Cutting Right to the Code:

A mobile developer's guide to help eliminate non-coding tasks and get code done faster

Microsoft Azure

Microsoft

Abstract

This eBook helps mobile developers cut right to the code and get code done quickly by providing an overview of key services in the cloud, what services to use based on your needs, step by step guidance, sample code, sample applications, and a free account to get started.

Developers get out of bed wanting to write code. Unfortunately, many non-coding tasks need to be wrangled before coding can begin. We don't mean coffee; we mean things like waiting for servers to be purchased, installing your database engine, configuring your firewall ports, local data caching options, figuring out your deployment toolchain, creating a reasonable authentication approach, getting your shared libraries in order, and getting the testing frameworks lined up. Those are just the things in the way, not the things that aren't in the way but can't be forgotten, such as backups, scale planning, and logging. These tasks cost you precious time that could be better spent coding, brainstorming, standing around the whiteboard, playing foosball, etc. Once you jump through the necessary non-coding hoops and finally get to coding, the question quickly becomes, "How do you get code done faster?"

Taking advantage of the right services running in the cloud helps developers cut right to the code by eliminating a number of non-coding tasks. Examples include environment management, spinning up a SQL Database in Azure to skip version problems, automating backups, enabling advanced "encryption at rest" features, using repeatable scripting of environments to make sure services are enabled, adopting the CI toolset of Azure Mobile Services to publish straight from VSTS (oh, hello again, three otherwise lost weeks), and provisioning Azure AD for authentication and authorization.

Then there's "getting code done fast." As one developer put it, there are two types of code:

1. The fun stuff: Code that makes your app unique and valuable, and helps drive the business.

2. The not fun stuff: Glue code which is less game-changing but makes stuff work. An example would be building framework code to compensate for how different browsers or devices work.

Powerful finished services in the cloud can help mobile developers get from 0-60 in their app building process. How about a fully-built mobile backend that integrates with iOS, Android, and Windows? Services that enable push notifications, analytics, and caching. All of it can be accessed with a few lines of code.

The time is now. We invite you to cut right to the code.

PUBLISHED BY
Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Table of Contents

Overview

Introduction

With over sixty services, Microsoft Azure provides a modern platform for any app, written in any language. The lifeblood of the private and public cloud is applications that expose rich, interactive sites and services to everyone on the planet, accessible via a mobile device. Azure App Service helps you build modern mobile apps that scale. It is a cloud platform to build powerful web and mobile apps, for any platform and any device that connects to data anywhere in the cloud or on-premises. Built for developers, App Service is a fully-managed platform with powerful capabilities that make it easy to stage and then deploy to production with support for automatic updating. You can code in your language of choice, use your favorite development tools, and then easily scale up and out to meet the demands of your business and customers.

Azure App Service provides an abundance of services and features in four categories:

- Mobile Apps
- Web Apps
- API Apps
- Logic Apps

This guide will cover how you can build a great mobile experience by leveraging the Mobile Apps in Azure App Service. Other guides will cover other parts of Azure App Service.

Pro Tip

One thing to watch for in this guide is the *Additional resources* section. Each one will provide additional links to more detailed documentation about the subject you've just read about. In addition, many will have a link to the hands-on labs, sample code, and more. These labs contain tons of sample code, step-by-step walkthroughs, and ARM templates to help you explore and get to know Azure App Service and mobile apps.

Microsoft Azure App Service is a platform-as-a-service (PaaS) offering. You use it to create web and mobile apps for any platform or device. You can use it to integrate your apps with SaaS solutions, connect with on-premises applications, and automate your business processes. Azure runs your apps on fully-managed virtual machines (VMs), with your choice of shared VM resources or dedicated VMs.

App Service combines the web and mobile capabilities that were previously delivered separately as Azure Websites and Azure Mobile Services. In addition, there are new capabilities for automating business processes and hosting cloud APIs. As a single integrated service, App Service lets you compose various components into a single solution.

Why Use App Service for your Mobile Apps

Azure App Service provides a number of great benefits to you and your application. When looking at App Service over other options, a key foundational tenet is that App Service manages the underlying infrastructure. As a developer, you're not responsible for patching and maintenance. This is a major non-coding task that is eliminated. Speaking of code, App Service provides choice with first class support for ASP.NET, Node.js, Java, PHP, and Python.

App Service was built with DevOps in mind. You can set up continuous integration and deployment with Visual Studio Team Services, GitHub, or BitBucket. You can shepherd updates through test and staging environments as well as perform A/B testing. Because App Service has a rich API surface, you can manage your apps using Azure PowerShell or the cross-platform command line interface (CLI) tooling.

Once you've got your mobile app backend running in Azure, you're supported by the global scale and reach of twenty-four world-wide regions so you can host your app where you need it most. Additionally, you'll have the peace of mind that comes with App Service's high availability SLA.

With all of these benefits, you'll want to get started quickly. In the rest of this guide, you'll see how to use dedicated tools in Visual Studio to streamline the work of creating, deploying, and debugging your web app.

Getting Started with Microsoft Azure

In order to build and deploy your web app to Microsoft Azure, you need a subscription. You can get a free Azure account or it's possible you already have access and might not even realize it. You could have access via:

- Your organization's subscription
- Your own subscription provided via your MSDN Subscription
- Your own subscription provided via Visual Studio Dev Essentials

Regardless of how you have access, you'll need enough permissions to create and manage new objects in the subscription.

Common Mobile App Scenarios

Mobile apps without backend services are typically very limited. The promise of mobile apps is having access to information and being able to act on it from wherever you are. Azure App Service helps you by providing rich backend services so you can focus on the mobile user experience and application-specific logic. In addition, you often have parts of your application that can be accessed using both web and mobile interfaces.

When to Use It

Mobile Apps in Azure App Service offer a highly scalable, globally available mobile application development platform that brings a rich set of capabilities to mobile developers.

The modular architecture of Mobile Apps in Azure App Service allows you to choose the features you want to use in your mobile apps whether they are native, cross-platform Xamarin, or Cordova (Adobe PhoneGap). Also, because Azure provides the core set services as an app type under the general App Service umbrella, your mobile solution has access to all of the other Azure services like Azure AD, Redis Cache, and Azure CDN, to quickly build and deploy powerful apps.

As with most technologies, Azure continues to grow and evolve at a rapid pace. Steps to complete certain tasks get optimized out, tools get updated, and generally things just get better. This document reflects the state of the art as of June 2016, using versions with the latest updates and patches. Additionally, as with many things, there is more than one way to create something.

The tools that enable you to get the most done in as few steps as possible include:

- Visual Studio 2015 with Xamarin
- Azure SDK and Tools for Visual Studio 2015

Azure App Service and the Mobile Apps component allow you to get started from nothing or to integrate mobile clients with existing services you may have already built for your application. When starting from "File | New", you can start from the Azure portal or you can start from Visual Studio. It's up to you. Starting from the Azure portal gives you additional flexibility if you want to have a non-.NET backend like Node.js and/or want to start with native templates for iOS or Android that aren't built into Visual Studio. When you start from the Azure Portal, it will give you a sample app server backend—Node.js or .NET—with a SQL Database and a matching client. Azure preconfigures the sample with a "to do" style application. This can be a great starting point if you're new to Azure or mobile client development. You can evolve the samples as you see fit. If you choose a .NET backend, you will be able to download a Visual Studio solution. Regardless of what client type you pick, you'll get a set of files you can use in the appropriate client tooling environment. This type of starting point is great when your focus is on the mobile clients and backend services specifically for the mobile clients. You get to "cut to the code" and not focus on setup and configuration hassles.

Azure provides client SDKs to make it easier to build your mobile clients and work with the various services exposed via an Azure Mobile App. You'll find support for working with data, authentication, push notifications, and more. The set of Client SDKs covers native development (iOS, Android, and Windows), cross-platform development (Xamarin for iOS and Android, as well as Xamarin Forms), and hybrid application development (Apache Cordova). Each client SDK is available with an MIT license and is open source.

You're not limited, however, to building your mobile app solution one way. You can also use Azure App Service to expose APIs and features in a large-scale application that includes not only mobile clients, but desktop, web browsers, and other services. Visual Studio with the Azure SDK and Tools allows you to add REST API Client support to existing applications via a context menu. You can point Visual Studio at your Mobile App or other Azure App Service endpoints and have it help create types that match your server-side API. This allows you to have a cohesive set of backend services for both mobile and other "clients".

Additional steps

After you've got your mobile app service loaded in Azure, there are additional steps you might want to perform.

Custom Domain

By default when you publish your web site to Azure, you'll be able to access it via myapp.azurewebsites.net, where "myapp" is the name of your deployed service. In particular, this is the only option when using a free App Service Plan—the settings in Azure that control scale and implicitly pricing. If you upgrade to at least the Basic plan, you can configure your web app with a custom domain name. If you don't have a custom domain already, you can buy and register one from the Azure portal.

SSL Certificate

One great thing about the free Azure App Service plan is that you get SSL included. However, once you add a custom domain, you'll need to provide your own SSL certificate. To do this, you'll need to acquire an SSL certificate from a recognized certificate authority. You'll then need to ensure you're using the correct pricing tier. You then configure your web app to use the cert. Finally, you might need to force your app to support SSL only.

Continuous Delivery

While using the Visual Studio tools for initial deployment is great, long term use, especially when working on a team, is not ideal. You'll really want to consider using one of the great tools out there such as Visual Studio Team Services (VSTS) or GitHub. Both services provide mechanisms to update your web app after changes are pushed to a centralized source repository. VSTS in particular supports a full Release Management tool that supports release pipelines with deployment to staging slots and automated regression testing before going live. The Team Services docs provide additional details, pointers, and walkthroughs.

Additional resources

- Configure a custom domain name in App Service
- Enable HTTPS for an app in Azure App Service
- Visual Studio Team Services
- Hands-on labs, sample code, and more

Identity Management

Overview

Modern mobile apps often need to restrict what data and features are made available to individuals and/or groups of users. Azure App Service supports single sign-on (SSO) for your apps regardless of where users are logging in from, whether it be the public internet or your own internal network. In addition, you can integrate social logins—Facebook, Twitter, etc.—for customers and non-corporate members through integration with OAuth 2.0, OpenID Connect, and SAML 2.0.

The problem

How do you get an app that's run on-premises to be cloud aware and support logins outside of your corporate network? What about customers and business partners who need access to your app but aren't employees?

The solution

Azure Active Directory (AD) provides organizations with enterprise-grade identity management for cloud applications. Azure AD integration gives your users a streamlined sign-in experience and helps your application conform to IT policy. By enabling Azure AD support in your application, your users won't need to remember an additional set of credentials. Instead, they'll use the same information already in use to access your organization's resources. In addition, using Azure AD, you gain access to a very large social network in that you can allow users with Microsoft accounts. Finally, Azure App Service enables integration with popular social media providers.

Services used

- Azure Active Directory
- Azure App Service

Azure Active Directory

Adding Azure AD support to your application means that management is done with familiar concepts for managing user provisioning and access control. If you have an on-premises directory, you can sync with Azure AD. You can re-use existing Azure AD groups and distribution lists, and more. Administrators can assign access to apps, whether it be for specific users or entire groups.

In addition, your application has access to this information using the Azure AD Graph API. This API allows your app a number of identity and access control operations including:

- Create a new user in a directory
- Get a user's detailed properties, such as their groups
- Update a user's properties, such as their location and phone number, or change their password
- Check a user's group membership for role-based access
- Disable a user's account or delete it entirely

To get started, access the classic Azure portal at https://manage.windowsazure.com/ and log in (Microsoft's migration from the classic portal to the new portal is near complete so only use the classic portal when necessary). The portal contains a section labelled **Active Directory** that provides the controls to define users, groups, and applications. To be able to control who can use your application, you can create your own Azure AD directory or use an existing one—it depends upon how you want to segment access to your application. Within your AD you will need to create an *application*, which is a set of configuration settings that tell AAD about your mobile app.

If you've created a new directory, you will need to add users. One of the great things about Azure AD is that it provides support for adding new and existing users from other sources. Using the Azure Management port, you can add:

- A new "native" user to the directory
- An existing user in another Azure AD directory
- An existing user with a Microsoft account
- Users in partner companies

Assuming you have users in place, you need to define an application object in the directory that you'll map to your web app. When you add your application, you provide a name and specify the type of application. Currently you have two choices:

- Web Application and/or Web API
- Native Client Application

The choice in this case may not be obvious if you're focused on a mobile app, but, since you're building a service, you'd want the first option. Once you pick the web app related option (which covers mobile app services as well), you need to provide two items. The first is the *Sign-on URL* which is the URL a user will use to log in to your app. The second item is the *App ID URI*. This just needs to be a unique URI (often provided in the form of a URL) the Azure AD can use to identify your app. Once you've entered these basic items, you can then use the **Configure Azure AD Authentication** wizard in Visual Studio to add SSO support via Azure AD to your web app.

Social Sign In

When using Azure AD, you gain access to a very large social network in that you can allow users with Microsoft accounts—the same ID shared by Xbox and Outlook.com for example—to use your application. Adding social sign-in via other providers to your web app requires three steps. First, you need to gain developer access to your social networks of choice, register your application, and get items like access tokens. Second, once you have this information, you use the Authentication / Authorization blade in the Azure Portal for your web app. Once you turn this feature on, you have access to a number of social networks like Facebook, Google, and Twitter.

Third, you will need to tweak your application so that users can choose the supported login types that you've enabled, both for successful and failed logins, and redirect the user to the right place in your web app once it has completed the authentication and authorization dance.

Additional resources

Azure App Service Security ➲

Azure Active Directory Management Portal ➲

Hands-on labs, sample code, and more ➲

Push Notifications

Overview

Smartphones and tablets have the ability to "notify" users when an event has occurred. These notifications can take many forms.

The problem

Your mobile solution runs on multiple mobile platforms like iOS, Android, and Windows Mobile. You need a single backend to be able to integrate with your server code, yet work with all of the vendor's notification systems.

The solution

Azure Notification Hubs allows you to send cross-platform, personalized mobile push notifications with a single API call. You can easily integrate them into your apps. Notification Hubs eliminate complexity; you do not have to manage the challenges of push notifications. They use a full multiplatform, scaled-out push notification infrastructure, and considerably reduce the push-specific code that runs in the app backend. In addition, they implement all the functionality of a push infrastructure. Devices are only responsible for registering Platform Notification System (PNS) handles, and the backend is responsible for sending platform-independent messages to users or interest groups.

Services used

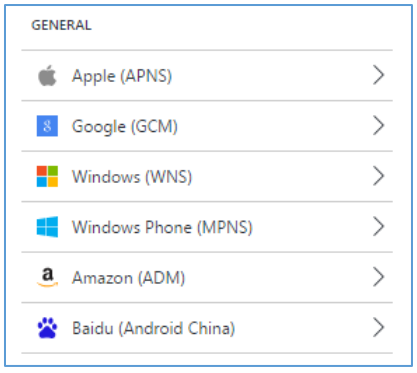
- Azure Notification Hubs

Getting notifications out to your users and their mobile devices requires three engineering sections:

- Vendor specific configuration for push notifications
- Notification Hubs in Azure
- Notification support in your mobile app

Each of the major mobile OS vendors—Apple, Google, and Microsoft—require that you register your mobile application with them as well as register for push notifications. Each vendor documents the specifics at their developer portals to do this so your app can use each vendor's PNS. In addition, the Microsoft Azure documentation covers specific details. See the additional resources section for links.

Once you've got the required tokens and keys for working with the vendor PNS, you can define your Notification Hub in Azure. You do this from the Azure Portal. When you create your hub, you need to provide with a name, a globally unique to Azure namespace, and you need to assign it to an Azure Resource Group. Once you've created your Notification Hub, you'll need to use the Settings pane to access the Push Notification Services pane. Here you provide your vendor specific PNS information to enable the Notification Hubs infrastructure to talk to each provider. Each provider is different but the general pattern is you need to provide one or two app-specific tokens or secrets given to you by the vendor.



Once you've completed these two tasks, you can turn your attention to your client applications. Regardless of platform, the same basic concepts apply. Microsoft provides SDKs for all of the supported client platforms. Your app will need to link those libraries. Then your application needs to register at startup with Notification Hubs using the correct connection string provided in the Azure Portal. Each mobile app registers with Notification Hubs and possibly with the vendor PNS. Then you put in the code that processes the notifications and provides the user experience within the guidelines of each platform. Last but not least, your backend needs to send notifications to Azure which in turn will forward this information to registered apps on your devices. In order for you to work in parallel as a team with some folks working on the client and some on the backend, you can create test messages from the Azure Portal.

Additional resources

[Hands-on labs, sample code, and more](#) ➲

[Azure Notification Hubs](#) ➲

[Sending push notifications to iOS](#) ➲

[Sending push notifications to Android](#) ➲

[Sending push notifications to Windows Store Apps](#) ➲

Offline Data Sync

Overview

Offline data sync is a client and server SDK feature of Azure Mobile Apps that makes it easy for developers to create apps that are functional without a network connection. When your app is in offline mode, users can still create and modify data, which will be saved to a local store. When the app is back online, it can synchronize local changes with your Azure Mobile App backend. The feature also includes support for detecting conflicts when the same record is changed on both the client and the backend. Conflicts can then be handled either on the server or the client.

The problem

You have a mobile solution that needs to work disconnected from networks. In addition, you want to be able to handle conflicts when multiple users edit the same data, including when offline.

The solution

Azure Mobile Apps gives both server-side and client-side support for offline data sync and conflict resolution.

Services used

- Azure Mobile App Service

Not all mobile apps need offline data sync. However, when they do, it readily becomes apparent that there's a quite a bit of work to do. This is a clear example where Azure lets you cut to the code that's more domain-focused and less framework-focused. Offline sync has a number of benefits. It can help improve overall application responsiveness by caching server data locally on the device. It lets you create robust apps that remain active and functional when there are network issues, or even allow device users to create and

modify data when there is no network access at all. It helps you sync data across multiple devices and detect conflicts when the same record is modified by two devices. You can choose to use this feature from the start or add it later.

To get started and learn how the server and client components work together, you would want to at least create a new Mobile App Service from the beginning with a client from the Azure portal, even if it's just for learning. Once you've created a Mobile App this way, you can examine the code. Assuming you created one with a Xamarin Android client, you could download and open the solution in Visual Studio.

Once there, you'll see the sample already contains code to support offline synchronization using a local SQLite database. There are a number of help functions that help you initialize the local data store, define tables representing application types, and activity classes to support CRUD operations (Create, Read, Update, and Delete). In addition, there are methods like PushAsync and SyncAsync that let you send your changes to your data backend in Azure and pull new data down to the client.

While the sample app is focused on a To Do List, you can take the sample framework code, reference the Azure SDK, and, with the appropriate sever-side updates, use the same type of code in your existing mobile clients.

Additional resources

Mobile Apps Docs and Gateway to Tutorials ➲

Offline Data Sync in Azure Mobile Apps ➲

Scaling your Mobile App's Backend

Overview

Scale up or scale out—Azure App Service has you covered with easy-to-configure tools right within the Azure Portal. You control how your mobile app scales by adjusting settings of your App Service Plan.

The problem

You want your users to have fast access to your mobile app. You need to be able to handle temporal pressures caused by end of month and seasonal demand. The bottom line is your web app needs to be up and ready when your users need it.

The solution

Azure App Service provides what you need, when you need it with easy to set scale up and scale out options that can provide your app with elastic scale with a few mouse clicks and a slider or two.

Services used

- Azure App Service

The three Basic options give you one or more dedicated CPU cores and dedicated memory, along with the ability to specify custom domains. With the Free plan, you have to use the azurewebsites.net domain. In addition to allocating up to four cores, you can also scale out, with up to three machines with four cores each. With the Standard plans, you can scale out to up to ten machines in a web farm, along with support for using SSL with your custom domains. (As mentioned earlier, even free web sites support SSL automatically, but you have to use the azurewebsites.net domain if you want SSL on the Basic, Shared, or Free plans.) The Standard plan also offers automated daily backup of your site, along with slots to support staged deployments with rollback. The Standard plan also offers support for Azure's Traffic Manager, enabling you to deploy to multiple sites across the world and direct incoming requests to whichever is nearest to the user. Finally, the Premium plans allow you to scale out to up to twenty instances, with multiple automated backups every day (up to fifty), as well as offering up to 250GB of filesystem storage and support for BizTalk Services.

As you dig in and move up out of the free plan, you can choose different "scale by" formats: CPU Percentage, manual instance count, and schedule and performance rules. The high-end options will vary based on the plan level you select. The first two options provide "large" levers to adjust the app service scaling properties either by simple CPU percentage load—add more power when the CPUs are taxed—or total instances which simply sets a hard-coded upper bound based on count.

You'll find the schedule and performance rules provide more granular control. You can create your own rules and create a schedule that adjusts your instance counts based on time and performance metrics. You can configure auto-scaling rules for different performance metrics, including CPU, memory, disk queue, HTTP queue, and data flow. One other feature that comes in handy is that you can send emails to administrators when scale actions kick in.

Also, don't forget: If your mobile app service uses a SQL Database, the sizing choice you made when you configured it can affect your application's overall performance, too. The next section covers how you can mitigate that.

Additional resources

Scale up an app in Azure App Service ➲

Hands-on labs, sample code, and more ➲

Caching for Performance

Overview

Data persistence takes many forms. Modern mobile apps can be viewed in two pieces. You have your mobile client apps installed on devices. You then have your backend that processes application logic, provides large scale data persistence, and interaction with other systems. Relational databases like SQL Server, traditional file repositories, NoSQL databases, and in-memory caches are all useful tools to have as your application's needs change. Adding caching support is one of the easier ways to enhance your application's backend performance.

The problem

Hitting the database or other persistent store every time information is needed by a mobile app's backend can be expensive. It slows down the client app as it waits for the service layer to process the request and return. This can cause your application run time costs to go up needlessly and negatively impact the user's experience.

The solution

Azure Redis Cache provides high throughput, consistent low-latency data access to power your fast, scalable Azure-hosted web app.

Services used

- Azure Redis Cache

Azure Redis Cache is a high-performance, memory-based cache that can significantly improve performance and scalability when you have data that needs to be accessed very frequently. Azure Redis Cache is based on the popular open-source Redis cache. It is accessed over the network, so it can be shared by multiple machines in a web farm if you choose to scale out your service. This makes it more flexible than simply caching data in a web server's memory; if you update a Redis cache entry, everything will have access to the new value. By using a Redis cache, you can take significant load off persistent storage systems such as SQL Database.

To use Azure Redis Cache, you need to first add it to your web app via the Azure Portal. You need to provide a DNS name that is globally unique. As with other Azure assets, you'll want to add it to the correct Resource Group and location used by your web app. Finally, you'll need to determine your pricing tier. Once you've done that you can turn to your mobile app.

As with many other technologies, you add support to access Redis Cache to your application by importing the relevant NuGet package. The most common one used is the Stack Exchange Redis package. This package provides your application with access to the Redis Cache Connection Multiplexor and other objects necessary to cache your application's data. Once you've worked out the areas that need caching, you need to add the Redis Cache connection information to your app so it has access to the cache when you deploy your updated version to Azure.

Additional resources

How to Use Azure Redis Cache ➲

Hands-on labs, sample code, and more ➲

Develop, Distribute, and Beta-test your Mobile Apps

Overview

HockeyApp is a service for app developers that helps you during the development process by providing SDKs and services that include the management and recruitment of testers, the distribution of apps, and the collection of crash reports. Today, HockeyApp supports apps on iOS, Android, OS X, and Windows. Crash reports and user metrics are working on all those platforms. Beta distribution is fully functional on iOS, Android, Windows Phone, and Mac OS X. The service is under active development and continues to evolve and expand. In particular, similar features found in Application Insights for web apps are coming to HockeyApp.

The problem

Nobody wants their app to crash, but when it happens, you want detailed crash reports and information quickly. In addition, when users are using your app, having metrics helps you understand how your app is used and helps focus your development efforts. Even if you can get anonymous metrics, getting feedback directly from your customers can be very valuable.

The solution

HockeyApp helps with all of these issues. It will create debug symbols and group similar crashes to help you easily understand their frequency and prioritize your backlog. Plus, integrate the crash reports with your existing DevOps tooling to manage all of your work items in one place. It supports usage metrics so you can see what features your users use most and it enables in-app feedback collection. All of this information and data is easily managed from the HockeyApp portal. HockeyApp even helps you distribute your apps for testing and provides specific support for running beta tests.

Services used

- HockeyApp

To get started with HockeyApp, you access the HockeyApp portal and create an account. You can work with two mobile apps for free. After that, you'll need to find a plan that fits your needs. Once you've registered for your account, you then generally install the HockeyApp on your test devices and/or emulators. You need to do this so you can authorize the devices to receive test versions of your app via the HockeyApp portal. If you do this right away, you'll be ready to go, but you won't have any apps to install yet.

The next step is to publish your app via HockeyApp. This can be done manually—which is how all the major vendors make you publish to their app stores—or you can automate the process via a DevOps process. Visual Studio Team Services, for example, supports a Team Build and Release Management task that will help you publish the output of a build to HockeyApp in a continuous delivery pipeline. You can also integrate with Jenkins, Travis CI, and more. Once you've published the app, it will show up in the mobile HockeyApp on your test devices and you'll be able to install and run the app.

You can do all of this with no code changes. However, HockeyApp shines once you add the SDK to your mobile app project and integrate support for feedback, metrics, and crash reporting. There are three categories of User Metrics HockeyApp collects and will provide to you. Basic User Metrics help you understand the adoption of your app and app versions. Custom Events tracking (as of June 2016, this is a Preseason (beta) feature) allows you to better understand how customers are using your app. Quality and performance metrics (such as users impacted by crashes) makes sure that you ship a high quality app. In order to get these benefits, you need to need to register your app's

unique ID at startup via the HockeyApp API. Once you do this, you use the API to access the FeedbackManager class from your code that allows you to integrate the HockeyApp feedback user experience. With the API, you can then register the crash manager and update manager components as you see fit with a couple lines of code.

Additional resources

HockeyApp Overview

Hands-on labs, sample code, and more

API Management

Overview

Azure API Management provides mechanisms to monitor and control access to an API, including the ability to prevent individual users from abusing the API by imposing rate limits. It provides you with a Developer Portal for your API that developers who wish to use your application's services can log into to discover information about the API. It can consolidate multiple backend services behind a single public-facing API. It also has a variety of options for authentication: it can impose various authentication requirements on API users, and it can also pass credentials through to the underlying service. It also provides a Publisher Portal which you, as the owner of an API, can use to view analytics to discover how your API is being used.

The problem

You have an API that is used by your organization's own apps. However, partners need access to your backend for their mobile apps and you need better control and management functions that you get by exposing your existing API on its own.

The solution

Azure API Management allows you to create an API Management instance, wrap your existing API up, and then allow partners to access your services with you in control.

Services used

- Azure API Management
- Azure App Service

The first thing needed for working with API management is, in fact, an API. That said, if you have a mobile app talking to Azure App Service, you by default have some sort of API. Once you've got that, jump to the Azure portal and create a new API Management resource. As with other Azure resources, you need to provide a globally unique name that becomes a part of the API Management App's URL. You'll also need to pick a region. Lastly, you need to provide your organization name and an administrator email address.

After you've done this, you can move to wrap your API up with API Management. Still using the Azure portal, you'll import your API. API Management supports multiple ways to "read" your API into the system. These include cut and paste from the clipboard, from a text file, or a remote URL that exposes information via WADL or Swagger. Next you'll want to configure security on your API.

Security can involve two mechanisms. First, you might want to secure your API so only API Management can access it. You can secure your API via mutual certificate authentication. You could even support VPN access to an on-premises API using ExpressRoute. Second, you want to control who has access to your API via API Management. For this, you can use OAuth 2.0 with Azure AD or Open ID Connect. Both options require you to have registered your app with Azure AD. You use the Azure AD and API Management portals to share secrets and specify how long the relationship is valid for—a year or two. Next, you can test your configuration from portal.

You will most likely want to create a test sample to make sure everything works as expected. Once in use, you can inspect API usage information in the Publishers Portal.

Additional resources

Hands-on labs, sample code, and more ➲

API Management Documentation ➲

Conclusion

Building a rich mobile experience means building great mobile apps for users to put on their devices but also a backend that will scale, perform under pressure, and be backed by an organization that's not going to leave you hanging. In this guide, we've explored how Azure App Service helps you save time and cut to the code. We've highlighted Azure's support for mobile apps, both client and server, and the time saving features to help you deliver a fantastic experience for the consumers of your app.

Cut right to the code today. Welcome to Azure.

Recommended Next Steps

- Download the hands-on labs, sample code, and ARM templates to cut right to the code.
- Create an Azure account and get started for free with $200 in Azure credit.
- Explore the range of free options available to get you started, like hosting up to ten free web and mobile apps on Azure App Service, Azure Search to add sophisticated search capabilities, and Visual Studio Application Insights to monitor live apps.
- Be our guest for up to an hour of Azure App Service experience with no subscription, free of charge and commitment.