



# TOP 10 CHARACTERISTICS OF ENTERPRISE BACKEND SERVICES (BaaS/MBaaS)

# Table of Contents

Introduction . . . . .	3
Characteristic #1: Common Capabilities. . . . .	4
Characteristic #2: Flexible Hosting and Deployment . . . .	6
Characteristic #3: Easy App Design . . . . .	7
Characteristic #4: Easy Data Modeling . . . . .	8
Characteristic #5: Enterprise Integration . . . . .	10
Characteristic #6: Code Generation and Portability . . . .	11
Characteristic #7: Monitoring, Debugging & Testing. . . .	12
Characteristic #8: Building, Deployment and Versioning	13
Characteristic #9: MDM and MAM Integration . . . . .	15
Characteristic #10: Smart SDKs . . . . .	17
Conclusion. . . . .	20

## Top 10 Characteristics of Enterprise Backend Services (BaaS/MBaaS)

*“80% of mobile app development projects will leverage an MBaaS platform by 2016”*

- Gartner

*There are plenty of options available for Mobile Backend as a Service, but not all of them will meet the needs of an enterprise.*

If you search the web for “MBaaS” you’re likely to get over 200,000 results. If you search for “mobile backend as a service” you’re likely to get over 4 million results. If you read through the first few pages of either set of search results, you’ll see that most of them refer to a dozen or so MBaaS solutions. How do you differentiate those solutions, and decide which ones are worth evaluating for your enterprise?

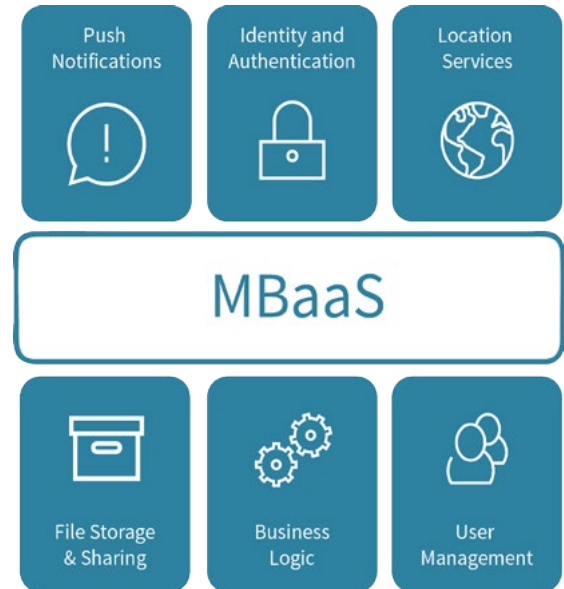
## #1 Common Capabilities

If you dig deeper, you'll see that the capabilities and pricing of MBaaS solutions vary significantly, although a common set of capabilities found in almost all of them include: push notifications, file storage and sharing, integration with social networks, location services, messaging and chat functions, user management, the ability to run business logic, and usage analysis tools.

You're going to need all of those services for almost any mobile app you build for your business, although for enterprise apps, integration with corporate identity systems based on protocols such as LDAP and SAML are often as or more important than social networks. In addition, you're going to want those services to be available day in and day out, as today's mobile app users have no tolerance for down time, error messages, unavailable services, hung apps, or even slow response times.

The first key enterprise MBaaS characteristic is a robust implementation of all the common mobile backend services.

What do all those services do, and why are they important to mobile applications?



» Figure 1: Common Mobile Backend Services

### Characteristic #1:

Robust implementations of the common mobile back-end services: Push notifications, file storage and sharing, integration with social networks, location services, messaging, user management, the ability to run business logic, and usage analysis tools.

**Push notifications** allow mobile users to receive messages on their phones and tablets from your app even when they are not using your app. The mechanism is different on each mobile platform, so extra points for MBaaS implementations that give you a unified API (application programming interface) – or at least automatically keep track of which push method to use for each registered device. Note that web apps can't receive push notifications.

**File storage and sharing** is self-explanatory. You'll need it to store and retrieve binary files, such as images and videos. Data is usually stored in a database rather than files, typically in a NoSQL database such as MongoDB.

**Integration with identity providers** such as social networks allows your app to support social media sign-up and sign-in, typically via OAuth protocol. For enterprise applications, however, it's more common to support LDAP authentication into systems such as Microsoft Active Directory.

**Location services** in an MBaaS are very useful for mapping and routing apps. All current smartphones have location sensors, and a backend that can not only store locations but also perform geographical queries is a bonus. Geographical queries usually require corresponding support in the database, such as Oracle, SQL Server, or Postgres.

**Messaging and chat functions** allow you to communicate with your mobile users when they are using the app, and allow the users to communicate with each other. While all smartphones support text messaging, it's better,

more convenient, and sometimes cheaper for business users to communicate within the app.

Even if an app authenticates users with LDAP, an enterprise app will want to **manage user roles, permissions, and devices**. Support for this in the MBaaS can greatly reduce the need to write code for user management.

Of course, it's rare that any application can be created without writing server-side code. The ability to **run business logic on the server** is essential for any MBaaS worth the name; it helps if that business logic can easily work with stored data in the MBaaS and in systems of record as well as data from the mobile device.

It also helps if your developers already know the programming language(s) supported by the backend server for business logic. Common choices are JavaScript, either in an actual node.js environment or one that is similar, and Ruby on Rails.

From a management viewpoint, **usage analysis tools** are essential for the backend of mobile applications. You need to be able to track signups, logins, and often the usage of specific APIs over time. How much data did users store, and how often was it read? How many push notifications went out in a given month? Did responses to push notifications vary by the time of day and/or the day of the week? Ideally, you will be able to customize the type of logging you need with backend logic, and export the data for analysis in your analytics reporting tool of choice.



## #2 Flexible Hosting and Deployment

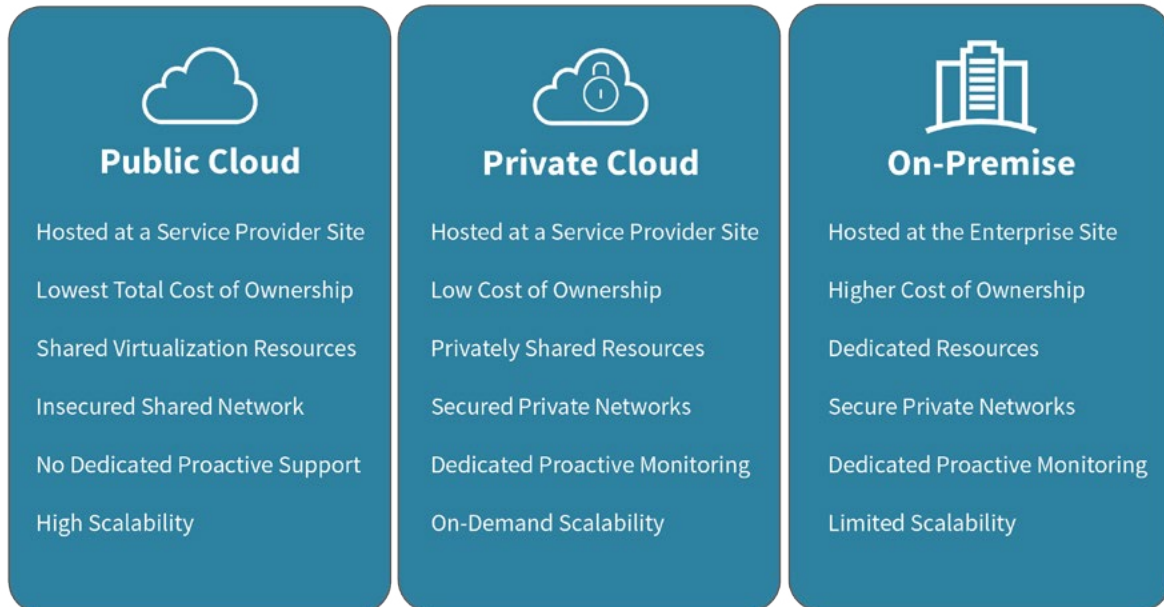
An MBaaS does you little good unless it can handle all of your users, keep your data private, respond quickly to your app, and meet all the regulatory requirements that apply to your app and your data. That leads to our next key enterprise MBaaS characteristic:

### Characteristic #2:

Flexibility to support scalable, secure, high-performance hosting on all major public clouds, in private clouds, on-premise, and on clouds that conform to HIPAA, PCI, FIPS, and EU data security standards.

“Lock-in” is sometimes used as a bogeyman by vendors to scare you away from other vendors. However, you really don’t want to be locked into a single cloud for an enterprise backend. There are many reasons for this, including legal requirements for data privacy and business considerations for hosting costs.

For example, suppose your company expands to European Union customers: all of a sudden, you will need to keep European personal data in Europe, to avoid the difficult “third country” (meaning non-E.U.) data protection tests in EU Directive 95/46/EC. In that situation, if your MBaaS only runs on a public cloud with no data centers outside the U.S., you’ll have your work cut out for you rewriting your app and backend for a different MBaaS.



» Figure 2: Public vs Private Clouds vs On-Premise Deployments

## #3 Easy App Design

In today's brave new world of mobile apps, there's a plethora of target devices and technologies: native iOS (iPhone, iPad, and iPod), native Android, mobile web, and mobile hybrid. There are also mobile device management systems for secure native apps. While there are other, less popular native devices, they can all run mobile web and mobile hybrid apps. (Mobile hybrid apps run a mobile Web user interface inside a native shell, and are usually built with Adobe PhoneGap or Apache Cordova.)

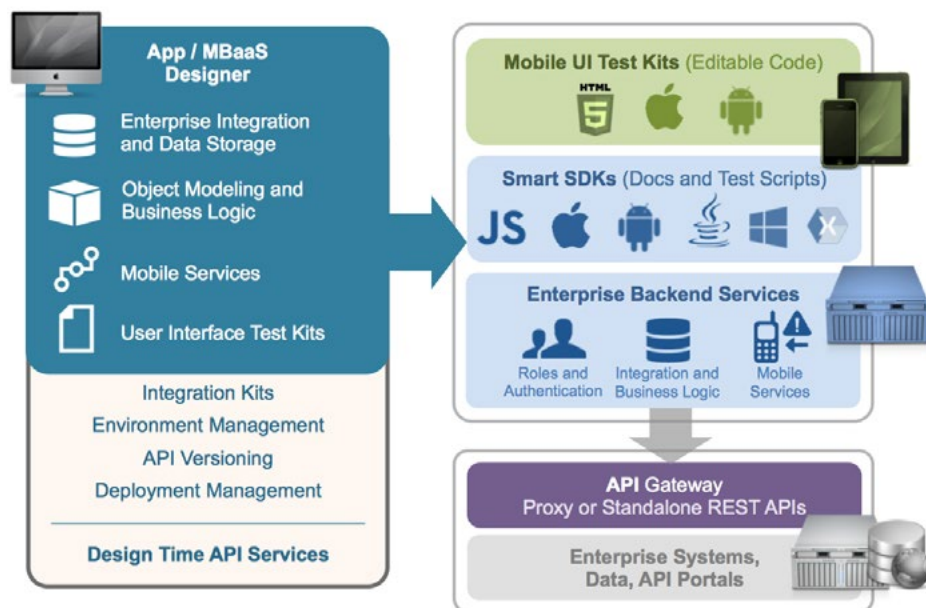
A good MBaaS will support all of your desired mobile clients without requiring a lot of work. An ideal MBaaS will help you to generate working apps once you've defined a basic user interface.

Both web-based and desktop-based mobile app designers exist. Most reduce app UI design to a drag-and-drop process; some also help you to generate your local device code and backend service calls.

A few app designers add a useful abstraction layer by generating an SDK (software development kit) to be called from the UI as well as generating a prototype UI that consumes the APIs from the SDK. Having that layer makes it easy to modify the UI and add app features later on without necessarily returning to the app designer.

### Characteristic #3:

Easy design of native, hybrid, and mobile web apps that connect to the MBaaS, ideally with starter user interface kits that show end-to-end connectivity between the app, SDK, and backend server.



» Figure 3: Example Enterprise MBaaS Architecture for Design and Run-Time Separation

There are business and technical advantages to each type of mobile client. Native clients typically have the best performance and the slickest look and feel, but require the most programming time, the highest level of skill and are inherently restricted to one class of device.

Mobile web apps run the same code on almost every smartphone and tablet and take the least time and skill to program, but they can lose app state when a user switches to another app, unless the programmer is careful about saving and loading state. Mobile web apps often don't run as fast as native apps, and don't have a native look and feel unless the programmer knows how to simulate that, or uses a framework that generates a native-looking UI.

Hybrid mobile apps require an intermediate amount of programming time and skill, and do not lose app state on a context switch. Otherwise, hybrid mobile apps have essentially the same advantages and disadvantages as mobile web apps, assuming you use a hybrid shell that is well supported on all your target devices.

## **#4 Easy Data Modeling**

Many app designs hinge on three questions:

1. "What data do you need to see?"
2. "What data do you need to capture?"
3. "How do you need to process your data?"

### **Characteristic #4:**

Easy data modeling and programming of business logic, including abstraction/ORM layer for composite apps, user roles, and permissions.

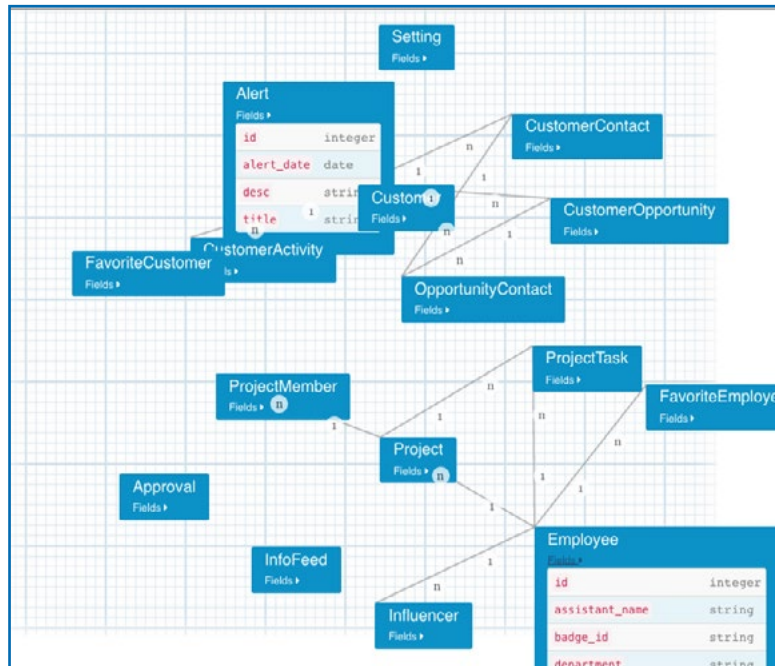
These three primary questions lead to additional questions that drive your data modeling:

4. "Which data should live together in any given table?"
5. "What are the one-to-many and many-to-many relationships between tables?"
6. "Where should the data be processed?"

Often, the data modeling drives the designs of your mobile app's screens. For example, a shopping app for a clothing vendor will have categories of goods, such as men's, women's, hats, coats, and shirts. Within each category are many items, and each item may be available in many sizes and colors. For each unique, fully-qualified item, such as a large green men's long-sleeved sweatshirt, there needs to be a price and an inventory count for each store location and each warehouse. You can see how the structure of the shopping app needs to flow from the structure of the data relationships.



You want as much as possible of that built into your backend, you want the data access to be abstracted to allow for future portability, and you want permissions to be enforced at every step. For example, it should be easy for the app to access the correct pricing data for display, and impossible for the customer to change the price.



» Figure 4: Data Modeling within an MBaaS Designer

You want the data-modeling designer to be easy to use, to translate the design to implemented database schemas, and you want the schemas to automatically propagate throughout your app as far as possible.

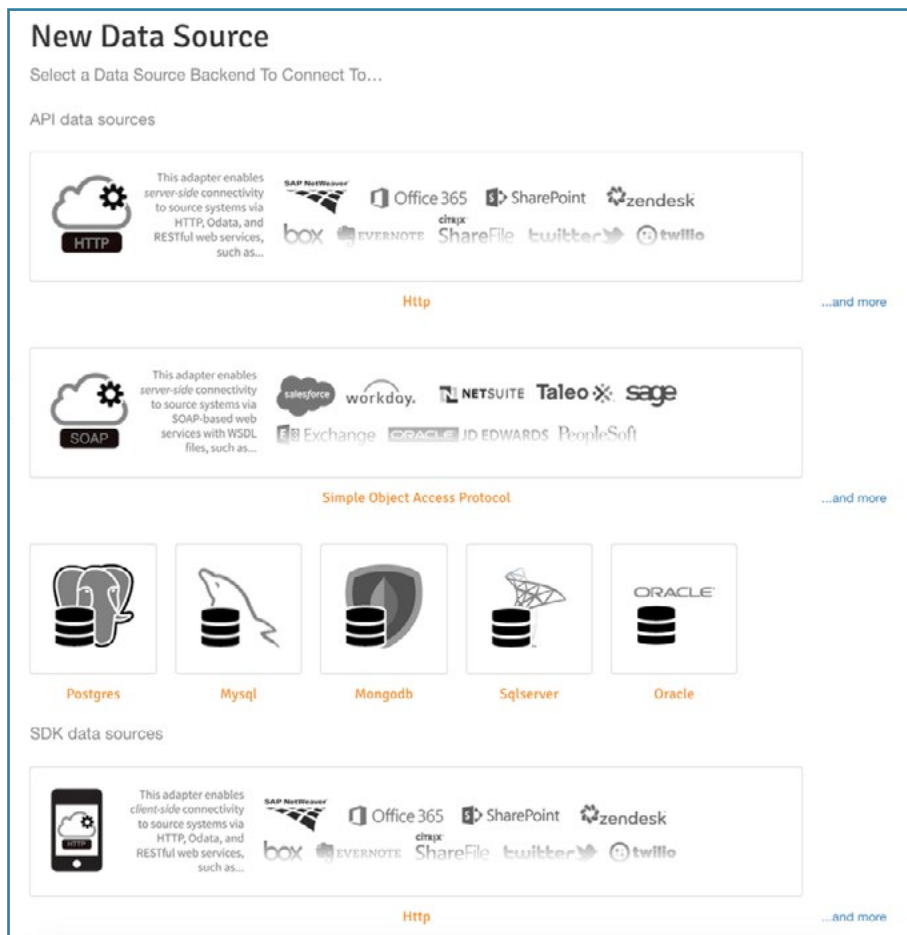
One good way to make the schema propagate throughout the app is to generate customized classes (see #10) to provide CRUD (creation, retrieval, update, and deletion) functionality for your actual schemas. It's much easier and more maintainable, and results in a much cleaner separation of functionality, if you can write mobile client code that essentially says “read the inventory for the currently viewed stock-keeping unit” rather than writing code that embodies the current navigational details of the location of that data.

## #5 Enterprise Integration

One of the big differences between enterprise apps and consumer apps is that enterprise apps almost always need to integrate with systems of record, including relational databases such as Oracle, enterprise applications such as SAP, and identity management systems such as Active Directory. It is rarely the case that enterprises want to rip out and replace such systems. If your MBaaS doesn't already integrate with your systems of record, writing the necessary wrappers and interfaces can be time-consuming for your developers.

### Characteristic #5:

Convenient integration of the MBaaS with all major back-end enterprise applications and databases, support for common identity management systems and authentication protocols.



» Figure 5: Examples of Enterprise Data Integration Capabilities

## #6 Code Generation and Portability

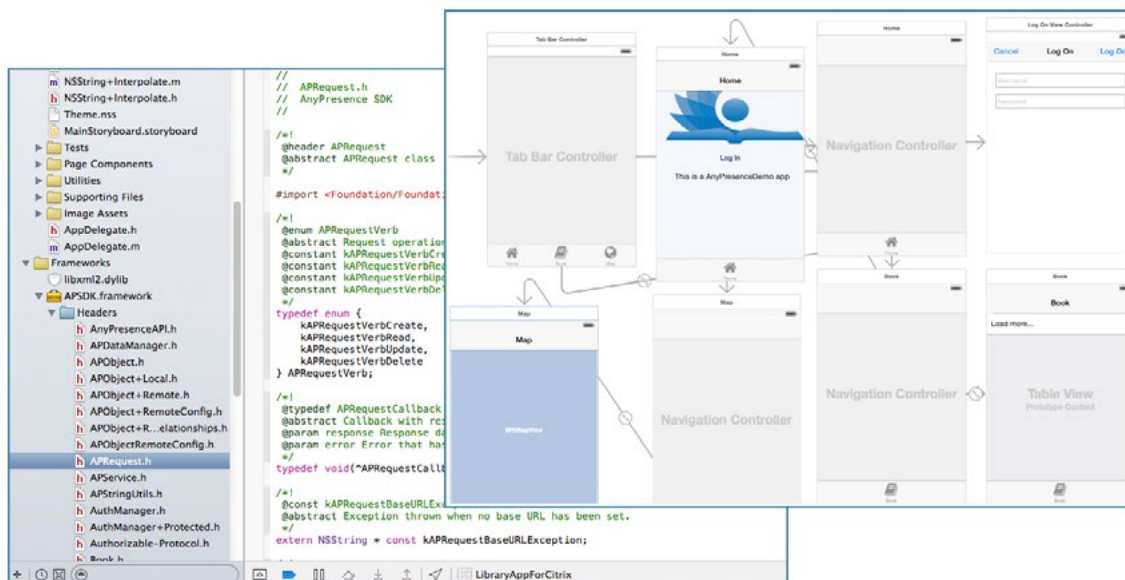
It's not enough for a good MBaaS to allow you to model your data and design your app's user interface, data flow, and user experience: It should also generate the code, for at least a prototype app. Ideally, it should generate the best possible code for each mobile device that you want to target using your desired technology, be that native code, HTML5/CSS3/JavaScript, or code for a specific framework. Of course, you may not always get exactly the code you would write yourself, but there are a couple of things you need to ensure about the generated code.

First of all, the code must be maintainable. You need the generated code to be understandable by mere mortals, and structured so that a developer who has to implement a change long after the original programmers have moved on can figure out where to look for any given part of the functionality.

Second, you need the generated code to be portable. If it's JavaScript, you need it to work on every major standards-based mobile and desktop browser, at least for the most current version of the browser. If it's native code, you need to be able to compile it using the native toolset for the platform – Eclipse and/or Android Studio for Android code, Xcode for iOS, Visual Studio for Windows Phone, Xamarin Studio for mono code.

### Characteristic #6:

Generation of easy-to-maintain, portable code for all desired mobile device targets.



» Figure 6: Editable, Human-Readable Client-Side Source Code

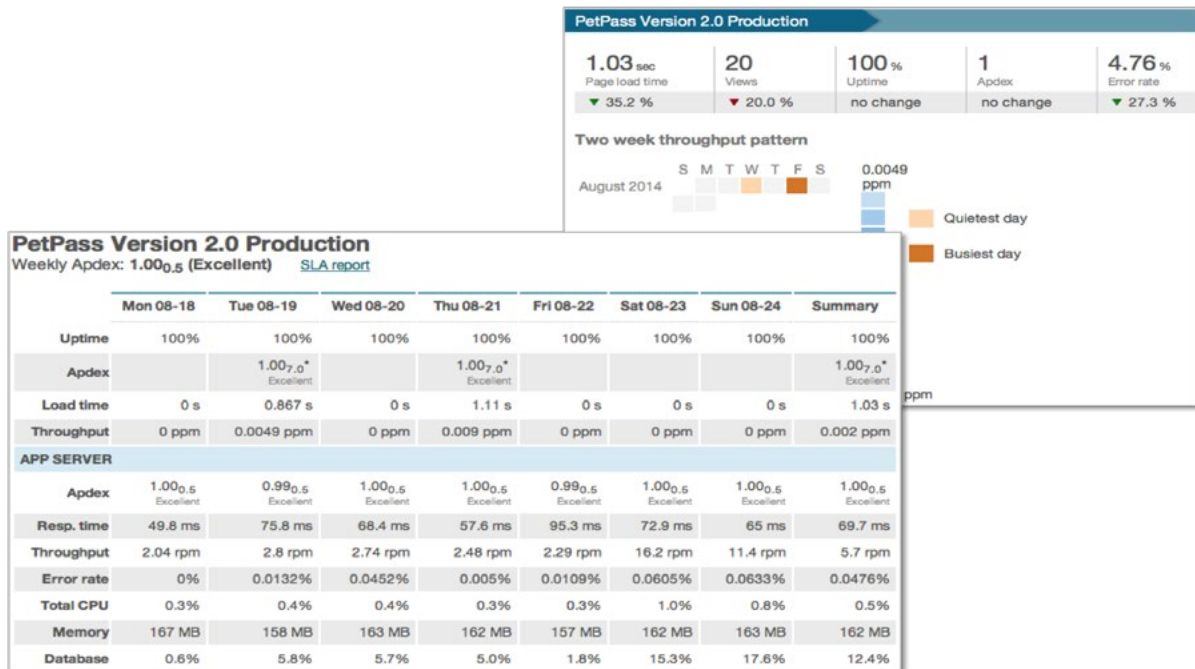
The generation of maintainable, portable code should be the end product of the design process discussed in characteristics #3 and #4. The ultimate test in source code portability is, of course, whether the vendor will let you own and maintain the run-time source code of your app, without continuing to subscribe to their service. This applies to the backend server code as well.

## #7 Monitoring, Debugging and Testing

Of course, coding is just the beginning of the application lifecycle. You'd like to think that code generated automatically from a clean design would be inherently bug-free, but, alas, the world doesn't always work that way. Code always needs to be tested on every change and monitored in production. If either testing or monitoring flags an error, then the code will need to be debugged.

### Characteristic #7:

Integration with monitoring, debugging, and testing tools.



» Figure 7: Example Integration with Best-in-Class Performance Monitoring Tools (New Relic)

If you don't already have monitoring, debugging, and testing tools, then ideally your MBaaS will include them. If you do have such tools, then ideally your MBaaS will integrate with the monitoring, debugging, and testing tools that you own, know, and love.

An ideal enterprise MBaaS will offer interfaces to all the systems of record with which you need to integrate, and will reduce the mapping problem to a matter of entering the system credentials and picking the fields of interest to your app from the available fields.

## #8 Building, Deployment and Versioning

Building and deploying mobile apps can sometimes be a pain in the neck, for various reasons. Often, each different mobile platform has its own special compilers, which may require special platforms. For example, you can only build native iOS apps (for iPhone, iPad, and iPod) on Mac OS X platforms, using Xcode. You can only build native Windows Phone apps on Windows platforms, using Visual Studio. People who want to build both often run Windows and Visual Studio in a virtual machine on a Mac OS X host. Android, fortunately, has operating-system-agnostic development tools.

### Characteristic #8:

Easy building, deployment and versioning of enterprise mobile applications.

Even running the builds can be difficult, if you target multiple mobile clients. For a complicated app, each target build might take five minutes, and the build might have to be run sequentially because of the RAM and CPU limits of your development computer.

Just maintaining all those development systems and keeping them up to date can be a chore. In addition, you have to keep your code current as the underlying system APIs change with new versions.

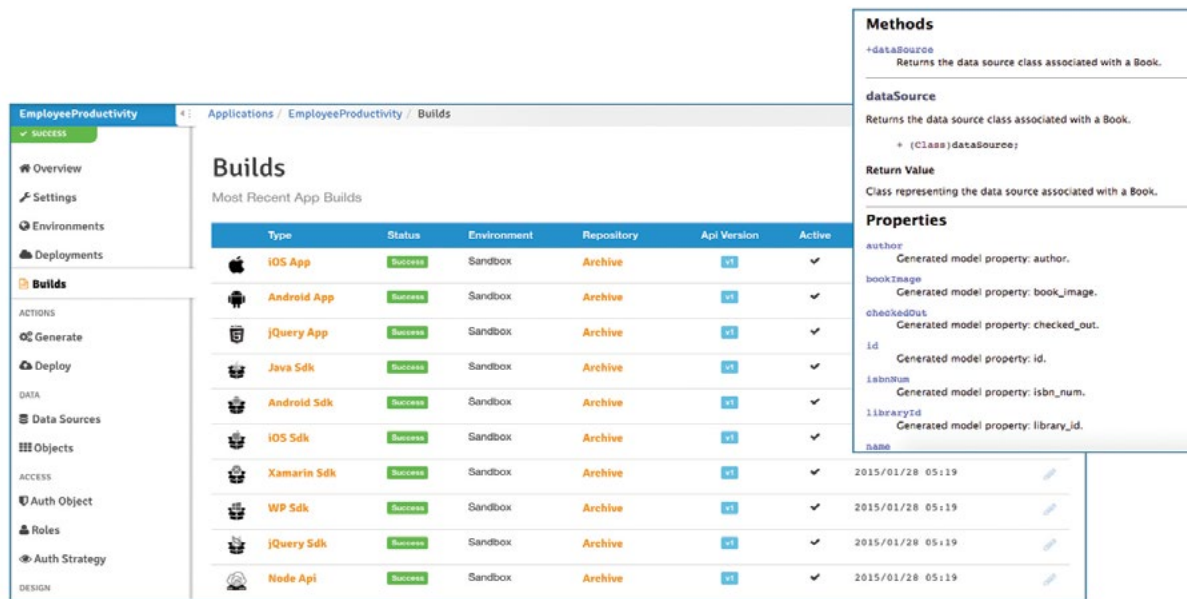
Ideally, your MBaaS will take care of most if not all of these issues for you. Some mobile tool sets and mobile backend services have their own build servers or build clouds; others integrate with Adobe PhoneGap Build, which provides a build cloud specifically for hybrid apps built using a PhoneGap wrapper. Using cloud build services speeds up the build cycle, takes the build load off your computer, and relieves you of maintaining all the different build systems.

Deployment to public app stores can be time-consuming and problematic, particularly the Apple App Store, which currently has an average 9 day review time for submissions, and a spotty record for consistency of approvals. While it helps a bit for the MBaaS to automate deployment, it helps even more if you deploy to enterprise app stores, especially ones that integrate with mobile app management platforms. (See #9)



Versioning of apps ought to be nearly automatic, controlled either in the IDE or in the MBaaS control panel. Often MBaaS platforms either provide their own version control system, or integrate with a popular external source code control system such as GitHub, and propagate the version to the deployment points as well as creating tags in the repository.

In addition to client-side deployment considerations, also note the importance of server-side deployment flexibility as discussed in characteristic #2. The ability to deploy the backend server in any cloud, and more importantly for enterprises, on-premise, is a key characteristic for enterprise MBaaS solutions.



Type	Status	Environment	Repository	Api Version	Active
iOS App	Success	Sandbox	Archive	v1	✓
Android App	Success	Sandbox	Archive	v1	✓
jQuery App	Success	Sandbox	Archive	v1	✓
Java Sdk	Success	Sandbox	Archive	v1	✓
Android Sdk	Success	Sandbox	Archive	v1	✓
iOS Sdk	Success	Sandbox	Archive	v1	✓
Xamarin Sdk	Success	Sandbox	Archive	v1	✓
WP Sdk	Success	Sandbox	Archive	v1	✓
jQuery Sdk	Success	Sandbox	Archive	v1	✓
Node Api	Success	Sandbox	Archive	v1	✓

**Methods**  
+dataSource  
Returns the data source class associated with a Book.  
  
**dataSource**  
Returns the data source class associated with a Book.  
+ (Class)dataSourceee;  
  
**Return Value**  
Class representing the data source associated with a Book.  
  
**Properties**  
author  
Generated model property: author.  
bookImage  
Generated model property: book\_image.  
checkedOut  
Generated model property: checked\_out.  
id  
Generated model property: id.  
isbnNum  
Generated model property: isbn\_num.  
libraryId  
Generated model property: library\_id.  
name

» Figure 8: Build and Version Management with Github Integration

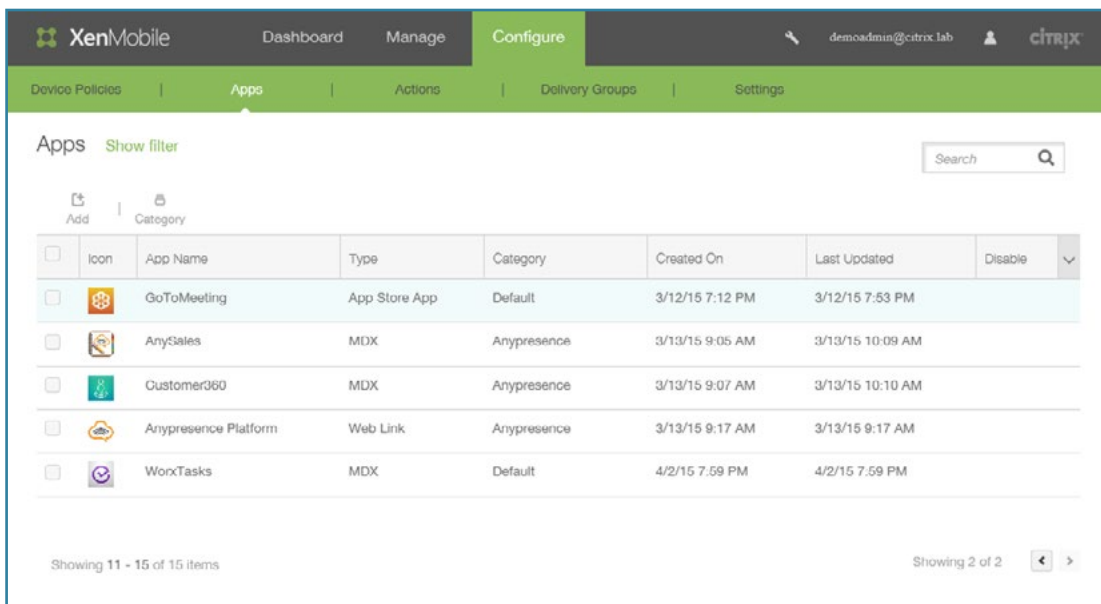
## #9 MDM and MAM Integration






Mobile device management (MDM) and mobile application management (MAM) systems help enterprises protect corporate data, control user access to network resources, and distribute apps without going through public app stores. MDM often refers to systems with deep control of the mobile devices, down to the firmware level; MAM now usually refers to app-oriented systems that control provisioning and updating of apps through an enterprise app store, although some MAM systems use secure app wrappers.

If your company uses MDM and/or MAM, you'll want your MBaaS to integrate with those services automatically. Otherwise, you will need to either implement your own integration based on the methodology employed by your MDM/MAM vendor(s), or live with some manual steps in your deployment workflow.

### Characteristic #9:

Integration with enterprise MDM and MAM solutions for data management, user management, and app distribution.



<input type="checkbox"/>	Icon	App Name	Type	Category	Created On	Last Updated	Disable	
<input type="checkbox"/>		GoToMeeting	App Store App	Default	3/12/15 7:12 PM	3/12/15 7:53 PM		
<input type="checkbox"/>		AnySales	MDX	Anypresence	3/13/15 9:05 AM	3/13/15 10:09 AM		
<input type="checkbox"/>		Customer360	MDX	Anypresence	3/13/15 9:07 AM	3/13/15 10:10 AM		
<input type="checkbox"/>		Anypresence Platform	Web Link	Anypresence	3/13/15 9:17 AM	3/13/15 9:17 AM		
<input type="checkbox"/>		WorkTasks	MDX	Default	4/2/15 7:59 PM	4/2/15 7:59 PM		

Showing 11 - 15 of 15 items

Showing 2 of 2

» Figure 9: Example Integration with Best-in-Class MDM and MAM Tools (Citrix XenMobile)

Secure app wrappers typically add several capabilities to the app packages they contain: remote wipe, secure connectivity, authentication, and encryption. Since the app wrapper mediates all communications for the app, it can force all connectivity to go through a VPN, and provide single sign-on. It can also automatically delete any local file storage created by the app when the app closes, if that's the applicable policy. Secure app wrappers are often used in addition to an enterprise app store, since secure enterprise apps really have no business being listed in the public app store.

Secure apps without secure app wrappers often use a MAM SDK to implement single sign-on, remote wipe, secure connectivity, authentication, and encryption. Your MBaaS may already have support for selected MAM SDKs, or you may have to mix the MAM API calls into your mobile app code.

Enterprise app stores offer a single place to go for apps for all platforms used by the company, and give companies an easy way to apply role-based security to authorizing internal, partner, and external app users to download, install, update, and delete sensitive apps. Enterprise app stores are sometimes deployed on-premise, and sometimes deployed as private sites in a public cloud.

It simplifies your deployment workflow if your MBaaS knows how to deploy directly to your enterprise app store. However, your worst case would require you to manually upload your app packages to the enterprise app store whenever you need to release a new version.

## #10 Smart SDKs

As we discussed earlier, code generation that creates SDKs as well as prototype apps that consume the SDKs makes it easy for developers to build other apps against the same SDKs. The more common alternative leaves out the SDK layer and simply generates apps. While that certainly makes developers more productive than building every app from scratch, it doesn't support an ecosystem of departments or partners using your centralized APIs.

For example, your business might have an ERP system that includes your suppliers. Historically, you might have allowed your suppliers to connect to your ERP system through your web site, but now they want to allow their staff to connect and find out your needs, as well as the needs of their other major customers, using their tablets from their factory floors. If you simply create an app, they'll have to switch between your app and their other customers' apps; if you create an SDK, they will be able to include your information along with other customers' in their own app.

A similar use case for SDKs applies to enterprises with multiple subsidiaries, divisions, departments and locations. The French agricultural sales operation will want a different app than the Texas agricultural sales operation, not just because of speaking a different language, but also because of different currencies, tax regulations, and product lines. At that point writing a single app for both operations no longer makes sense, but writing two apps using the same SDK to communicate with the common backend does – and the two apps will be much more maintainable than they would have had they each been written from scratch.

Another benefit of generating an SDK comes from the use of classes. Once you have the SDK classes imported into your IDE, you'll have the benefit of code completion for all the members of the classes, meaning that the IDE will help you write correct code based in a few keystrokes. (see figure 10)

One of the hardest problems in enterprise app development is offline operation. You would expect to need to support offline operation for a sales app, as you expect the salesperson to go out of cellular and Wi-Fi coverage regions at times. You might not expect to need to support offline operation for an order fulfillment app, but many warehouses have no cellular reception at all, and spotty Wi-Fi that can vary from day to day as box placement changes.

### Characteristic #10:

Support of SDKs for use by departments and business partners. SDKs should have rich features including support for offline cache, data synch, and ideally classes based on object definitions to support type-ahead IDE features.

If your inventory changes slowly, you can load your salespeople's tablets with the current numbers before they go on the road every day, and store that in offline cache, typically using local storage on the device. Periodic updates when the tablets have connectivity might keep the local copies of the inventory sufficiently accurate, but that begs the question of how to handle order entry.

The worst case for the salesperson would be having no connectivity at the point of sale – not an uncommon situation. The customer's order would be entered on the tablet, but couldn't be transmitted back to the company until the tablet went online again, so it would have to be stored locally. If the app didn't have that capability, the salesperson would be reduced to taking the order on paper.

In the meantime, however, another salesperson could have sold some of the same items and transmitted the order back to the company, making the first salesperson's local inventory numbers inaccurate, possibly changing some items from in-stock to out-of-stock status. When the app goes online and tries to complete the transaction, some items may be marked as back-ordered. That would lead to an apologetic phone call from the salesperson to the customer, and possibly a canceled sale.

Ideally, the MBaaS would generate most of the online-offline data synchronization and conflict resolution logic. The appropriate conflict resolution strategy depends on the use case, however, and a backend vendor that airily tells you “server wins is always the answer” is not to be believed.



» Figure 10: Smart SDKs with Custom-Generated Native Object Classes



For example, take the case of a field inspector who has no connectivity while recording observations and taking pictures. The inspector would have gotten the location information before starting the inspection, but suppose there was a typo in that record that was fixed in the back office while the inspector's tablet was offline?

When the inspector regained connectivity, his app would try to synchronize all his observations with the central database, but the fixed typo would not match. A blanket "server wins" strategy could lead to the loss of the entire field inspection's data, while a blanket "client wins" strategy would lead to re-introducing the original error. Ideally, the app would show the inspector the data in conflict and make an informed decision, without any possibility of losing a morning's inspection data.

## Conclusion

As we have seen, an enterprise app's needs for a mobile backend as a service are more complicated than a consumer app's needs.

At the base level, a true enterprise MBaaS will include:

1. Robust implementations of the common mobile backend services
2. Flexible hosting and deployment (cloud or on-premise)
3. Easy app design
4. Easy data modeling
5. Integration with all major backend enterprise applications and databases
6. Generation of easy-to-maintain, portable code for desired mobile devices
7. Integration with monitoring, debugging, and testing tools
8. Easy building, deployment and versioning of mobile apps
9. Integration with enterprise MDM and MAM solutions
10. Smart SDKs for use by departments and business partners

Other considerations beyond these basic enterprise capabilities are:

- The emergence of non-traditional endpoints such as wearables and IoT endpoints: does your MBaaS platform enable you to take advantage of these newer technologies as they gain adoption?
- Creation of custom APIs and microservices: traditional MBaaS platforms provide prescribed or fixed API endpoints. More flexible platforms will enable you to define customized API endpoints with signatures that can take any shape or format.
- Enabling faster innovation across your enterprise ecosystem: how can your MBaaS platform enable people across or even outside your company to build innovative solutions that incorporate your digital assets

Now that you know some of the factors you need to consider, you should be able to do deeper evaluations and make more informed choices about enterprise MBaaS.