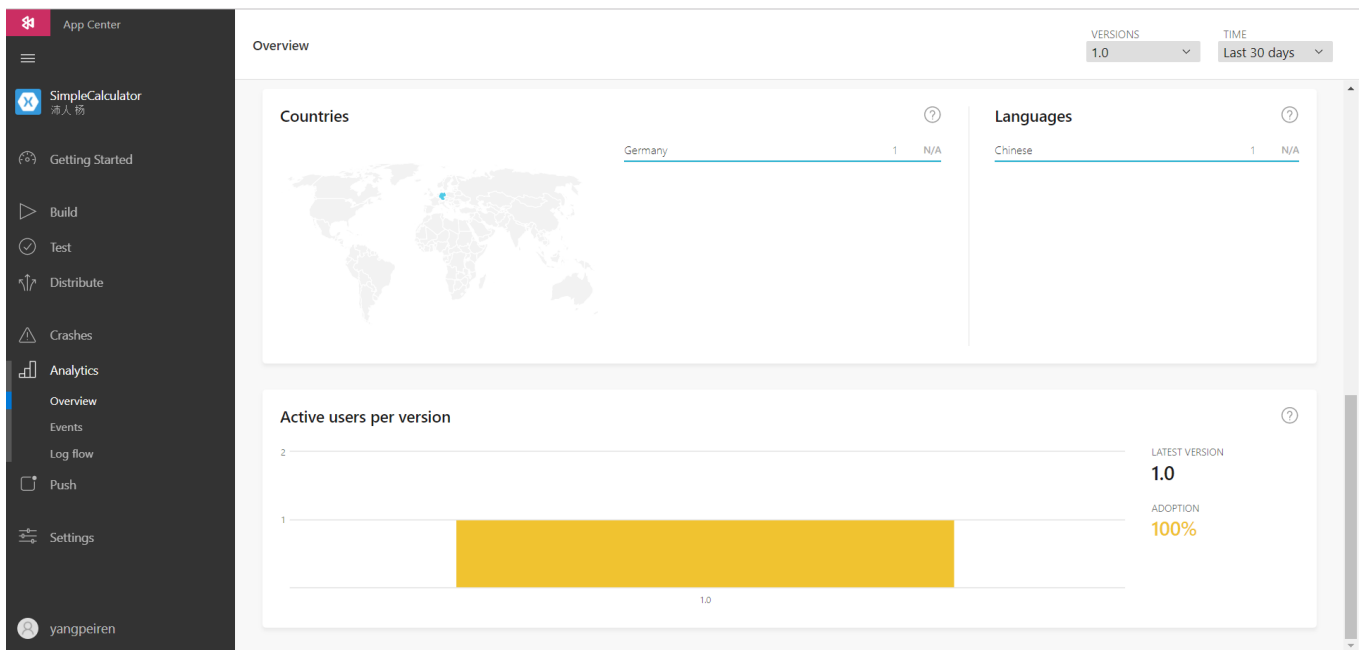
The Visual Studio App Center offers not only management of the project itself but also the management of the apps after publishing. The management includes:

- Crash analysis

All crashes will be collected and grouped by device types as well as app versions, which statistically facilitates the troubleshooting (reference of the crash SDK is needed).

- Analytics of users

Each download of the app will be recorded by App Center, the actual users can be easily analyzed, which simplified continuous improvements.



- Push notifications to target user

By referencing the push SDK from App Center, notifications can be pushed down to target groups in terms of their devices, their location, their app versions and other features.

## 7. Summary

Visual Studio App Center is a powerful tool that particularly suitable for cross-platform app development since it offers project build in different platforms and supports app tests with an enormous device farm. In addition, it offers interfaces for the management of app lifetime.

The total price for using App Center as build and test platform is basically **\$139 per month**, which is apparently expensive. However, compared to the CI tools like Jenkins, acquisition expenses for the physical server and test devices are avoided. With more build and test flexibility, the cost performance of using App Center is yet excellent.

One disadvantage of using App Center is that the job customization is not quite flexible, only three scripts can be configured and they must be executed consecutively, while CI tools like Jenkins can be configured freely with arbitrary many scripts and arbitrary execution sequence.

Since App Center is a new product of Microsoft, until now not all claimed features are available, [here](#) is a roadmap which lists all features that will come in the next three to six months.

As alternative, Jenkins is a powerful CI tool which supports building and testing, here is a [setup instruction](#) form Xamarin official documentation for cross-platform app development.

Jenkins could also act as an assistant tool to Visual Studio App Center, in which Jenkins takes over all customized scripts (the disadvantage of using App Center is thus eliminated) and works as an agent between several systems, one possible work schema is demonstrated with the figure below:

