7

Configuring Team City for CICD with Xamarin

In the old days, application development used to happen in separate not so integrated teams and developers were not used to merge their work with other developers’ code for quite long time and that used to create merge issues. Things that were working once on the developer’s local copy, used to stop working when merged with others’ code. This kind of not so integrated development environment increases the development time and delayed discovery of issues. Solution for this is to have continuous integration build into the development cycle where developers are merging their code multiple times a day and getting issues fixed at earlier stages.

In this chapter we will learn more about Continuous Integration and Continuous Delivery and different tools that we can use for the same for a better development integration and delivery process.

In this chapter we’ll be covering following topics:

* Introduction to continuous integration
* Various tools for continuous integration
* Using TeamCity with Xamarin
* Preparing build server for TeamCity and installing TeamCity
* Creating build script
* Creating a TeamCity project

…

# Introduction to Continuous Integration

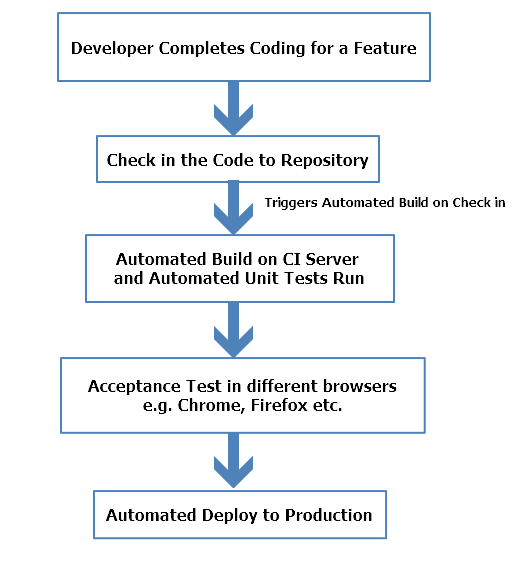
Continuous Integration (CI) is a development and integration practice in which developers checking code into a shared repository frequently, preferably several times a day. Each code merge can then be verified by an automated build and automated tests if applicable.

There are many benefits of following continuous integration; one of the advantages is that it helps detect defects quickly and at early stage. The check-ins are usually very small and contains small portions of developments, thus helping in identifying the exact issues quickly.

Continuous Delivery (CD) in the other hand is the process after integration and as the name suggests, it is to make sure that the code base checked in is deployable at any point of time. Each environment from test to production can and mostly do have different configuration. Continuous Delivery makes sure that all configurations are always ready for the deployment to any environment and that the code passes all the tests necessary for release.

In short, Continuous integration improves the development and testing experience with frequent code merges and helps in quickly identifying the bugs and involves running automated tests if included in the process while Continuous delivery makes sure the codebase everything is in ready state for the code to be deployed in any environment.

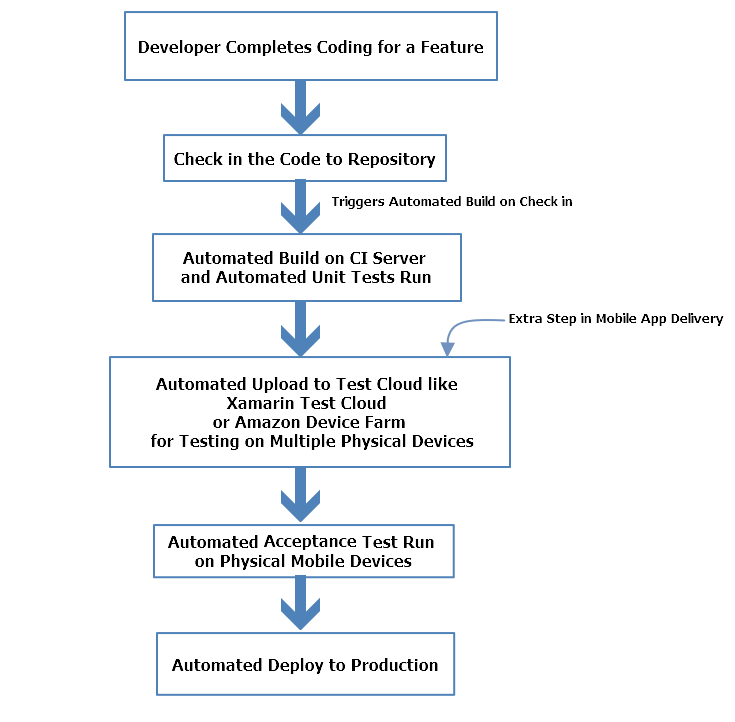
## CICD for a Web Application



B07738\_07\_01

For a web application once the build is ready from the CI server it is not a big task to test the application on different browsers since there are only limited number of them. But in a mobile application there is one more step involved to improve the experience since there are thousands of devices with different versions of operating systems available.

## CICD for a Mobile Application



B07738\_07\_02

As shown in the image above, mobile applications need to be tested on hundreds of devices with different operating systems and purchasing on those mobile devices which keep on coming every now and then, can be very expensive. To make sure the quality of application stays at the top including Test Cloud based solutions becomes an integral part of the process.

## Choosing Tools for Continuous Integration

There are many CI tools available in the market to implement continuous integration, just like there are many languages available in the market to developer applications. But choosing the right CI tool is very important for a good long-term benefit.

Choosing a CI tool for your project can depend on many variables like:

* Programming Language Support: This is one of the most important factors while choosing a CI tool. Some CI tools have better support for certain language specific builds and packages while others might not provide language specific packaging options.
* Operating System: Operating system support, meaning some teams might find an open source operating system like Linux to be a better choice for all their servers including the CI server and it might be more comfortable for them to configure a familiar operating system while other teams working on .NET applications might find Windows to be more comfortable and feature rich for their configurations. It all boils down to the preference different teams have and the kind of application they are working on.
* Integration with Code Repository: Different teams prefer different code repositories for various reasons, some might find Git to be more feature rich and supported on various IDEs with plugins while others who are more familiar with Microsoft environments find Team Services to be easier to use and better integrated. Different CI tools have different level of support for these repositories.
* Support for Application Platform Deployment: Some CI servers are better suited for web application deployment while other provide more features and better support for mobile application deployment to app stores. Depending on your type of application, the choice can vary.
* Cost: Cost is always an important factor while choosing any type of tool. Medium to big size companies can afford to have expensive feature rich CI tools while smaller companies and teams might want to stick to low budget and sometimes open source and freely available CI tools and customize them according to their needs.

## Various Tools for Continuous Integration

Let’s have a look at some of the widely used CI tools available in the market.

### TeamCity

TeamCity is a well known CI server, built by JetBrains. JetBrains is quite well known for developing various tools for different phases in software development life cycle like WebStorm and ReSharper. Team has both a licenced version and a free version with limited number of configurations and build agents, the free version is suitable for small teams that plan to grow over the time.

Despite being Java based solution, TeamCity offers the best .NET support among the tools on this list. There are also different enterprise packages, that scale by the number of agents needed.

You will be learning more about TeamCity later in the chapter.

Key Features:

* Extensive support for .Net based applications and Visual Studio
* Remote run, which can be used to test changes for failures without doing an actual commit
* TeamCity supports automated and manual both types of build triggers, you can configure automated build triggers for every commit

### Jenkins

Jenkins is one of the most popular open-source projects for continuous integration. With thousands of plugins to choose from, Jenkins can help teams to automate any task that would otherwise put a time-consuming strain on your software team. Common uses include building projects, running tests, bug detection, code analysis, and project deployment.

Key Features:

* Jenkins has an easy installation process by just running a command “java -jar jenkins.war” and deploy, nothing else
* Jenkins comes with a user friendly web interface and you can configure Jenkins entirely from that
* Jenkins has a huge plugin library and integrates with most of the build tools
* Customizing Jenkins to your project need is very straight forward by creating plugins and extending its capabilities
* Distributed builds are supported by Jenkins over different servers and even with different operating systems.

### Visual Studio Team Service

Visual Studio Team Services provided by Microsoft helps teams plan better, code together, and ship faster. You can code in any IDE and language, for any target platform. Various tools and plugins can be downloaded to customize it to your project requirements.

Key Features:

* Supports wide variety of tools including Visual Studio, Eclipse or any other tool available.
* Comes with unlimited free, private repos (including Git repos)
* Planning boards and tools available for Agile and even Kanban projects
* Automatically compile and test apps in cloud to avoid build failures

### Bamboo

Bamboo is a CI server being used by software teams worldwide to automate the process of release management for applications and general software, it allows teams to establish a streamlined pipeline of build delivery. Mobile developers can deploy their apps back to the Apple Store or Google Play automatically. Being the Atlassian tool, it has the native support for JIRA and BitBucket and you can even import your Jenkins configurations into the Bamboo easily.

Key Features:

* Unlike Jenkins, Bamboo has built-in Git branching workflows
* Because it is built by Atlassian, it has built-in integration for Jira and BitBucket
* Bamboo also supports automated merging to avoid conflicts and difference in working branch and master branch
* Test Automation in Bamboo makes a continuous flow from build to test and goes to even releasing the application to customer when ready
* Built in support for Jira makes bug tracking in the specific release and even builds automated and easily trackable

# Using TeamCity with Xamarin for CICD

As mentioned in previous section, TeamCity provides great support for .NET based applications.

While it can automatically detect build steps from the configuration files and project files, it can also detect automatic build triggers from GitHub.

## Requirements for using TeamCity

For using TeamCity, knowledge and availability of some hardware and technologies is required to make the setup process smoother.

* A dedicated build server for TeamCity Installation and setup: Ideally the build server should be a standalone server and should not be responsible for other responsibilities like, DB server or hosting server etc.
* Knowledge of MSBuild: Having knowledge of MSBuild can make this setup much better and would help in resolving any compilation related issues if required.
* Knowledge of Xamarin Test Cloud for Continuous Testing: Xamarin Test Cloud will be used in this chapter for continuous testing after a build and application package is ready. You would be familiar with this because it has been described in detail in the last chapter.

## Steps Involved in TeamCity Setup

Following steps are involved in setting up TeamCity

1. **Preparing Build Server:**In order to build our mobile app on the build server, there are some software that need to be installed on the build server that will be used while building the application.
2. **Creating Build Script:**A build script is basically a script containing set of commands to perform various actions in the build process, like compiling the application, building the apk and then submitting it to the cloud for testing etc.
3. **Installing TeamCity:**Once we have the required tools installed on the CI server, TeamCity needs to be installed and configured for the project and it’s users to run build scripts
4. **Creating TeamCity Project:**Once we have all the software required to build project and the script to perform build, a TeamCity project should be created.

# Preparing Build Server

In order to compile and build mobile app on the server, some software need to be installed that are required for the build process. For an Android application to be built on the build server it is important to have tools like Visual Studio SDK and Visual Studio build tools installed on the server. Also, acquiring Android Keystores is required to sign the application package to be released later. To avoid any configuration issues, it is recommended to install these software under the same user account as TeamCity.

## Firewall Configuration

For continuous testing, we are using Xamarin Test Cloud and it was described in the previous chapter in detail. For tests to be submitted to Xamarin Test Cloud automatically as part of CI, the CI server must be configured to allow network traffic to and from testcloud.xamarin.com on ports 80 and 443.

Once these configurations are done and firewall is configured to allow the communication between server and Xamarin Test Cloud, we will be able to use command line tools in the build steps to submit our UITests to Xamarin Test Cloud.

## Installing Visual Studio with Xamarin

To install Visual Studio with Xamarin, you can follow the same steps described in Chapter 3, “Cross Platform mobile app development with Xamarin”.

Following the steps should install both Visual studio and Xamarin with required tools and required SDKs.

## Android Keystores

Android Keystore is used for signing the application while distributing the application. This is required while before packaging the application so that our final package is singed with it.

#### Creating your own keystore

The first step is to create your own personal keystore that will contain the information used to digitally sign your Android package files. You can do this with the following command:

"C:\Program Files (x86)\Java\jre1.8.0\_45\bin\keytool.exe" -genkey -v -keystore youFileName.keystore" -alias your\_alias\_for\_keystore -keyalg RSA -keysize 2048 -validity 30000

The 30000 at the end of the command denote the length of validity of the certificates and Google require this to be past 2033.

Before you run this command make a note a few parameters first. When you run the command, it will ask you the following parameters to enter. These parameters will be used again later in the project file.

Password - <yourpassword>

Name - <yourname>

OU - <organisationunit> eg: JamSoft

Orgname - <organisationame>

Local - <locality>

State - <state>

Country - <2lettercountrycode>

Running the command should generate a .keystore file with the filename provided in the command. Now that our build server is ready, let’s prepare the build script that we’ll be using the build process.

# Creating Build Script

The build script should contain following steps:

1. Compile the Application:  
   Configuring the application project file to use proper keystore and compiling the application using Visual Studio SDK tools.
2. Submit the Application to Xamarin Test Cloud: Once the server’s firewall is configured to allow communication with Test Cloud servers as mentioned in previous steps, this step in the build script would contain the command to upload the signed application package to Test Cloud servers.

## Compile the Application

Now that we have our android keystore ready and prepped for use we can look at the Visual Studio project. In order to make this automated in the build system we need to configure the project to use our keystore credentials.

1. In Visual Studio edit the Android application .csproj file and add another PropertyGroup element as per the code below:

<PropertyGroup Condition="'$(Configuration)' == 'Release'">

<AndroidKeyStore>True</AndroidKeyStore>

<AndroidSigningKeyStore>myandroid.keystore</AndroidSigningKeyStore>

<AndroidSigningStorePass>yourpassword</AndroidSigningStorePass>

<AndroidSigningKeyAlias>myaliasdroidpub</AndroidSigningKeyAlias>

<AndroidSigningKeyPass>yourpassword</AndroidSigningKeyPass>

</PropertyGroup>

1. Now our .csproj file knows how to use our keystore unattended. We can tie into the Xamarin build process from within our automated builds and produce the base Android package.You can test that this is working using the following command:

msbuild.exe PhoneCallApp.csproj /p:Configuration=Release /t:Rebuild

This command uses MSBuild to build the application with the given configuration, in our case it should be release.

1. We have our application package now and we can apply the signing processes. To sign the package created in the previous step we need to execute the following command:

"C:\Program Files (x86)\Java\jdk1.7.0\_71\bin\jarsigner.exe" -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore youFileName.keystore -storepass yourpassword -keypass yourpassword -signedjar \bin\Release\packagename-signed.apk \bin\Release\packagename.apk your\_alias\_for\_keystore

This package is now digitally signed using your certificate from the keystore we made earlier.

1. Now that we have a signed package we can zip align this package and then publish this as an artifact of our TeamCity build process. This command makes use of the Android SDK zipalign.exe program. You’ll have to find where this is on your machine as there are many potential locations. The command you need will look something like this:

“C:\Users\<name>\AppData\Local\Android\android-sdk\build-tools\<version>\zipalign.exe” -f -v 4 packagename-signed.apk packagename-zipaligned.apk

1. Now it is time to upload our tests and android package to Xamarin Test Cloud to be UI tested. We had created Xamarin.UITest In previous chapter and it is assumed that you are aware of the process of creating and uploading the test to Xamarin test Cloud.
2. So include the following command to your build process to upload test to Test Cloud:

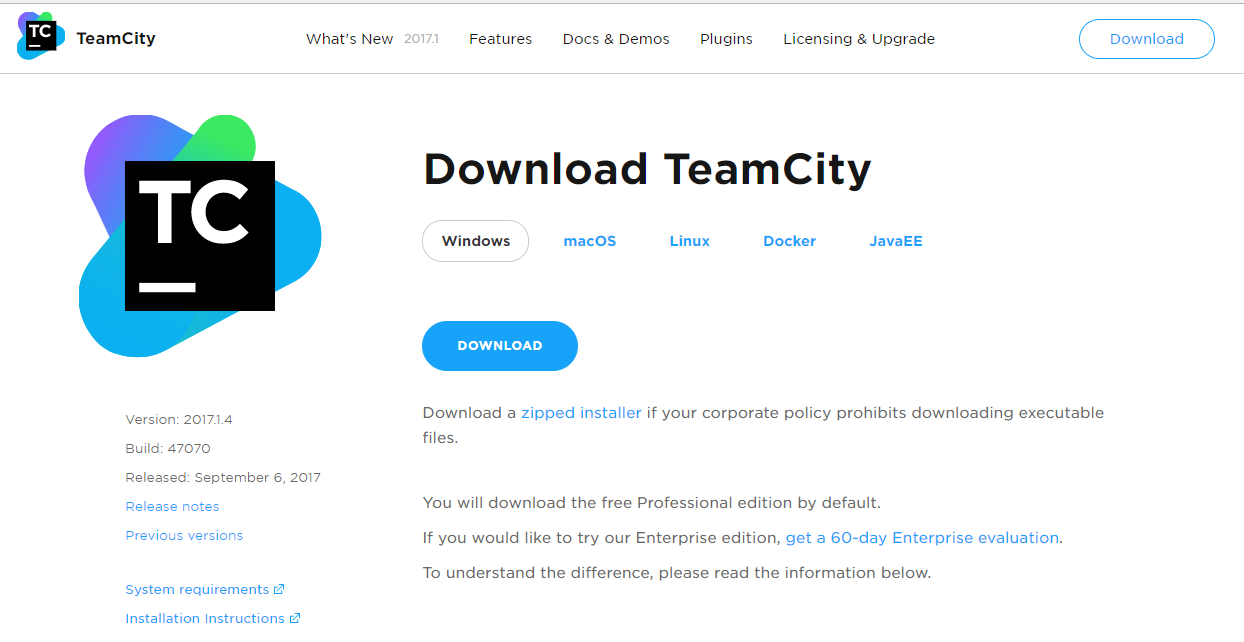
test-cloud.exe <path-to-apk-or-ipa-file> <test-cloud-team-api-key> --devices <device-selection-id> --assembly-dir <path-to-tests-containing-test-assemblies> --nunit-xml report.xml --user <email>

When the test is run, the test results will be returned in the form of a NUnit style XML file called report.xml. TeamCity will display the information in the Build Log.

# Installing and Configuring TeamCity

To install and configure TeamCity on Windows machine, follow below steps.

1. Go to URl https://www.jetbrains.com/teamcity/download/#section=windows and click on the download button to download TeamCity installation package from TeamCity website.



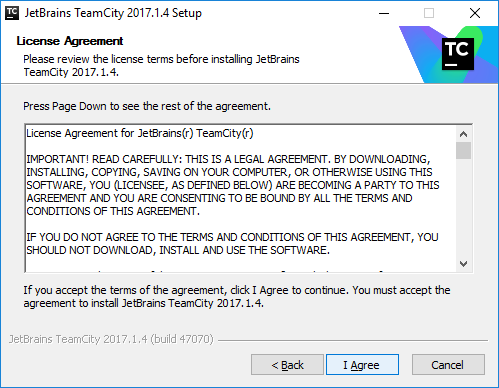
B07738\_07\_04

1. Once downloaded, open the installation package and click next.



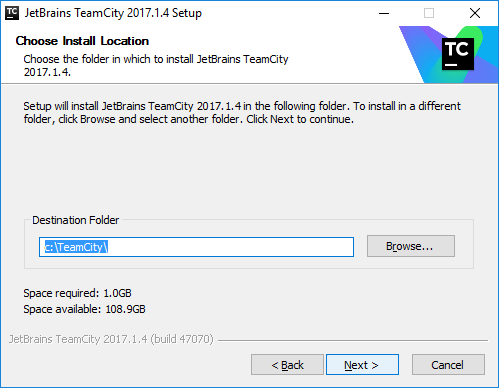
B07738\_07\_05

1. On the next screen, agree to the license and go to next step.



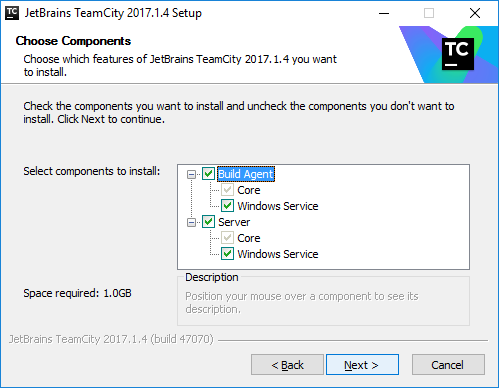
B07738\_07\_06

1. Select the path where to install TeamCity and click next.



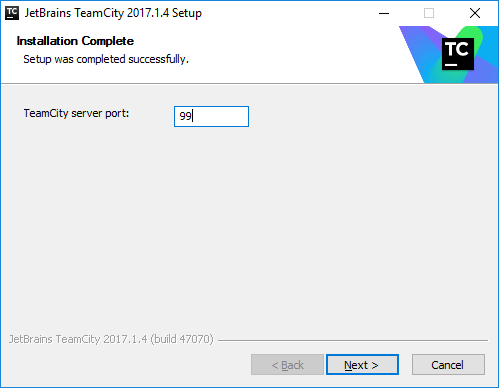
B07738\_07\_07

1. Select the packages to be installed, for example, if you are installing build agent and server on different servers then select accordingly. For learning purpose, you can select to install both on the same machine.



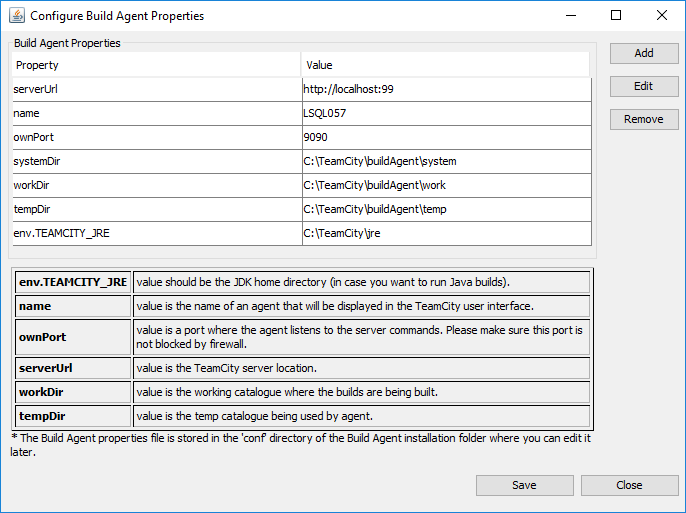
B07738\_07\_08

1. Once the installation is done, select the port where you would like TeamCity server to run on. Make sure this port is not used by other services on the machine and choose a unique port number and not the default one if possible.



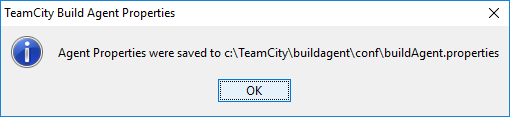
B07738\_07\_09

1. In the next screen, you’ll be able to see the configurations and ports configured for the server, you can also change them here if you want to.



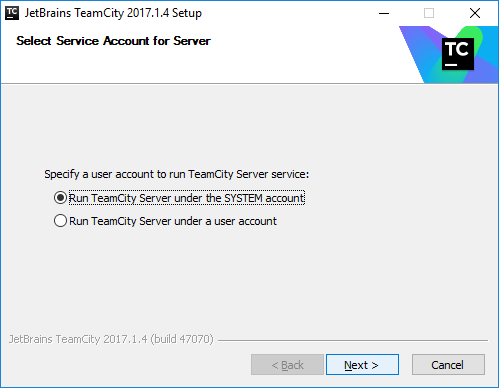
B07738\_07\_10

1. Click on save to save the configuration.



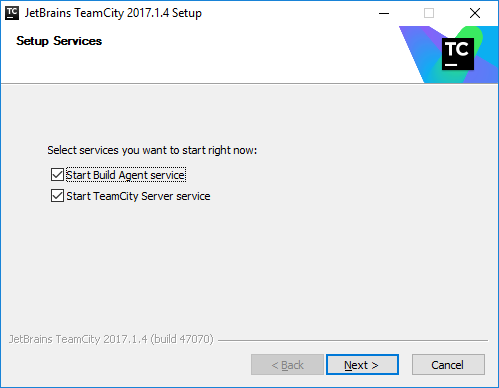
B07738\_07\_11

1. Select the account to run TeamCity.



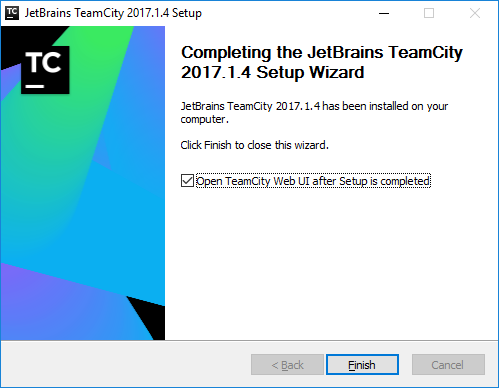
B07738\_07\_12

1. Start Build server and build agent.



B07738\_07\_13

1. Check Open Team City Web UI and click finish.



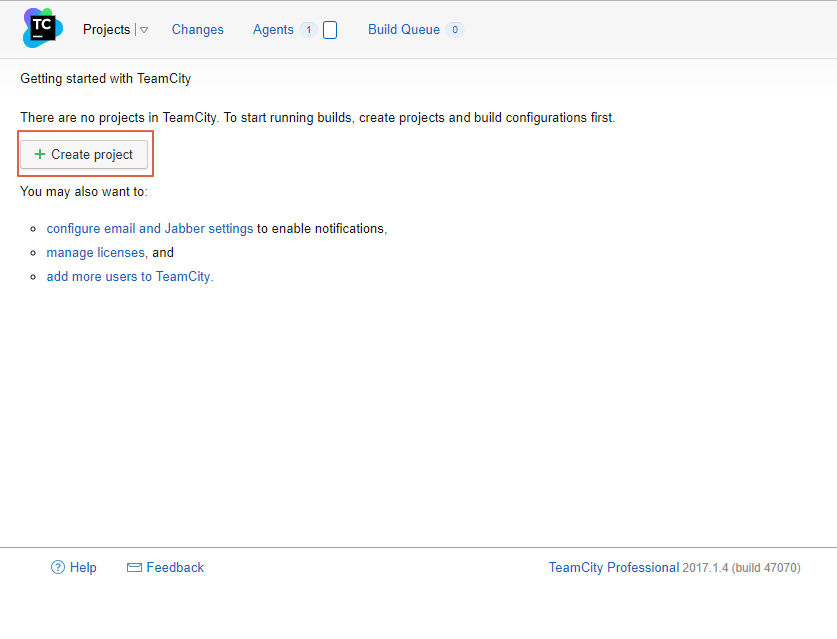
B07738\_07\_14

1. This will open TeamCity web UI where we can create TeamCity Project.

## Creating TeamCity Project

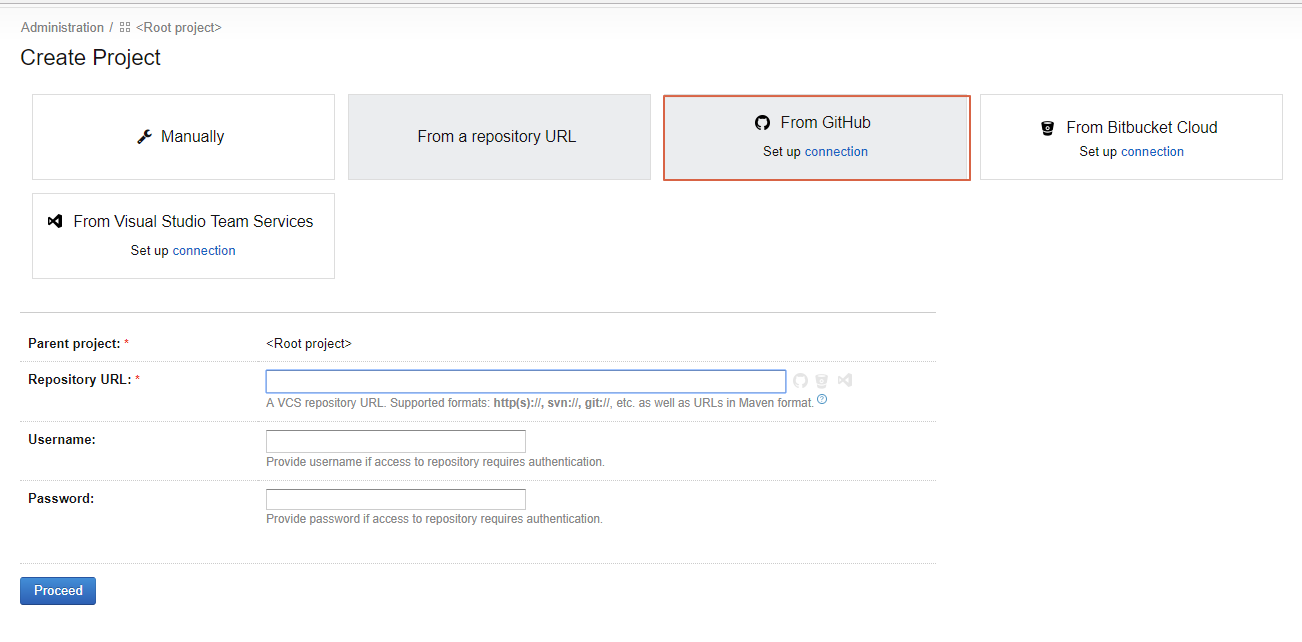
Once the installation is done, TeamCity web user interface will be opened in the browser and we can create new TeamCity Project there. To do so, follow below steps:

1. Once you have logged into TeamCity UI, Click on Create Project.



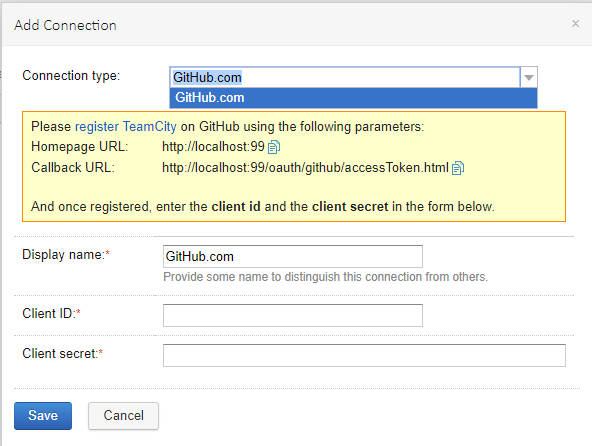
B07738\_07\_15

1. To connect our project from GitHub click on From GitHub on the next screen.



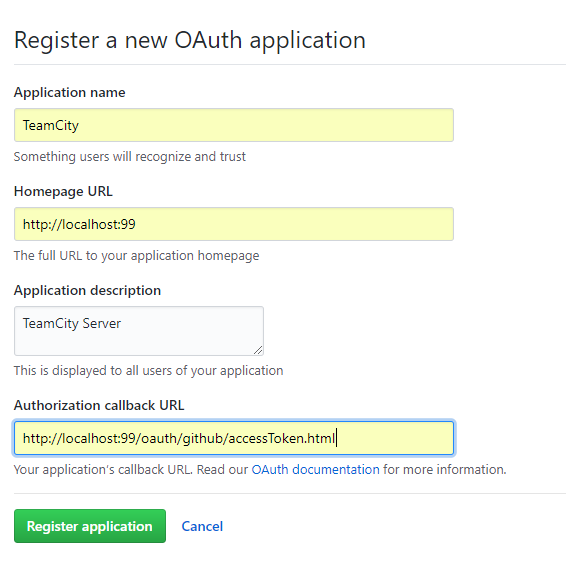
B07738\_07\_16

1. This will open a popup with some instructions to create add TeamCity application to your GitHub account.



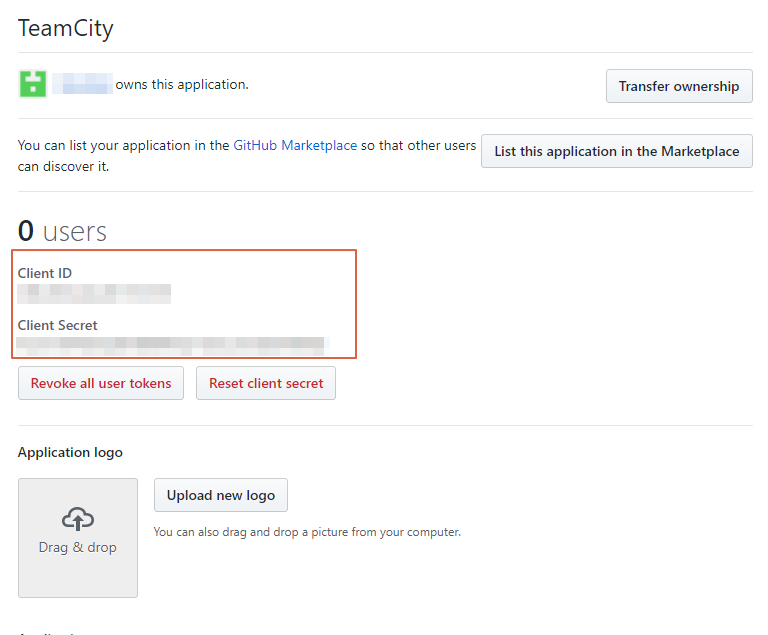
B07738\_07\_17

1. Click on register TeamCity link and it should take you to GitHub page where you can register a new OAuth app.
2. Give the details of the application and Homepage URl and Callback URL as shown in the image above (shown on the TeamCity screen) and register the OAuth app.



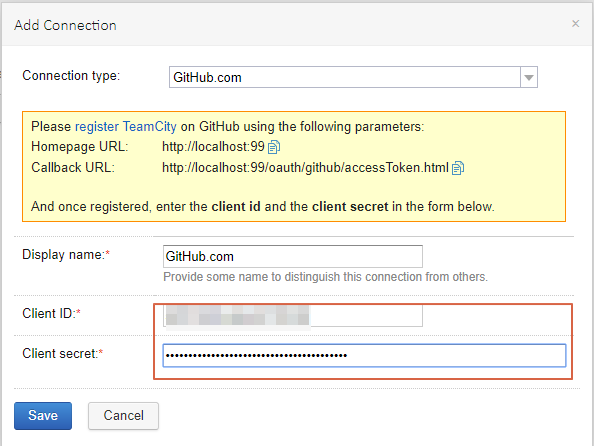
B07738\_07\_18

1. Once you register, on the next screen you’ll get Client ID and Client Secret, copy those details since it will be required on TeamCity project.



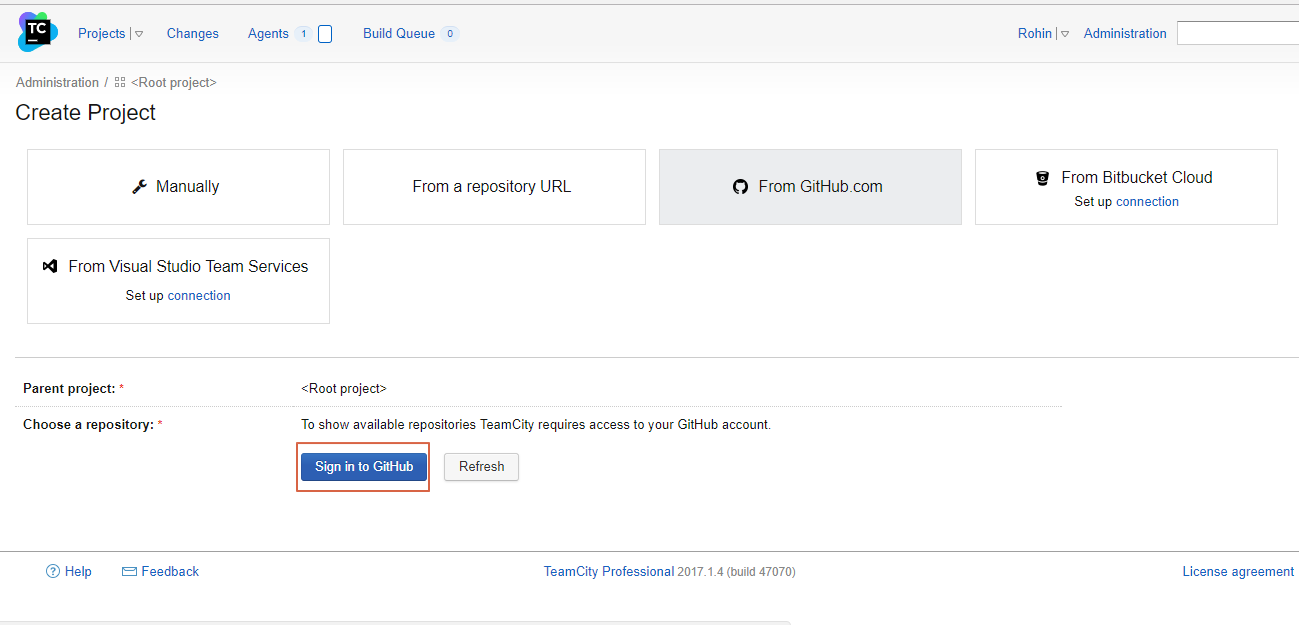
B07738\_07\_19

1. Come back to TeamCity and put Client ID and Client Secret in required fields and click Save.



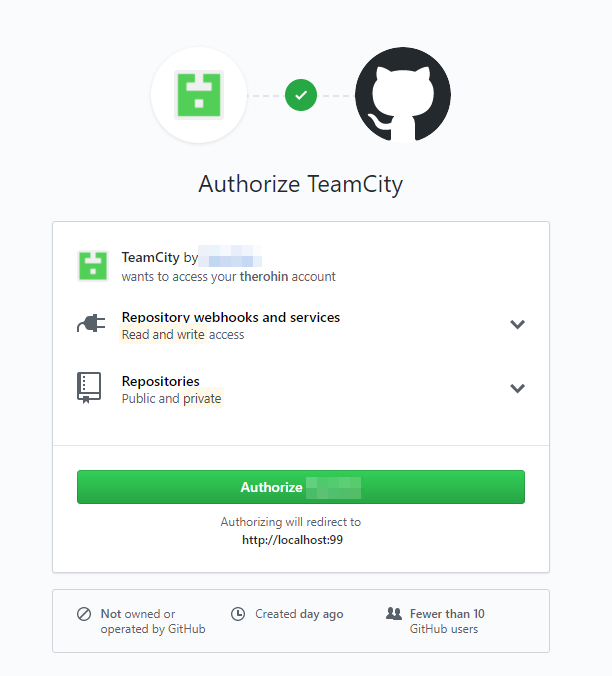
B07738\_07\_20

1. Next, you need to do a onetime sign in activity to allow TeamCity to use GitHub repositories. Click on Sign in to GitHub.



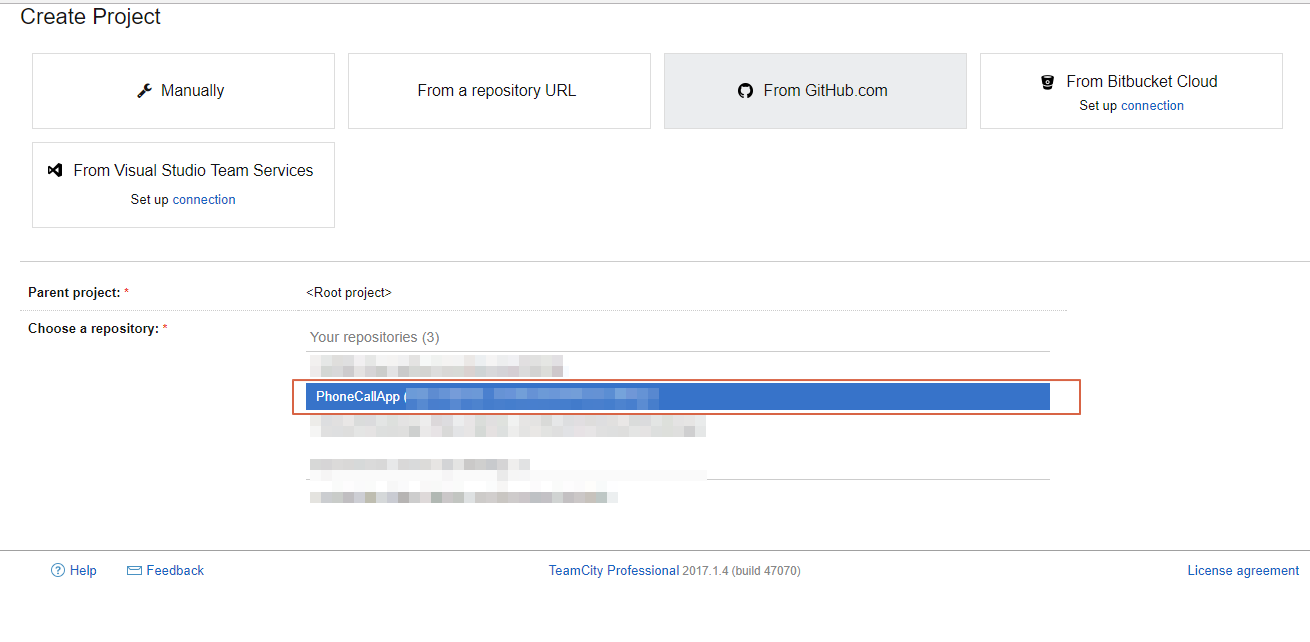
B07738\_07\_21

1. Authorize TeamCity App to use GitHub, click Authorize app.



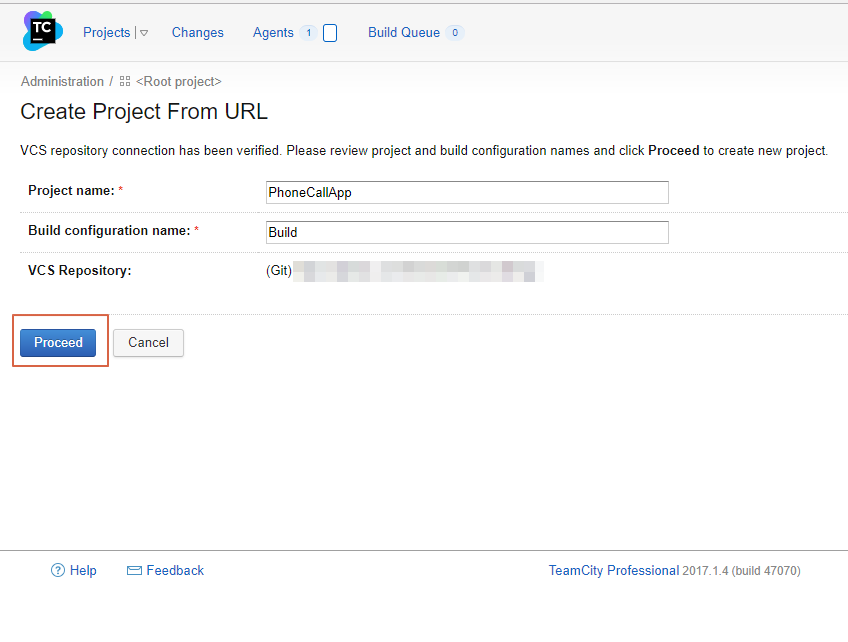
B07738\_07\_22

1. Once authorized, select the PhoneCallApp repository from the list of repositories shown on TeamCity.



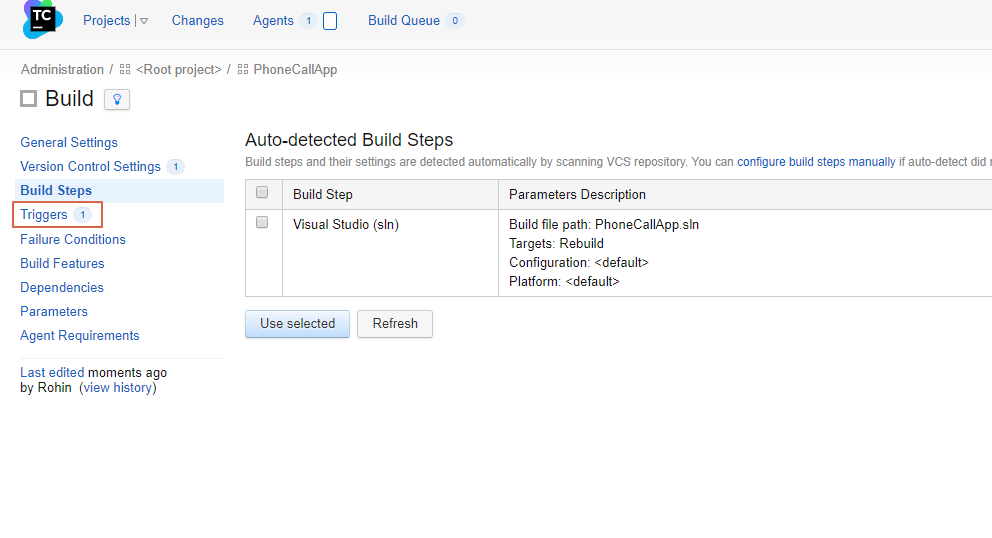
B07738\_07\_23

1. On the next screen, TeamCity would offer you to create a new project from the URL selected. Give it a name and click proceed.



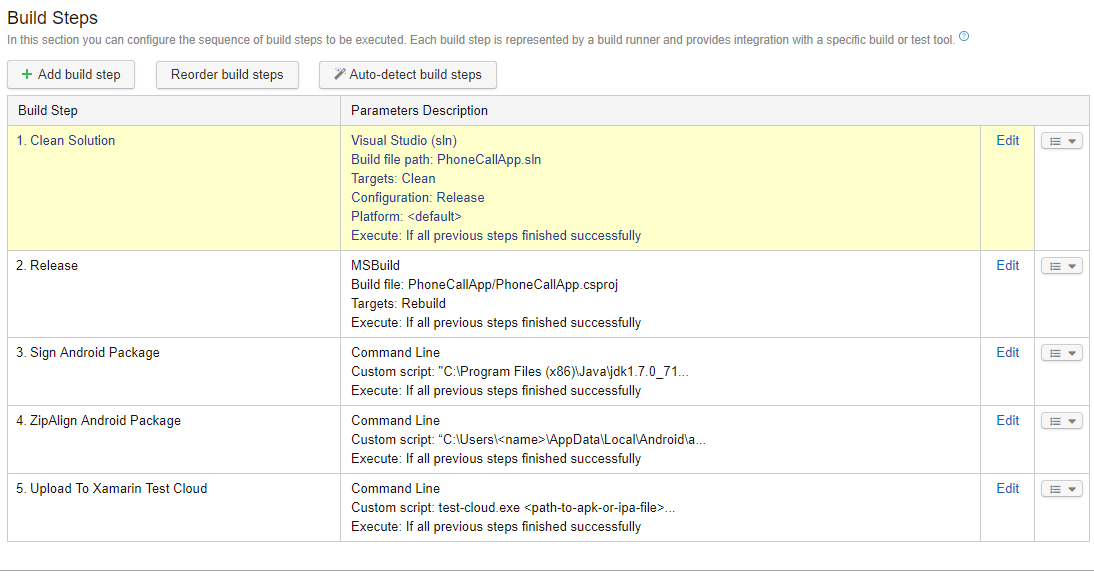
B07738\_07\_24

1. This should create two things, first would be that it creates a trigger in TeamCity for each code check-in you do, it will trigger the build. Second would be that it creates build step from the repository automatically.



B07738\_07\_25

1. We would need to configure the build steps manually and use the build scripts described in previous section Create Build Script. Use those scripts described step by step in previous steps to create Build Steps in TeamCity.
2. Finally, your build steps should look like below image, comprising of all the steps mentioned earlier in Create Build Script section.



B07738\_07\_26

1. Now your TeamCity continuous build is ready, and a trigger is already configured for perform this build on each code check-in or whenever it finds any code changes in the repository. This finally provides you with an Android package that is ready to be distributed.

# Summary

In this chapter, we learned about continuous integration and continuous delivery. We learned about various tools for continuous integration. We used TeamCity for implementing CICD with the Xamarin project developed in earlier chapters and learned how to create a project in TeamCity to automate the build process and finally get a distributable android package.

In the next chapter you’ll learn more about continuous distribution and delivery using Visual Studio Team Services.