YOUR GUIDE TO MOBILE DEVOPS

Microsoft

What is DevOps and why does it matter in the mobile world?

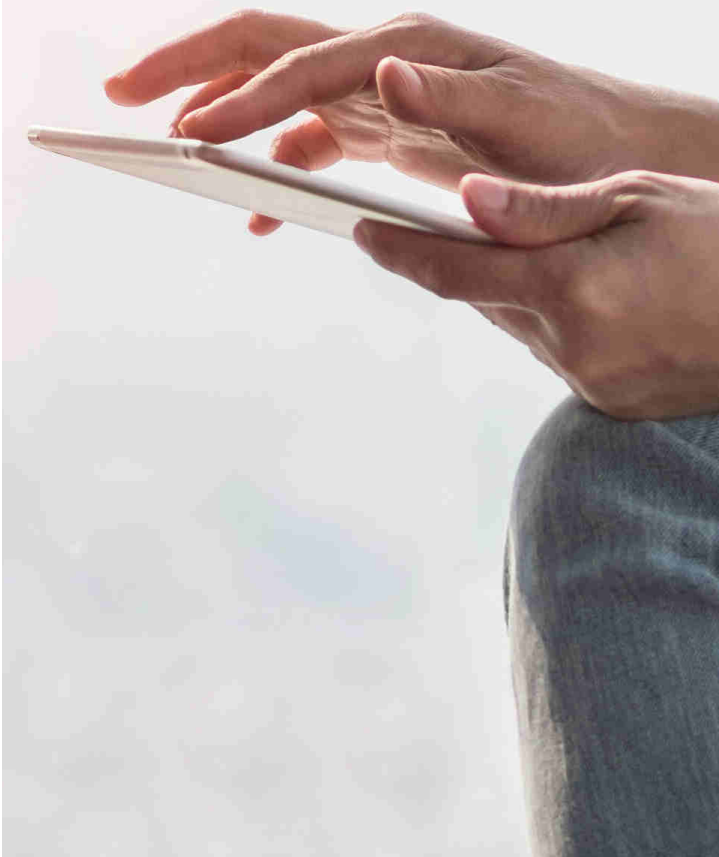
There are over two million apps[1] in each of the leading app stores today. User expectations are sky-high for consumer-facing apps, where a competing app is just a swipe away and reviews can make or break adoption. It's important to think of employees as consumers, too, as they expect the same high-performance, mobile-first user experiences in their work apps that they get from the consumer apps they engage with on a daily basis.

> **"DevOps provides a set of practices and cultural changes— supported by complementary tools— that automates the software delivery pipeline, enabling organizations to win, serve, and retain consumers better and faster than ever before."**
>
> FORRESTER®

DevOps for mobile apps is unique and more challenging than for other types of applications. The device operating systems are constantly changing, requiring updates to your apps that take advantage of the latest features. Device fragmentation, with hundreds of combinations of devices, operating systems, and screen sizes, makes it difficult to ensure that apps look, behave, and perform well for all users across all devices. Most teams think about getting apps to market first, but fail to consider the time and costs for maintaining apps and retaining users post launch.

40% of users will be less likely to come back to a mobile site or app if it doesn't satisfy their needs.[2]

29% of users will immediately switch to another app if an app fails to meet their needs.[2]



[1] Source: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] Source: Think with Google

For your apps to be successful, you need to think beyond just getting them out the door. You *must* have a strategy for each stage of the mobile lifecycle. When considering these milestones, there are some key questions to think about: how will you build apps for multiple platforms most efficiently? How will you test them to ensure they work on all of the devices your users will access them on? How will you know when crashes are happening for your users? How will you add new features without breaking existing productivity? How will you deploy bug fixes over the lifetime of the app?

To be a successful mobile developer, you need:

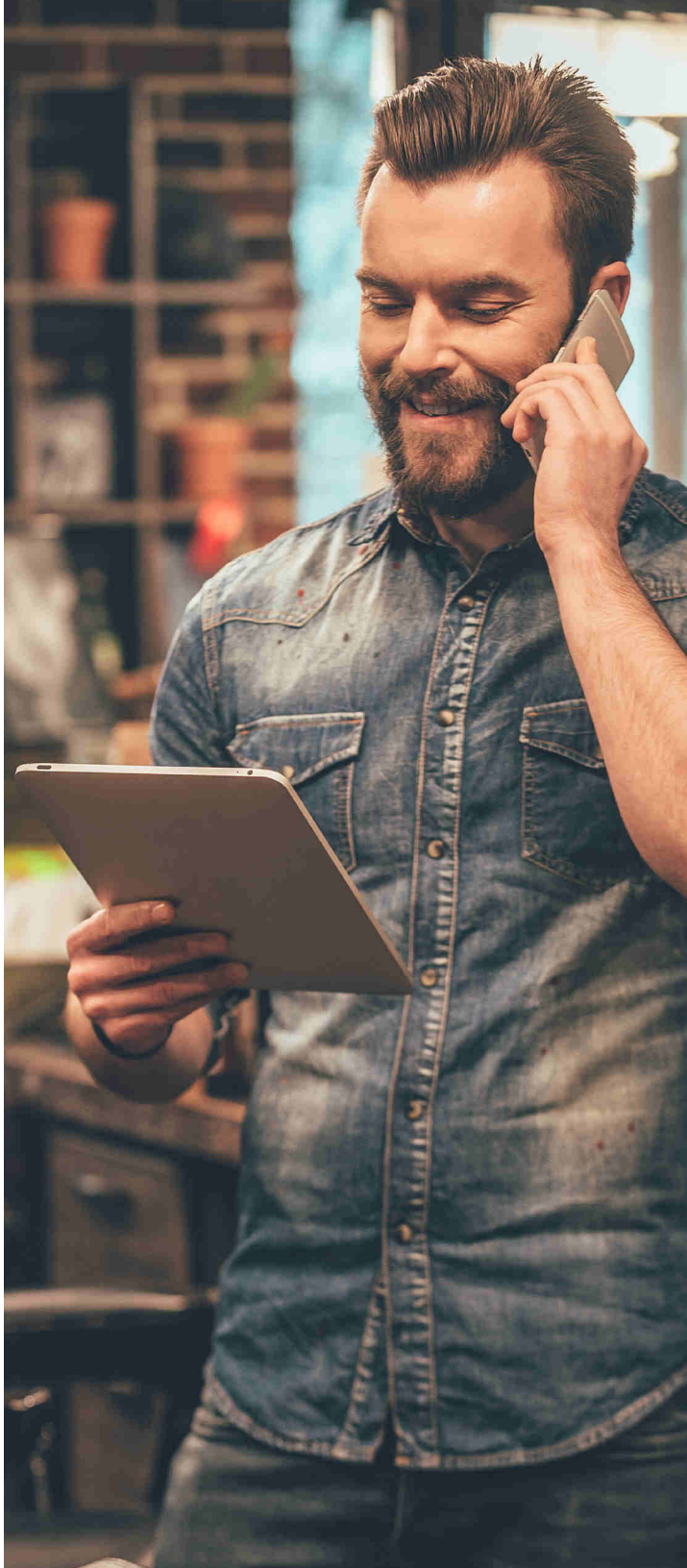
- Fast release cycles across a diverse device and platform landscape.
- A continuous feedback loop to address passive and active user feedback.
- The ability to scale your app with ease whether you have one or millions of users.

Mobile and the power of the cloud

The mobile DevOps lifecycle, along with a mobile-optimized cloud backend, is the key to fast release cycles and great user experiences.

With a mobile cloud backend, you're able to:

- Quickly add mobile-essential features such as push notifications, user authentication, and offline data sync.
- Securely mobilize on-premises and cloud data with ease.
- Automatically scale your app as your user base grows.
- Create tailored experiences for your users based on context and behavior.



How to use this guide

This guide uses our Shopping Demo App, a consumer-facing Android, iOS, and UWP app, to walk you through each stage of the mobile DevOps lifecycle, as well as how to connect mobile apps to Microsoft Azure's backend services.

The chapters below are organized in the ideal order of execution. Start with Part One and continue in sequential order through the mobile DevOps lifecycle, or select parts that are most relevant to you.

Part One: Plan and track your mobile projects

Part Two: Build native Android, iOS, and UWP apps using C#

Part Three: Create user-centric experiences with powerful cloud services

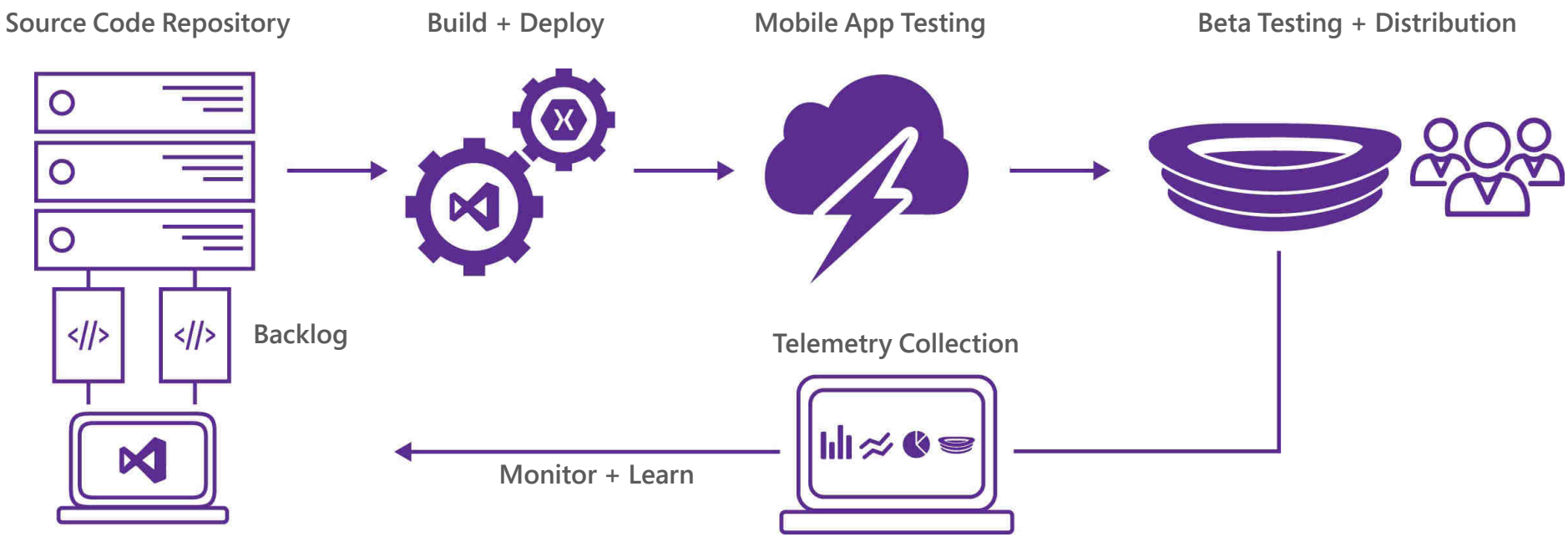
Part Four: Ship high quality apps faster with automated testing

Part Five: Get early user feedback and analytics

Part Six: Make each release better than your last with Continuous Integration

Part Seven: Continuously improve apps and make your users happy

Microsoft's unique mobile DevOps solution at a glance



Microsoft's mobile DevOps solution solves unique challenges at every stage of the mobile lifecycle. It simplifies mobile development by automating your entire release pipeline, from planning to Continuous Improvement. Accelerate your release cycles, deliver high quality apps, and spend more time on new features.

Part One:

Plan and track your mobile projects

Part One:

Plan and track your mobile projects

Any successful development project begins with a plan, and mobile development isn't any different. It's critical to orient all team members around the project's goals and benefits and to have complete visibility of project status and progress at any time.

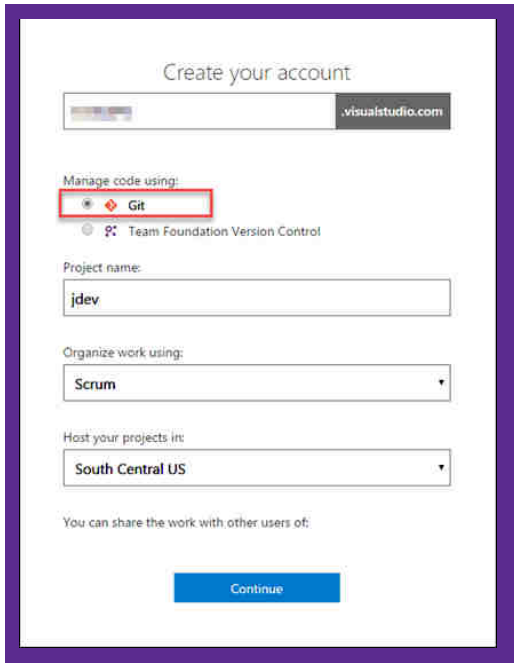
Visual Studio Team Services provides a set of cloud-powered collaboration tools that work with your existing IDE or editor (including Visual Studio, Eclipse, and Xcode). Plan and track your work, including builds and commits, and automate the build and deployment process. With dashboards and charts that update in real-time, get visibility into the things that matter, such as work items, owners, bugs, and your backlog. Spot issues before they get out of hand and keep your team focused.

Part One:

Plan and track your mobile projects

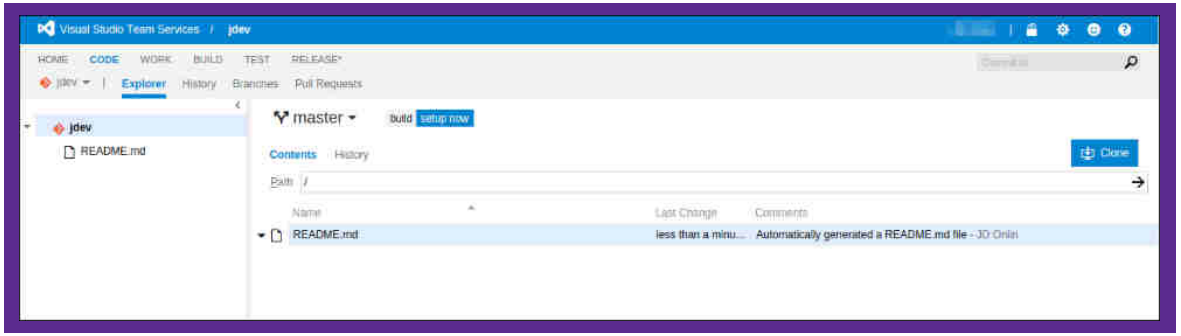
Creating a new project in Visual Studio Team Services

1. Log in to your Visual Studio Team Services account and start the **New Team Project** wizard. If you don't have a Visual Studio Team Services account, follow steps 1–10 here to set up your account.
2. Under the **Manage code using**: option select **Git** *(required)*. Enter **jdev** in the **Project name** field for your Team Project. If you use **jdev** while you are completing this demo, your screen will match the screen shots and instructions here. Under the **Organize work using**: option, select **Scrum** *(required)*. Select the appropriate region to host your projects, then click the **Continue** button.

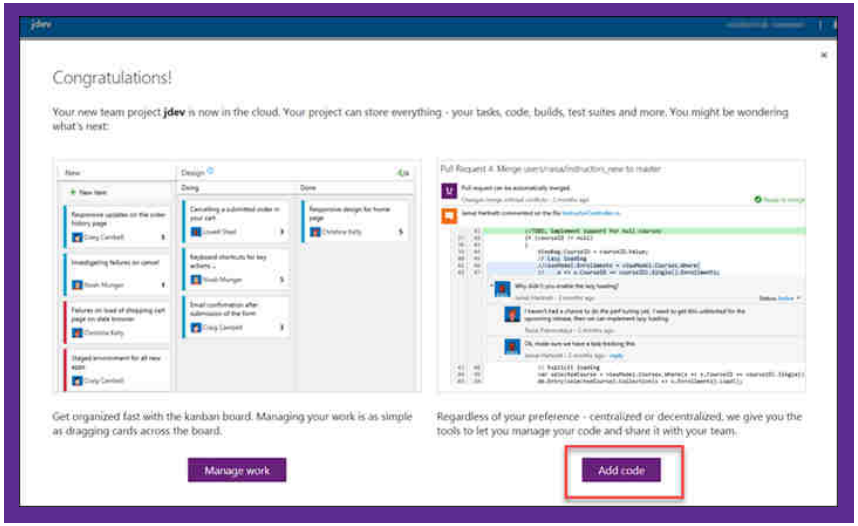


Note: If you have picked an account name in use, you will receive a warning. Pick a different name and try again. Creating a new account is fast and can take as little as a few seconds to complete. Visual Studio Team Services queues up a job to create your Team Project and again, in just a few seconds, you'll have a new Team Project.

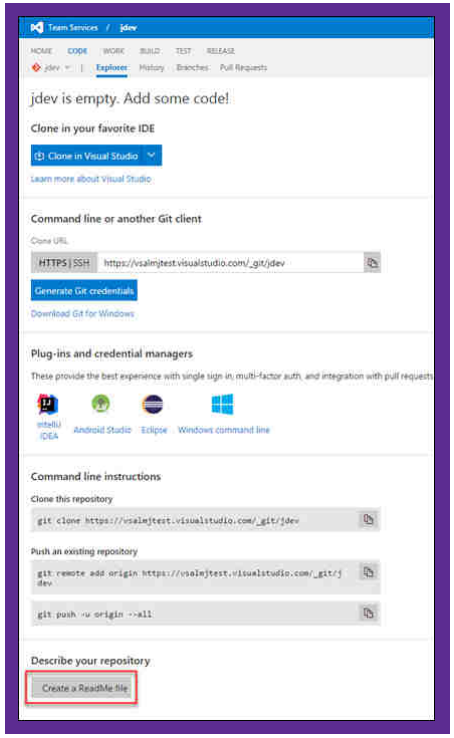
3. Once done, you should be in the CODE hub of your Team Project. Continue reading if you don't see a screen that looks like the one below.



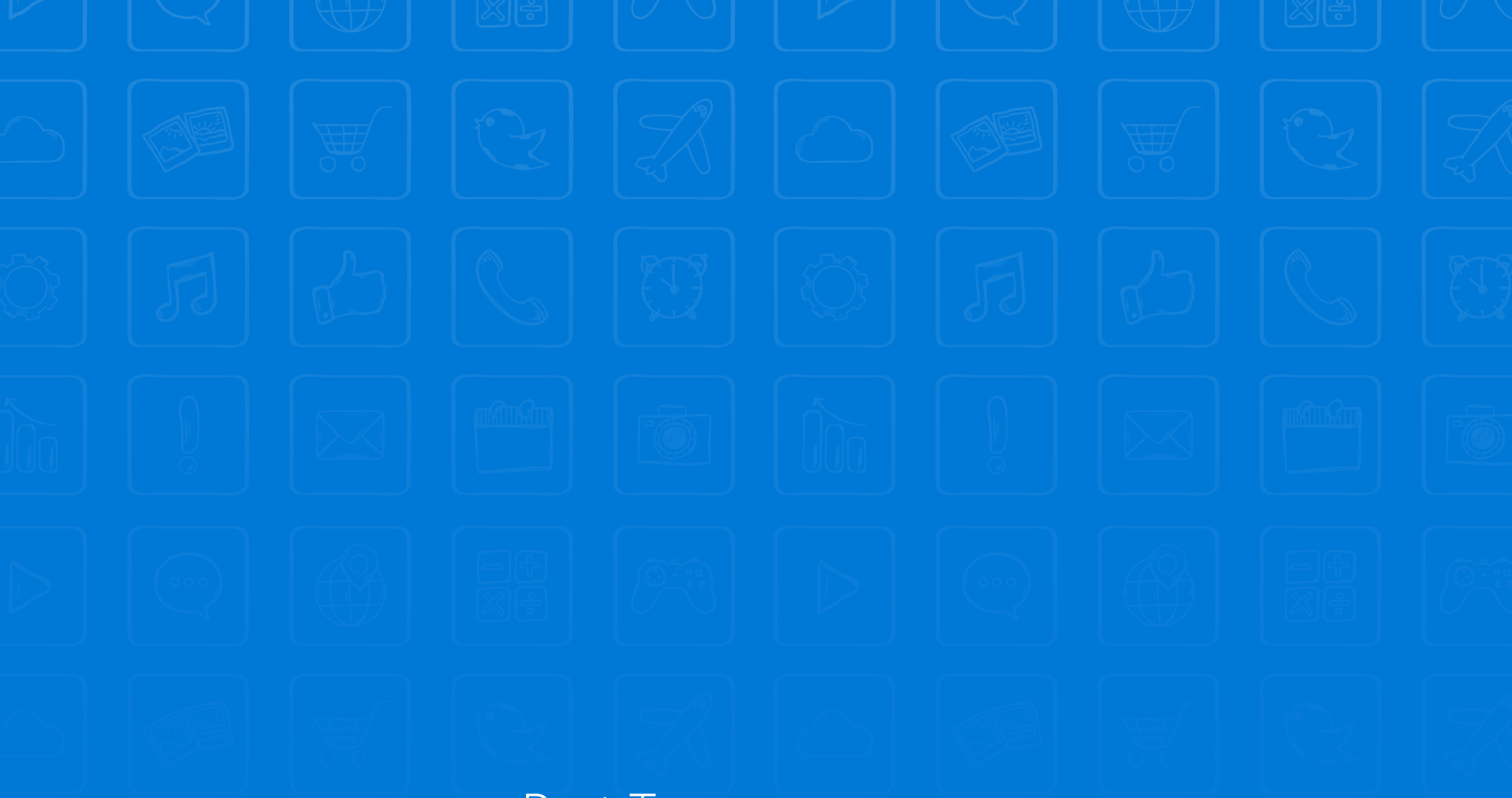
4. If you don't see the CODE hub, but instead see the following dialog, click **Add code**.
   If you just see your Team Project's home page, click the CODE hub link and continue.



5. If your CODE hub looks a bit different than the figure from earlier (usually because you don't have a README.md), click the **Create a ReadMe** file button. Once you've done this, you're ready to continue.



Now you're ready to manage your backlog with Visual Studio Team Services. You can read about account creation and more here. We will revisit Visual Studio Team Services in Part Four of this Guide.



Part Two:

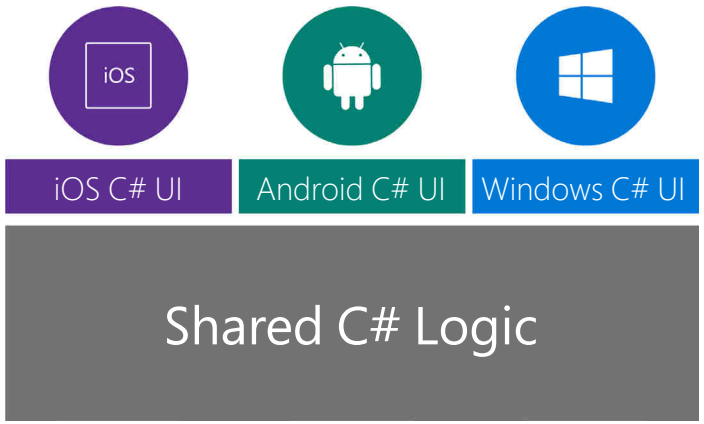
Build native Android, iOS, and UWP apps using C#

Part Two:

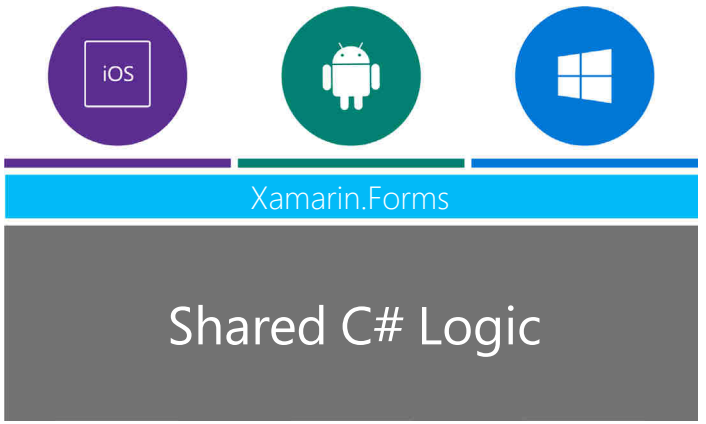
Build native Android, iOS, and UWP apps using C#

Xamarin for Visual Studio makes it possible to build fully native Android, iOS, and Windows apps using a single C# codebase. Anything you can do in Objective-C, Swift, or Java, you can do in C#.

Xamarin provides two ways to architect apps. Whichever you choose, you create fully native apps with shared business logic—all in C#.



Traditional Xamarin approach



With Xamarin.Forms: More code-sharing, all native

Xamarin.Android and Xamarin.iOS provide direct access to platform-specific APIs for advanced customizations and UI.

The Xamarin.Forms API allows you to build native UIs for Android, iOS, and Windows using 90% shared C# code, maximizing code sharing while still delivering 100% native apps. You can author your UI in C# or XAML (eXtensible Application Markup Language), which allows developers to define user interfaces using markup rather than code. Xamarin.Forms is included with Visual Studio.

Read this guide to learn which approach you should use.

Part Two:

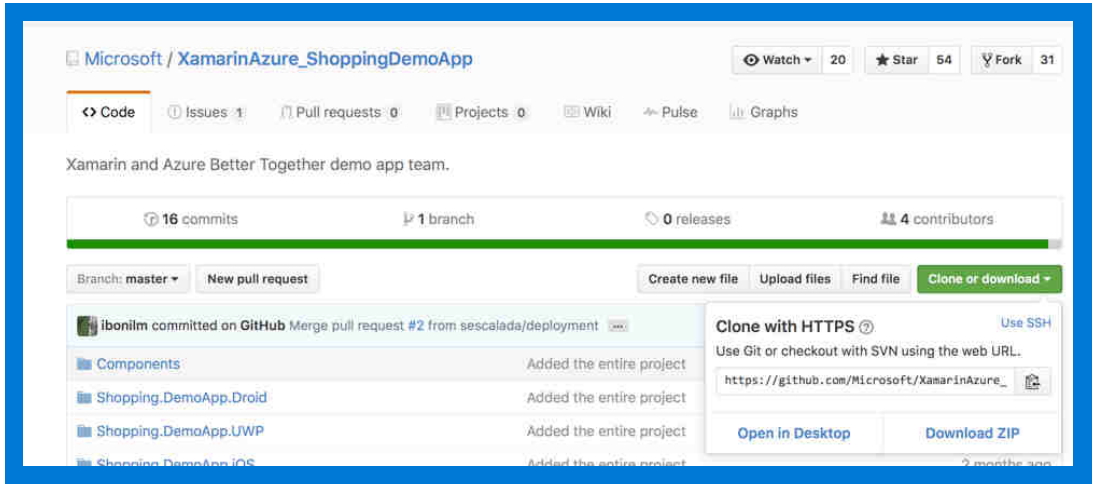
Build native Android, iOS, and UWP apps using C#

Visual Studio Enterprise offers exclusive Xamarin features that simplify development and help you improve your apps.

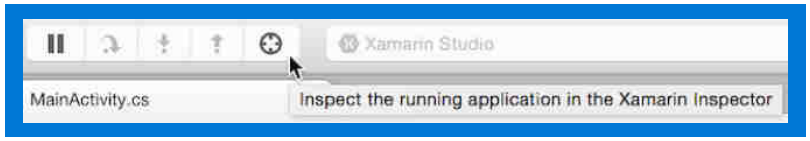
- **Xamarin Inspector** enables rapid design experimentation, so you can interact in real time with production apps. You are able to make immediate changes without having to compile, build, and deploy each time.
- **Xamarin Profiler** is a full-featured profiling tool that helps you analyze issues at code level to identify memory leaks and resolve performance bottlenecks.
- **Xamarin Test Recorder (Preview)** lets you manually interact with your app, record your actions, play them back, and automatically create test scripts that can run in your local environment or on Xamarin Test Cloud's hundreds of device configurations.
- **Embedded Assemblies** uses assembly bundling for compiled code to give your Android apps an extra layer of protection.

To understand the benefits of Xamarin for Visual Studio Enterprise for yourself, follow these steps:

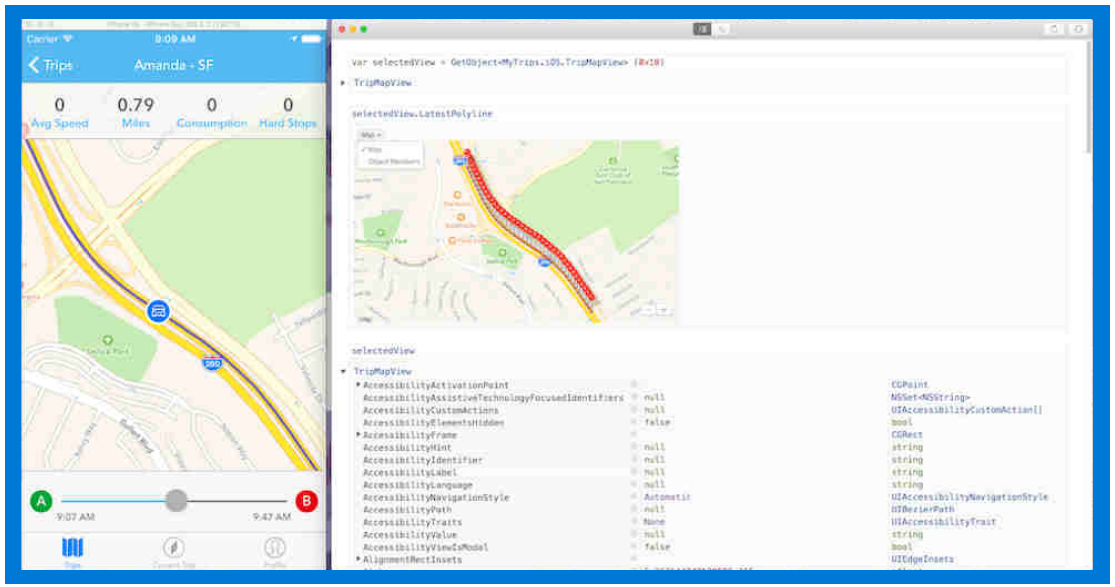
1. If you don't have Visual Studio Enterprise, install Visual Studio Enterprise 2015 and ensure it's completely updated. Xamarin for Visual Studio is part of Visual Studio 2015—ensure this option is enabled.
2. Go to this link: Xamarin-Azure quick starts (Shopping Demo App), click on the green **Clone or download** button, and download the ZIP file.



3. Unzip the file and open the solution file in Visual Studio.
4. Close any open IDE(s).
5. Download and install the Xamarin Workbooks & Inspector for Mac (or download for Windows).
6. Open *Shopping Demo App,* downloaded in Step 3.
7. Run your app in debug mode.
8. Click the **Inspect** button in the IDE toolbar (in Visual Studio, the **Inspect current app...** menu item is also available from the **Tools** menu).



9. A new Xamarin Inspector client window will open with a fresh REPL prompt.



10. Once this window appears, you have an interactive C# prompt that you can use to execute and check C# statements and expressions. What makes this unique is that the code is evaluated in the context of the target process. Any changes that you make to the state of the application are actually happening on the target process, so you can use C# to change or inspect the state of the application live.
11. For more details on Xamarin Inspector and how to use it, check out Xamarin documentation on inspecting live applications.

To start building your own app, download our Enterprise Developer's Guide to Building Five-Star Apps.

Part Three:

Create user-centric experiences with powerful cloud services

Part Three:

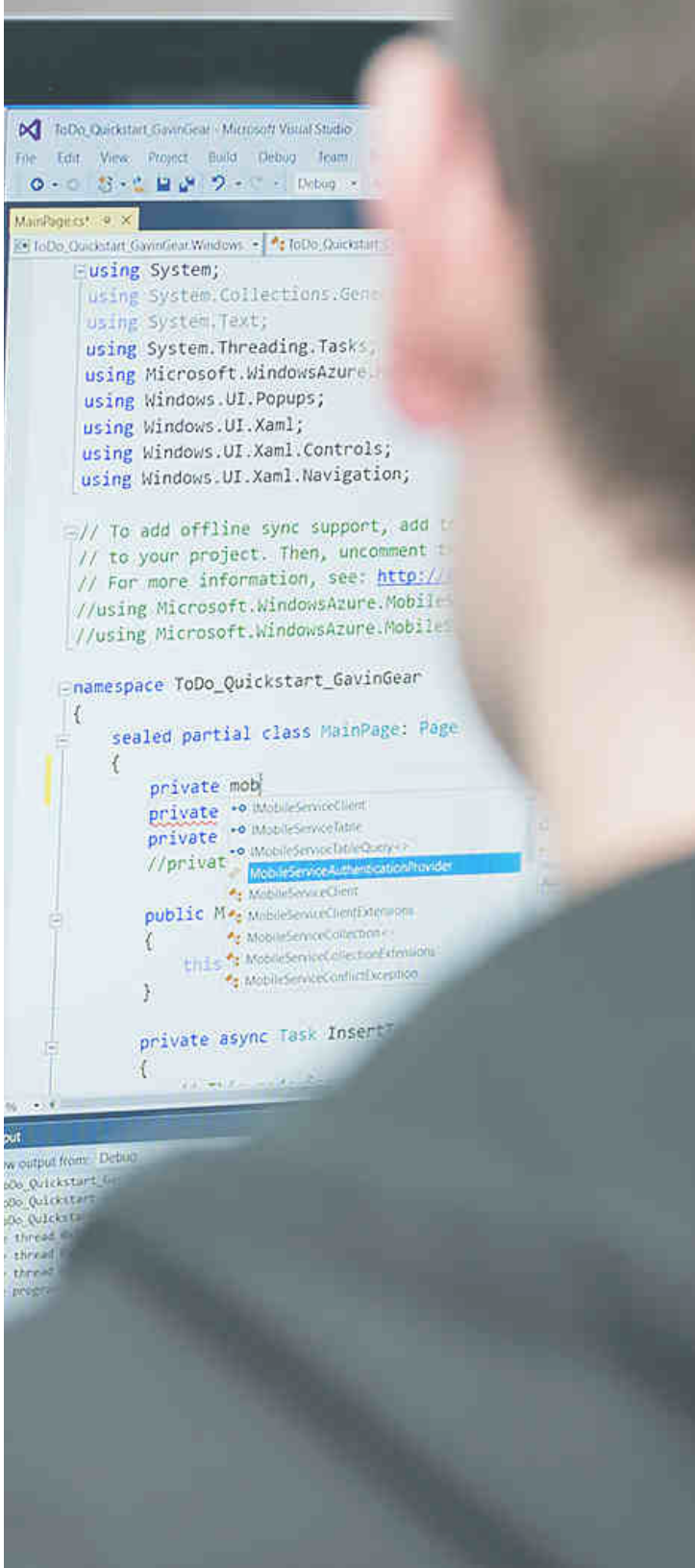
Create user-centric experiences with powerful cloud services

Azure App Service's Mobile Apps feature makes it easy to build powerful, cloud-connected native Android, iOS, Windows, and Mac apps. With Azure App Service, you can scale your app with ease from one to millions of users, and quickly add 50+ cloud services including offline sync, machine learning, image and text recognition, data services, and more. At the end of this section, we've provided an example of how you can use Microsoft Cognitive Services to add advanced functionality to your apps.

Mobile Apps in App Service feature overview

- **User authentication:** Easily authenticate users with supported cloud and on-premises providers.
- **Offline data sync:** Create robust apps that remain useful even when there are network issues, allowing users to create and edit data when they're offline. Cache data locally to improve your app responsiveness and give your users an identical offline experience on all platforms.
- **Storage and data connections:** Store data with ease. Whether it's images, photos, documents, or customer records—in the cloud or on premises—Azure makes it simple to create or integrate the right kind of storage. Azure Storage is massively scalable and accessible on any device, anywhere.
- **Push notifications:** Broadcast personalized push notifications to millions of users in seconds. You can hook Notification Hubs into any existing app backend with ease, whether that backend is hosted in the cloud or on-premises.

We'll use *Shopping Demo App* from the previous section to see how you can deploy some of the features in Azure App Service's Mobile Apps.



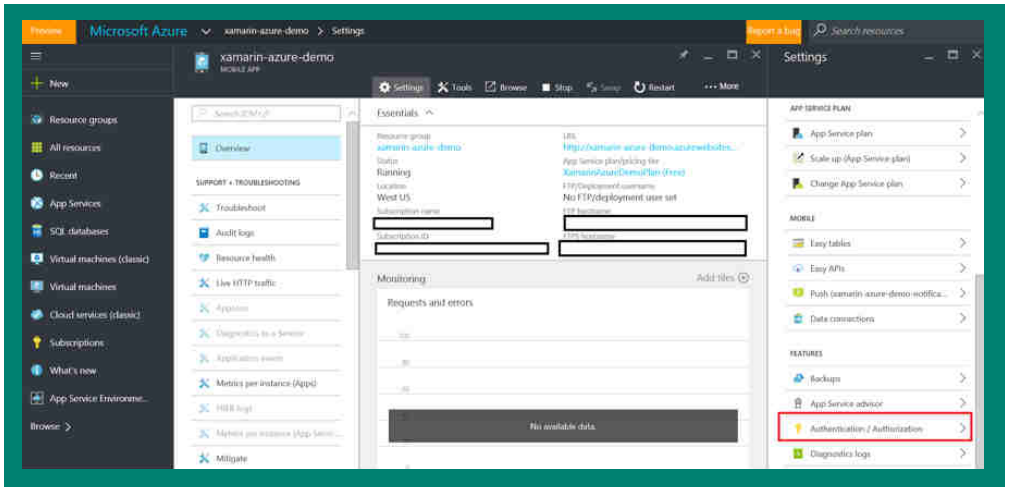
Part Three:

Create user-centric experiences with powerful cloud services

Authenticate and authorize users with Twitter and Facebook

App Service supports five identity providers out of the box, including Azure Active Directory, Facebook, Google, Microsoft Account, and Twitter, as well as your own custom identity solution.

1. Create an Azure account for free. You get credits that can be used to try out paid Azure services. If you're a Visual Studio subscriber, you get monthly credits for Azure services (activate Visual Studio subscriber benefits).
2. Ensure Visual Studio is installed, and open it.
3. You need an **Azure App Service** instance to continue. If you don't have it, follow these instructions to set it up on Android and iOS.
4. Return to the Azure portal and go to **Authentication/Authorization** in the **Settings Pane** under **Features**.

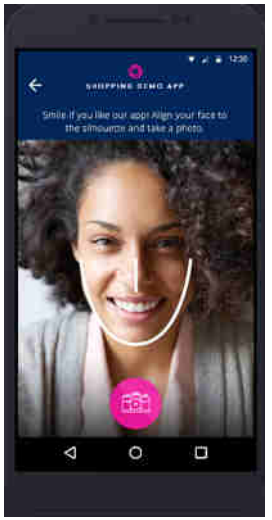


5. Turn the **App Service Authentication** switch to **On** to enable Authentication/Authorization. Since we want to authenticate users just for selling new items, we will allow requests when the user is not authenticated, i.e. taking no action, and control that scenario manually.
6. Twitter Authentication
   a. In the case of Twitter, Azure asks for the **API Key** and **API Secret**. Both are obtained on the social network's website. Learn how to configure your App Service application to use Twitter login.

6. Facebook Authentication
   a. Facebook requires almost the same parameters. Learn how to configure your App Service application to use Facebook login. You also can select which information your app can access once authenticated; however, for the sake of this demo, you won't need to take any additional steps.
7. Client authentication happens entirely in the shared project. The **AuthenticationService** class provides a single instance, simplifying everything to a single call: **RequestLoginIfNecessary()**. Authentication starts when the app asks user to select a preferred provider (Facebook or Twitter) and executes the **Azure Authentication** broker based on user selection. Depending on the target platform, the call to **LoginWithProviderAsync()** differs, although differences are subtle and assure consistency with the operating system's SDK.
8. Get Android, iOS, and Windows code snippets on GitHub.

Use Microsoft Cognitive Services to personalize your apps

Microsoft Cognitive Services are a set of APIs, SDKs, and services to make your applications more intelligent, engaging, and discoverable. Cognitive Services makes it fast and easy for you to add artificial intelligence, machine learning, and advanced data analysis to your apps in just a few lines of code. With 22 APIs across five categories, its powerful algorithms perform everything from OCR to facial recognition and linguistic analysis. Whether you're building consumer or enterprise applications, there's a Microsoft Cognitive Services API that will make your app more innovative, intuitive, and engaging (see all APIs here)



Shopping Demo App uses the Emotion API to analyze faces and detect a range of common facial expressions to rate the app. The device camera captures the user's face and calculates an app rating based on "happiness." Bigger smiles generate higher star ratings and more positive reviews.

Follow the detailed instructions located here to set up the Emotion API and add intelligence to your app.

Part Four:

Ship high quality apps faster with automated testing

Part Four:

Ship high quality apps faster with automated testing

Whether you're a mobile-focused startup looking for your next million users or a multi-million-dollar enterprise delivering apps for your employees, customers, and partners, app quality is critical. Users are less forgiving of performance issues, such as slow load times and crashes, for mobile apps than they are for web apps. Buggy apps don't survive in crowded app stores, where reviews matter and competing apps are just a swipe away. Employees are users, too, and will quickly abandon apps that don't meet their expectations.

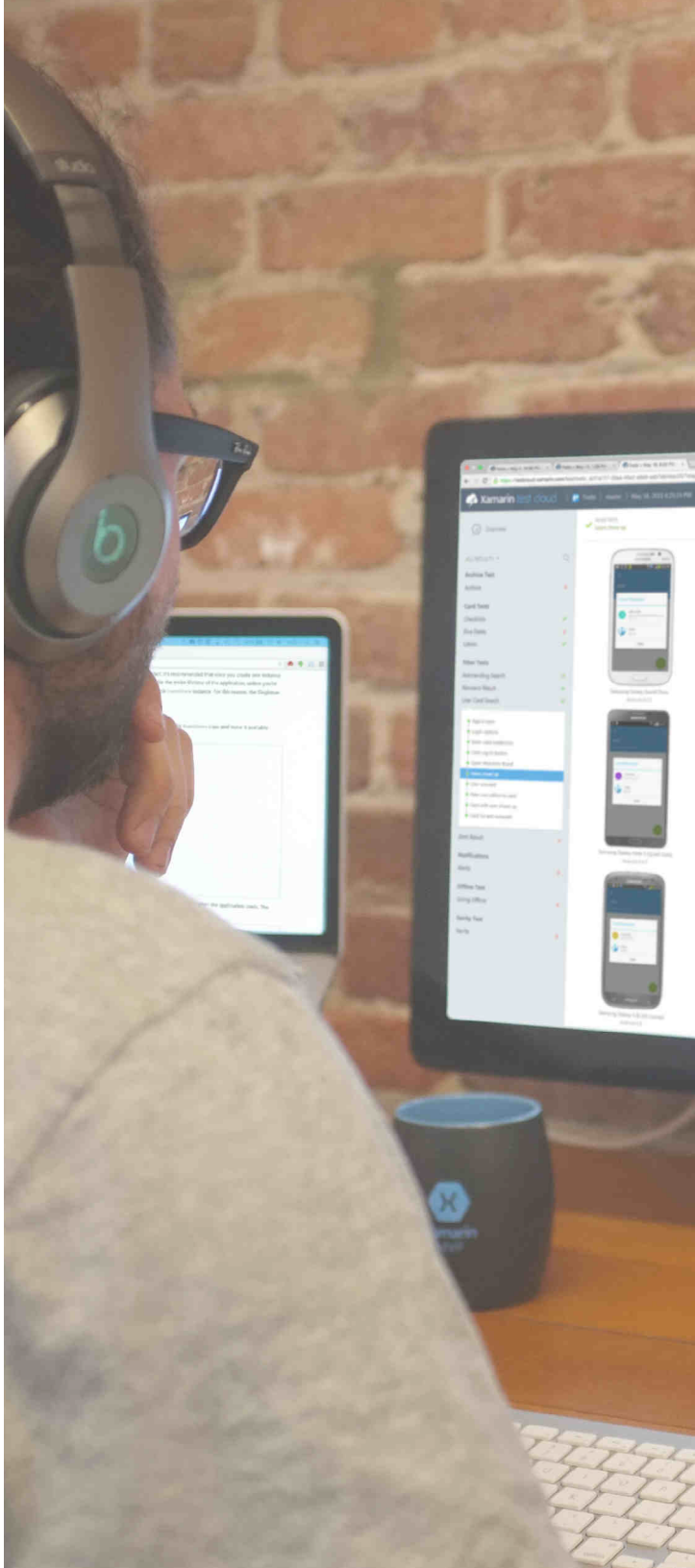
Xamarin Test Cloud allows you to run automated tests on over 2,000 real devices and 400+ hardware and operating system configurations so you can catch bugs early in your development process, find issues before shipping apps, and deliver high quality mobile experiences. To learn more about how Xamarin Test Cloud works, watch this two minute overview.

Getting started with Xamarin Test Cloud

Xamarin Test Cloud's automated test scripts support complex gestures, including pinch, swipe, tap, and zoom, allowing you to test apps in ways that users interact with them.

Xamarin Test Cloud supports two different testing frameworks:

- **Xamarin.UITest:** An automated UI acceptance testing framework based on Calabash that allows programmers to write and execute tests in C# and NUnit to validate the functionality of Android and iOS apps. Xamarin.UITest has IDE support for both Visual Studio and Xamarin Studio and requires NUnit 2.6.3 or higher. It is not compatible with NUnit 3.0.
- **Calabash**: Test scripts are written in Ruby. Calabash is based on Cucumber, a popular testing framework for Ruby.
- **Note**: you can test *any* app with Xamarin Test Cloud, including Objective-C/Swift, Java, Xamarin (C#), and hybrid apps. All you need is an app package.



Part Four:

Ship high quality apps faster with automated testing

Since *Shopping Demo App* is built in C# using Xamarin for Visual Studio, you can use Xamarin.UITest to set up a test for this app in Xamarin Test Cloud. As part of Visual Studio Enterprise, you get Xamarin Test Recorder (currently available for Android devices), which makes it easy to record automated mobile tests. All you have to do is turn on Xamarin Test Recorder and perform the actions in question in your app. Tests captured with Xamarin Test Recorder are written in UITest. You will need a Visual Studio Enterprise subscription to use Xamarin Test Recorder.

Recording tests for an Android application requires:

- A physical device connected via a USB cable OR an Android emulator such as the Visual Studio Android Emulator OR that the APK has been granted **INTERNET** permissions in the Android manifest.
- That the build does not use the **Shared Mono Runtime** setting in the Android project options.

For recording iOS tests, use Xamarin Test Recorder (Preview) for Mac OS X.

1. Start a 30-day free trial of Xamarin Test Cloud
2. Set up a Xamarin Test Cloud team as described in the Xamarin Test Cloud Organizations & Teams documentation.
3. Xamarin Test Recorder for Visual Studio can be downloaded and installed from the Visual Studio Gallery. Download the extension for either Visual Studio 2015 or Visual Studio 2013.
4. To start recording tests from Visual Studio on your Android device, follow the instructions at Xamarin Test Recorder for Visual Studio.
5. To upload your tests to Xamarin Test Cloud and start testing your apps, follow the instructions at Submitting UITests to Xamarin Test Cloud.

Part Four:

Ship high quality apps faster with automated testing

We recommend the following best practices to get the most from your testing efforts and ensure high quality apps.

Mobile App Testing Best Practices

- Create a culture of quality across all team members, including development, operations, and product.
- Be ambitious when deciding how often you want to ship updates. High performing organizations ship once a week, with the industry average around 30 days.
- When planning test automation coverage, make sure you cover critical paths, must-have features, and user interactions.
  - As you mature, plan to cover 100% of your app—including all user interactions—with automated test scripts.
- Start to think about your pipeline flow and what it means for developers early in the planning stages. Ensure that processes are not creating long wait or idle times.
- When it comes to device coverage, decide on a strategy *before* selecting devices. There are multiple approaches, based on your needs:
  - Analyze your usage data to determine which devices are popular or widespread with your users.
  - Determine which percentage of your userbase you want to cover (70%, 80%, etc).
  - Take cues from operating system vendors, as they release stats on percentage of users who've upgraded. As percentages shift, adapt your coverage strategy.
  - For revenue-generating apps: select user coverage based on which user groups and devices result in the most spending.

Part Five:

Get early user feedback and analytics

Part Five:

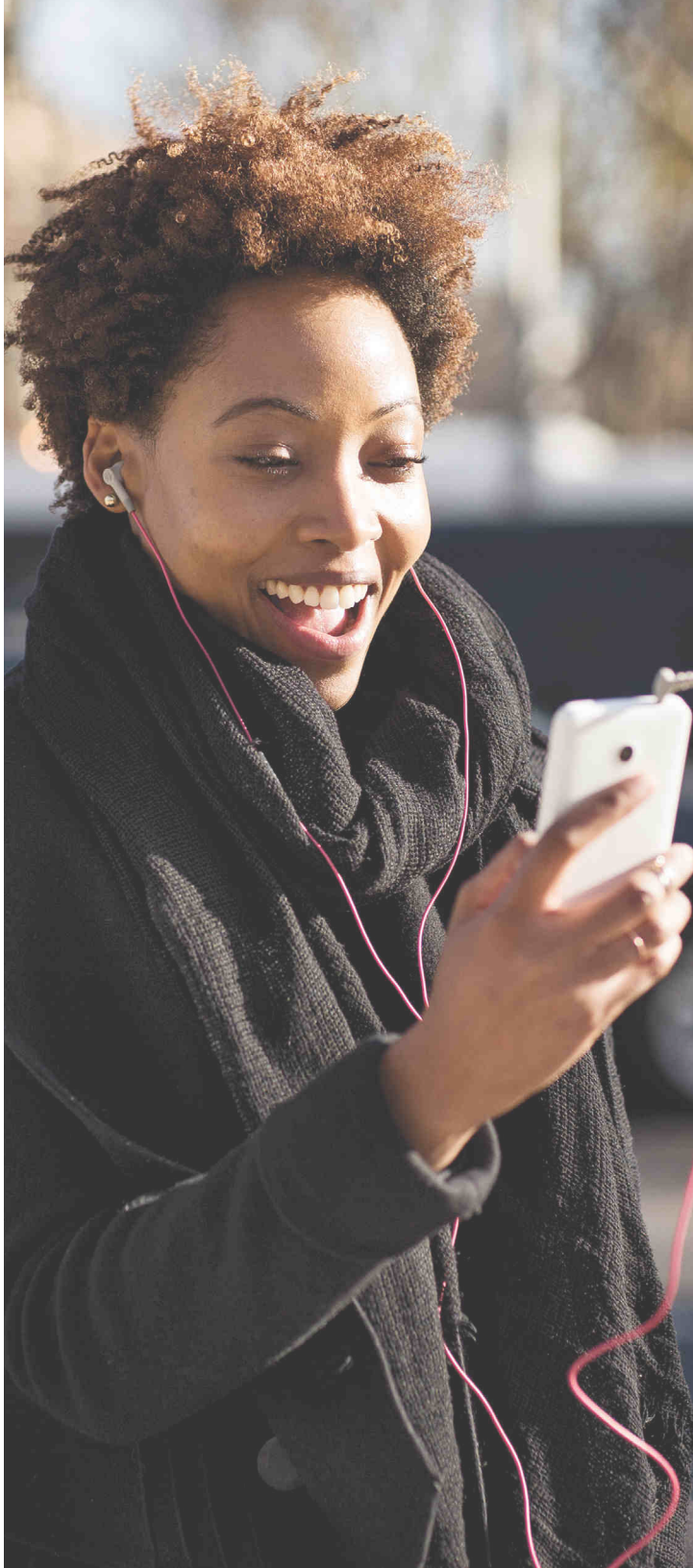
Get early user feedback and analytics

Before general releases or major enhancements to your apps, it's important to gather feedback, validate user behavior, and catch user-specific issues. **HockeyApp** integrates with Visual Studio Team Services or Team Foundation Server to streamline development, management, monitoring, and delivery of your mobile apps. You can easily distribute beta versions of your apps, inviting internal and external users to download, provide feedback, and suggest future improvements.

How does beta distribution work?

Before you get started with beta distribution, make sure you've integrated HockeyApp with Visual Studio Team Services. This allows you to implement changes based on the feedback you receive from your users and deploy those updates fast. Follow the steps in Part Seven to connect HockeyApp to Visual Studio Team Services.

On Android, testers open the downloads folder in the device's browser, download the .apk file and open the app from the download area after the download finishes. iOS uses the official over-the-air enterprise distribution mechanism, with testers visiting http://config.hockeyapp.net to install a web-clip on their home screen and get access to all installable apps. This process works for both ad-hoc and enterprise distribution.



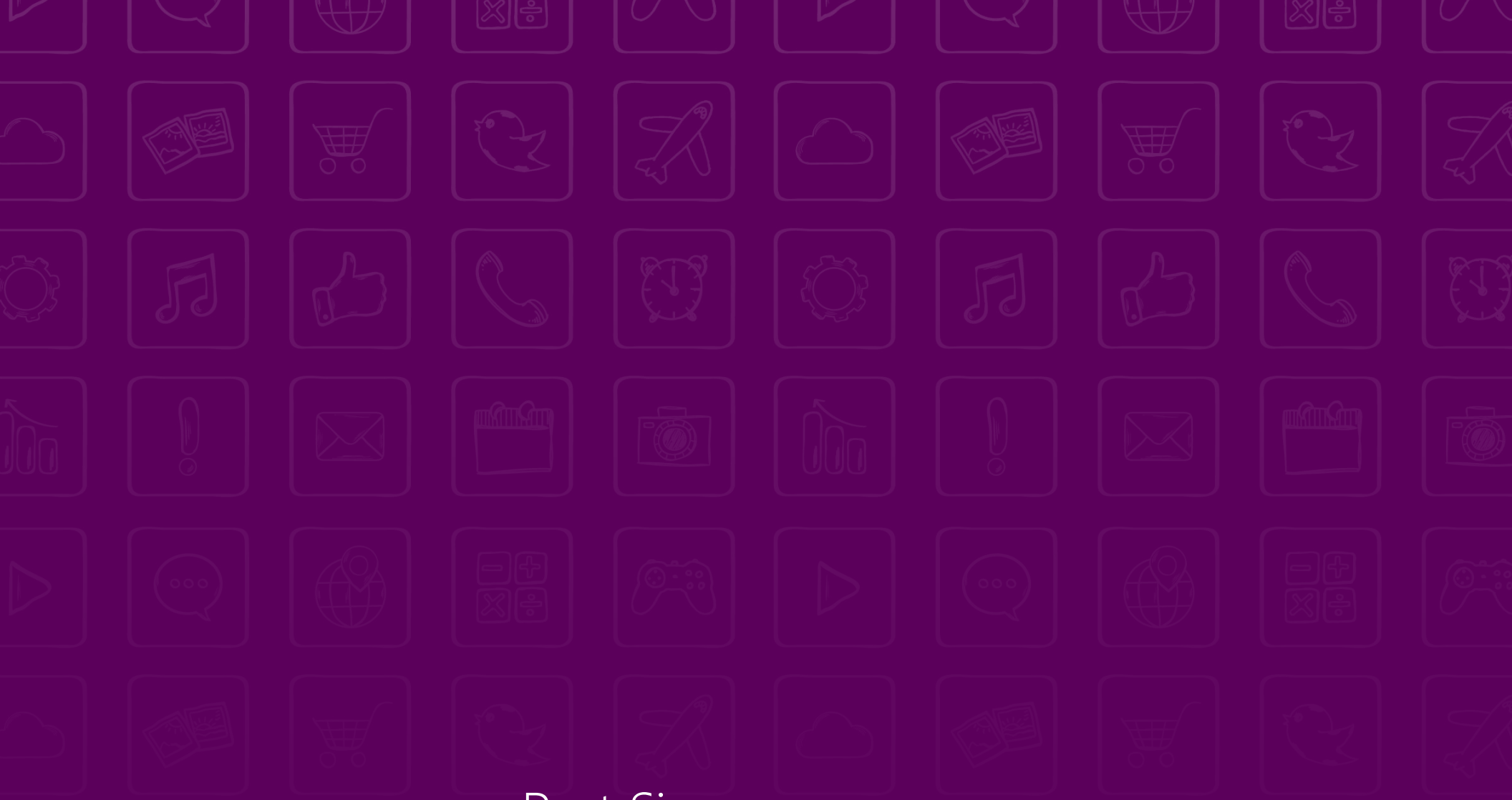
Part Five:

Get early user feedback and analytics

You can also distribute your app via a public URL. If you enable the public download page, anyone with the URL whose Unique Device IDentifier (UDID) is in the provisioning profile will be able to access and download the app. You can find this in the info section of your app's page. Your testers will be able to download the latest version and install it directly on their device from that web page. If you want to restrict the download page to registered HockeyApp users only, you can use the private download page.

Invite Beta Testers to test and provide feedback on your app:

- If you already have a group of beta testers and have added their UDIDs to your provisioning profile, all you need to do is to share the link to the public download page. To add new devices to your provisioning profile, follow the steps here.
- You can also send email invitations to testers by selecting your app from the HockeyApp dashboard, clicking on **Invite User**, selecting a role, entering the user's email, and hitting **Save**.
- To bulk invite users, choose the **Import** button at the top of the **Users** page. Save your list of email addresses in an Excel file on your desktop, one address per line. Upload this file using the **Import File** function on the page. If your app already has a provisioning profile and HockeyApp finds existing users for the UDID in this profile, it will allow you to invite these users directly from the import page.
- For more information on inviting users, refer to the detailed instructions here.



Part Six:

Make each release better than your last with Continuous Integration

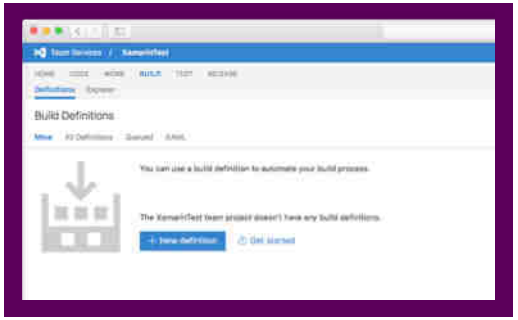
Part Six:

Make each release better than your last with Continuous Integration

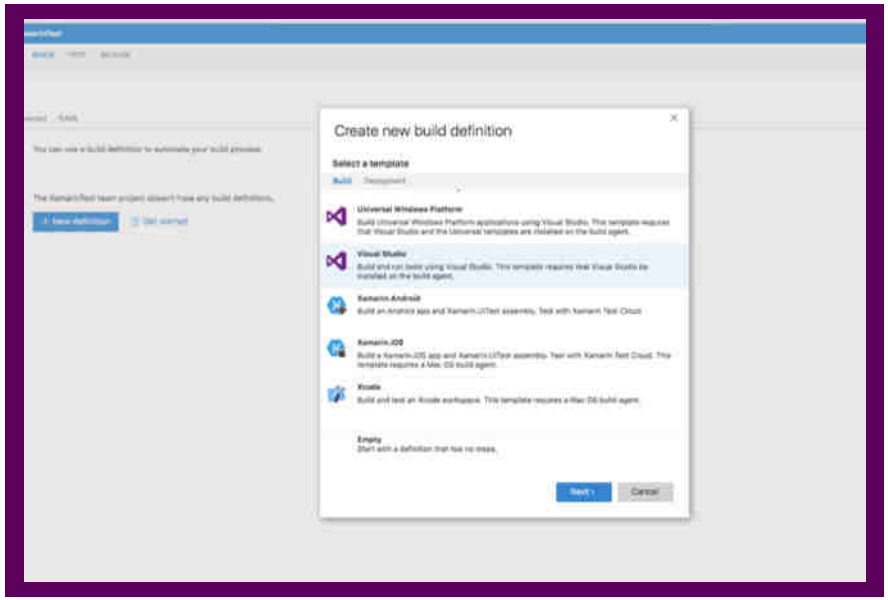
Visual Studio Team Services helps teams working in any language or platform to plan their work better, code together, and ship faster. Teams can plan and track work with agile tools like Kanban boards and task boards, code together with unlimited private Git repos, and set up Continuous Integration and delivery with build, test, and release tools. Visit our webpage for more details.

Set up Xamarin Test Cloud with Visual Studio Team Services

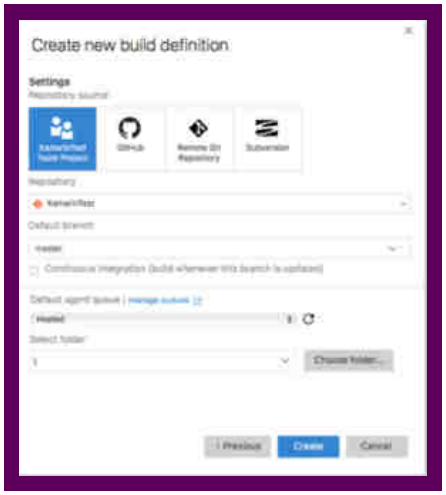
1. Sign up for a Xamarin Test Cloud account if you don't already have one.
2. Open Visual Studio Team Services and log in.
3. In the **Build Definitions** tab, click the **New Definition** button.



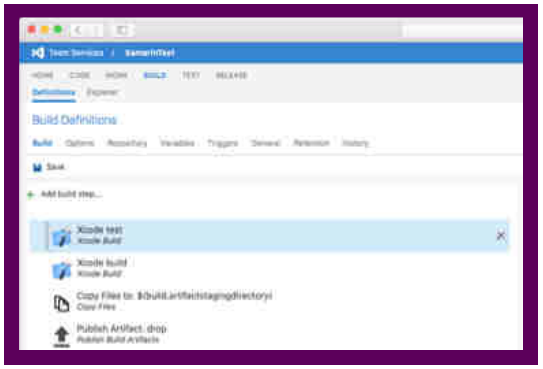
4. In the **Create new build definition** dialog, select a build definition template and click the **Next** button.



5. In the **Create new build definition** dialog, configure your settings and click the **Create** button. Your build definition may or may not have a Xamarin Test Cloud build step, depending on the build definition template you selected.



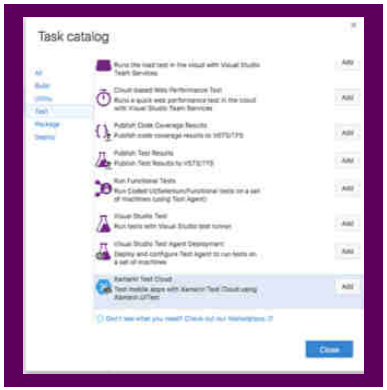
6. If your build definition lacks a Xamarin Test Cloud build step, in the **Build** tab click the **Add build step...** button.



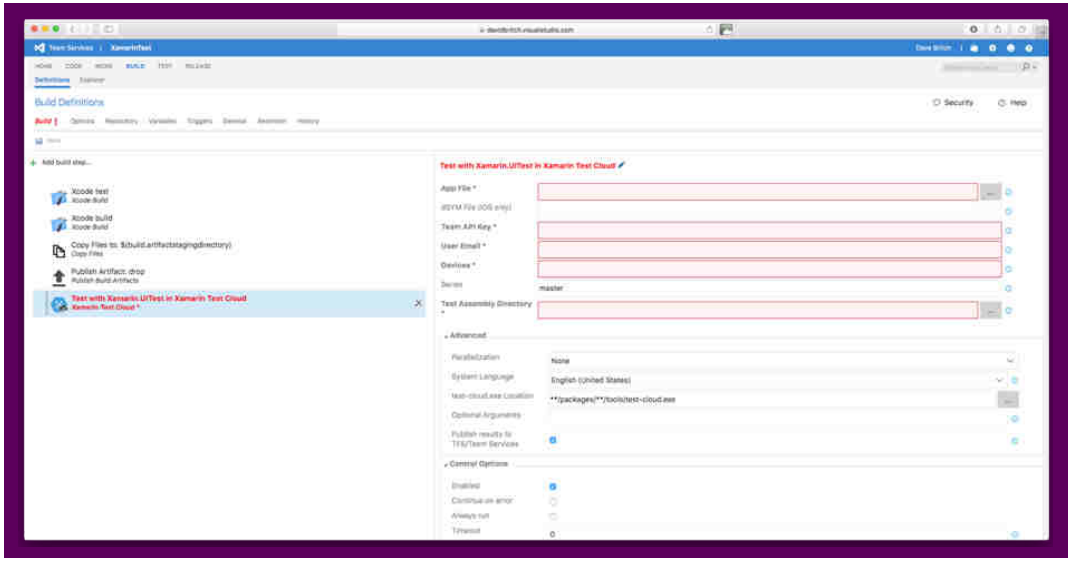
Part Six:

Make each release better than your last with Continuous Integration

7. In the **Task** catalog dialog, select the **Test** tab and then click the **Add** button for the Xamarin Test Cloud catalog item. Close the dialog by clicking the **Close** button.



8. In the **Build** tab, select **Test with Xamarin.UITest in Xamarin Test Cloud** and configure your settings. Input all of the parameters inside Visual Studio Team Services to run tests in Xamarin Test Cloud.



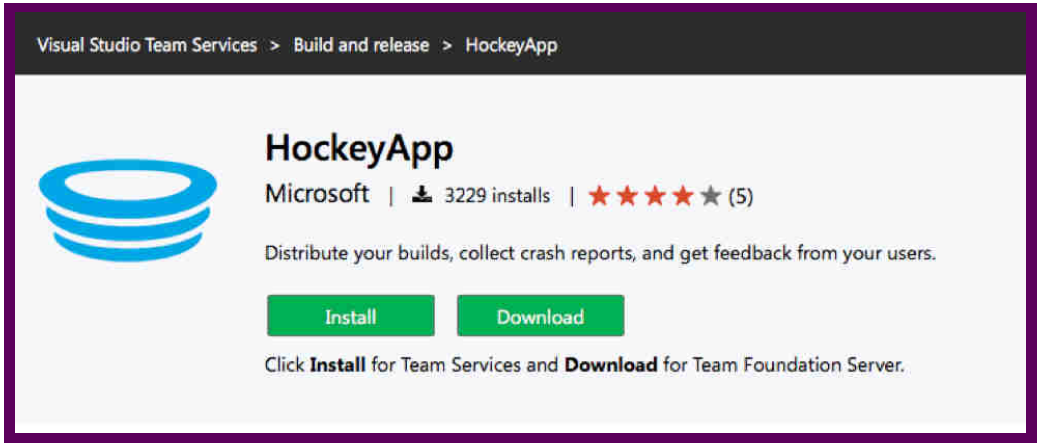
9. Follow the steps outlined here for getting the right information into each field/parameter as you see in the image above

Part Six:

Make each release better than your last with Continuous Integration

Follow the steps below to integrate HockeyApp with Visual Studio Team Services, allowing you to start collecting beta user feedback and monitoring for production app crashes.

1. Sign up for a HockeyApp account using your Microsoft Account, social media account, or email address.
2. Follow the instructions to install the HockeyApp extension for Visual Studio Team Services.



3. Further detailed instructions on how HockeyApp works with Visual Studio Team Services and TFS can be found here.

Part Seven:

Continuously improve and make your users happy

Part Seven:

Continuously improve and make your users happy

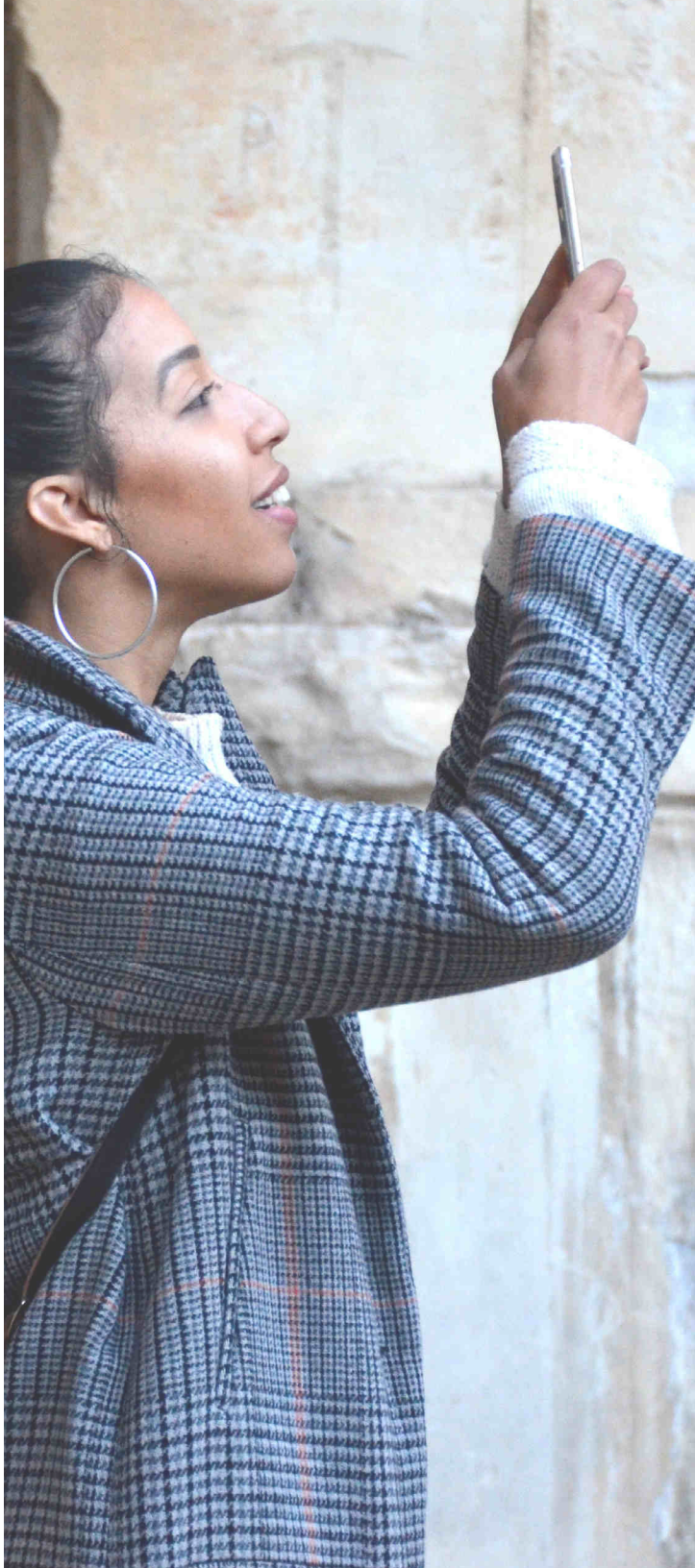
Users expect apps to have the newest features with each release of a new operating system, and that any bugs that pop up as a result of OS changes (or for any other reason) will be fixed on a regular basis. With HockeyApp, you monitor your apps in the wild, gathering data and insights from real users to plan future improvements.

You also need to continue to ensure top-notch performance. When apps crash, you need to know why and be able to resolve issues quickly. One way to do this is to look at ratings and feedback in the app stores. However, once you're at this stage, it means users have already had a bad experience and most likely found a competitor to fulfill their needs.

HockeyApp works with Visual Studio Team Services to help your organization implement Mobile DevOps. The HockeyApp SDK will collect usage data, crash reports, user feedback with screenshots, and show an alert when the next build is available. Closing the loop, HockeyApp can automatically create work items for a new crash group or feedback thread and keep the status in sync with Visual Studio Team Services.

Follow these instructions to set up HockeyApp for the Shopping Demo app:

- Android
- iOS
- Windows



Conclusion

You've now successfully built fully native Android, iOS, and UWP apps, as well as connected to powerful cloud services for critical mobile functionality. You've walked through each stage of the mobile DevOps lifecycle, from planning your projects using Visual Studio Team Services to collecting and responding to user feedback using HockeyApp.

Whether you're creating your first mobile app from the ground up or enhancing existing apps, Microsoft's mobile DevOps solution gives you everything you need to plan, develop, test, monitor, improve, and scale your apps.

With this guide, you have the technical resources, samples, and expert best practices you need to build fully native apps, all in C#.

Start your free Visual Studio Enterprise trial today