# InfoWorld DeepDive

5 CLOUDS FOR BUILDING MOBILE APPS

ART: STEPHEN SAUER/SHUTTERSTOCK COMPOSITE

Copyright © 2014 InfoWorld Media Group. All rights reserved. • **$129**

Deep Dive

# Pick one: 5 clouds for building mobile apps

BY MARTIN HELLER

The general idea of MBaaS is that mobile apps need common services that can be shared among apps instead of being custom developed for each.

**MBaaS (mobile back end as a service)** is a fairly new product category that has largely supplanted MEAPs (mobile enterprise application platforms). The general idea of MBaaS is that mobile apps need common services that can be shared among apps instead of being custom developed for each. Mobile apps using MBaaS follow a loosely coupled distributed architecture, and MBaaS systems themselves typically have more distributed architectures than MEAP systems, which tended to be unified middleware servers.

MBaaS systems typically provide push notifications, file storage and sharing, integration with social networks such as Facebook and Twitter, location services, messaging and chat functions, user management, the ability to run business logic, and usage analysis tools. Enterprise-oriented MBaaS systems also provide integration with existing applications and databases.

For extra credit, MBaaS systems can generate mobile SDKs. This is most useful when a vendor is exposing its services to partners doing mobile app development.

Back ends don't exist in isolation, so MBaaS systems provide some level of mobile client support. This ranges from exposing REST APIs to all comers to providing app generation for iOS, Android, some flavors of JavaScript, and perhaps other mobile platforms.

In addition, back ends need to be customized and programmed, so MBaaS systems provide a combination of online and desktop development environments. Finally, back-end services are intended to be in continuous operation, so they need a level of application monitoring and error logging in addition to usage analysis. Monitoring and analytics might be provided directly by the MBaaS vendor or through integration with a third-party service.

For extra credit, MBaaS systems can generate mobile SDKs. This is most useful when a vendor is exposing its services to partners doing mobile app development. In addition, MBaaS systems can support offline operation of their mobile apps and offline/online database synchronization. MBaaS systems may provide their own mobile device management or integrate with an MDM vendor. MBaaS systems may also support device-specific services where appropriate, such as iBeacon on iOS devices.

## Commonalities and differentiators

In the course of reviewing FeedHenry, Kinvey, Appcelerator, Parse, and AnyPresence, certain capabilities and implementations became very familiar. For example, all five MBaaS products provide storage using MongoDB, an open source NoSQL document database that stores JSON objects. All of these products provide a data design UI for their MongoDB data store, and these UIs all look similar. It wouldn't surprise me if the UIs were all based on the same MongoDB sample code.

All five MBaaS systems are available in a multitenant cloud. All have online documentation. All provide push notification and user authentication APIs. All support native iOS and Android apps at some level, and all have some way for developers to implement custom server logic.

The differentiators between these products are telling. For example, their support for integration with enterprise applications and databases ranges from the basic ability to call external REST interfaces that return JSON to deep integrations with common applications and databases. The time required for a developer to implement a given enterprise integration with an MBaaS ranges from days down to minutes, depending on how much of the work a given MBaaS vendor has already done for a specific integration.

Some MBaaS systems are available on-premise, and some are available in private clouds. Some can be hosted in compliance with HIPAA, PCI, FIPS, and EU data security standards. Some have their own testing capabilities, and some offer cloud builds of mobile apps.

Some support HTML5 and hybrid apps. Some compile JavaScript to native device code. Some support PhoneGap, some support Apache Cordova, and some avoid both wrappers for hybrid apps in favor of other solutions, such as generating native apps.

Some run their back ends on Node.js, some on Rails, and some on unspecified platforms. Some support BlackBerry, Windows Phone 8, Windows 8, or Unity clients.

Some have hosted app and back-end IDEs in their cloud, some provide multiplatform desktop IDEs, and some have desktop command-line interfaces for cloud control. Some support multiple popular JavaScript frameworks, such as Backbone and Angular, and some use their own JavaScript frameworks, which may be adaptations of specific open source frameworks.

## MBaaS five ways

As we'll see, the different MBaaS vendors have targeted slightly different markets and made slightly different technical choices. Nevertheless, they have a high degree of overlap and commonality.

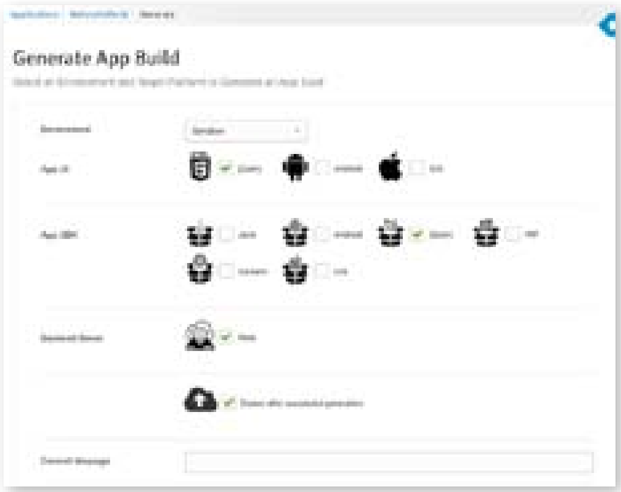
## AnyPresence

The goal of AnyPresence is not only to help enterprises build back ends for their mobile apps. AnyPresence combines app building, back-end services, and an API gateway.

AnyPresence has an online designer that generates back-end code, mobile app code, and even customized mobile API code. All the generated code can be downloaded, edited, and run on compatible platforms. To cite one of AnyPresence's favorite customer examples, MasterCard has used AnyPresence to enable partners to easily build mobile apps against MasterCard's Open API services.

AnyPresence generates app UIs (or starter kits, if you wish) for jQuery, Android (XML layout), and iOS (storyboard), and it generates app SDKs for Java, Android, HTML5, Windows Phone, Xamarin, and iOS. The design environment refers to the generated JavaScript/HTML5 SDK as "jQuery." In fact, AnyPresence actually generates CoffeeScript that uses the Underscore, Backbone, and jQuery libraries.

AnyPresence generates back-end servers for Ruby on Rails. In the future it will also generate Node.js back ends, which will be a good development. The AnyPresence environment can generate deployments to Heroku (usually for a Rails back end) to Amazon S3 (usually for HTML5 apps) to native iOS and Android apps with or without Apperian security. You aren't limited by AnyPresence's deployment choices, however. The generated code can always be downloaded and deployed elsewhere, assuming you have compatible deployment environments.



**AnyPresence's app build selection screen. Note the wide assortment of SDKs and the small assortment of prototype app UIs that can be generated.**

The AnyPresence design environment exists online and runs in most browsers. The design environment has a dashboard; a settings screen; screens to create and monitor environments, deployments, and builds; screens to generate and deploy apps, back ends, and SDKs; screens to add and manage data sources and data objects; screens for authorization, roles, and authentication strategy; screens for stock and custom extensions; the interface designer; and a customizable set of themes.

I found the selection of data sources to be good and the implementation of the provided MongoDB data store to be on par with other MBaaS systems. What sets AnyPresence apart is the way the data model integrates throughout the design environment and into all the generated code.

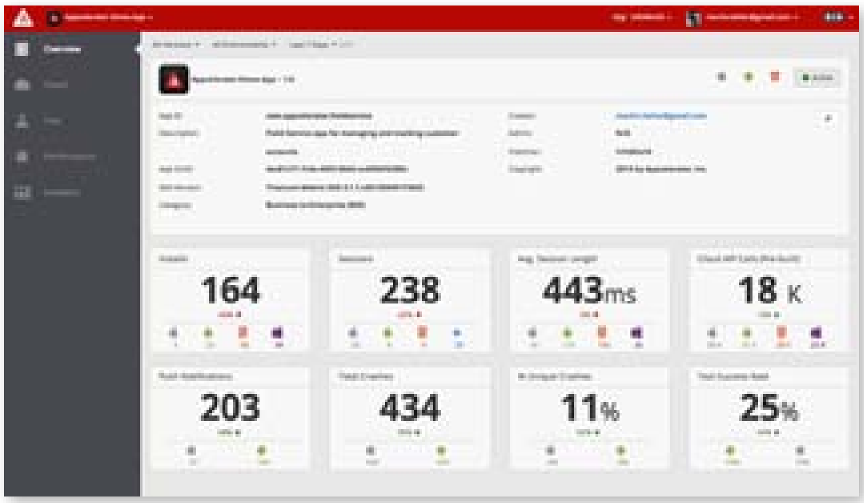
The place you add most monitoring integrations, such as Airbrake and New Relic, is hidden deep in the Deployments/Add-ons tab. Naturally, monitoring is dependent on the runtime environment, and AnyPresence is designed to be environment-agnostic. For Splunk integration, you have to enable syslog output on the back end to push all the logs/events into Splunk systems for reporting and monitoring.

## Appcelerator

Appcelerator Titanium has been a player in the mobile development space for several years, with a local development environment that compiles JavaScript to native code for iOS, Android, and other targets. With the release of Appcelerator Studio 3.3 and Appcelerator Platform 2.0 in July 2014, the company added an MBaaS with about 25 APIs, Node.js support, and online analytics. In addition, Appcelerator has published interfaces to its MBaaS that developers can add to apps built with native SDKs, although it hasn't yet supported native SDKs in its own Appcelerator Studio IDE.

Developers can see a quick overview of app installs, sessions, API calls, and crashes in the online Appcelerator dashboard overview page. Other parts of the dashboard allow for cloud management, testing, performance metrics, and analytics.

The Cloud panel shows usage, exposes data management, displays API request and push notification logs, lists custom services, and allows for cloud configuration. The testing panel uses SOASTA's TouchTest as an integrated mobile testing solution. The performance panel allows you to monitor your apps and troubleshoot performance, crashes, and exceptions. It also lets you view crash trends, integrate with bug tracking systems, and configure your monitoring.



**Appcelerator Platform's dashboard overview for the demo Field Service application. The crashes were deliberately coded into the app.**

Developers can define and view Appcelerator analytics online, as well as optionally publish selected analytics to the Appcelerator Insights app for the iPad, typically for use by a manager.

Appcelerator Platform allows you to build custom back-end services using Studio and Appcelerator's Node.ACS MVC (model-view-controller) framework. Node.ACS combines Node.js and Express with interfaces to Appcelerator Cloud Services. Appcelerator also allows you to run plain Node.js applications on its cloud platform.

Appcelerator has multiple frameworks on the client side, and multiple API types for the cloud. At the base level on the client, Appcelerator offers the Titanium SDK, which provides an interface between JavaScript and native services. At a higher level, Appcelerator offers the Alloy Framework, which is based on the model-view-controller architecture and contains built-in support for Backbone and Underscore. When you create a new client app from Studio, you typically generate one that uses Alloy.

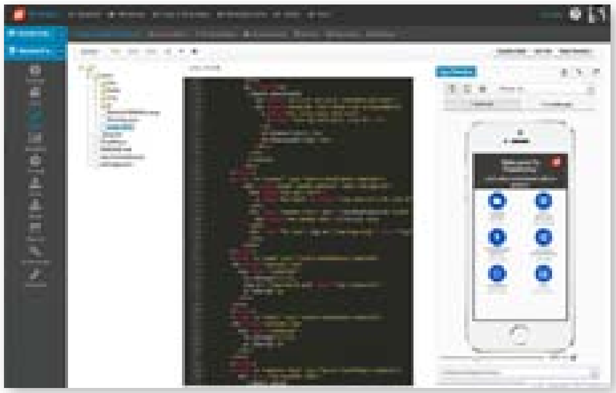
The Alloy framework handles some of what you need for offline/online data synchronization, but not all of it. Appcelerator lacks preconfigured, vetted enterprise data connectors other than for SAP and Salesforce.com. However, because it can run Node modules on its Node.ACS service, developers can draw on modules from the Node.js community. Appcelerator's only commercial sync server is currently limited to a Microsoft Dynamics connector.

## FeedHenry

FeedHenry, with a focus on supporting enterprise line-of-business apps, is a Node.js-based, enterprise-oriented MBaaS and mobile application platform. It has a wide array of integrations, both online and offline development options, collaborative app building, and a drag-and-drop form builder. FeedHenry was spun off from the Irish Research Institute in 2010 and acquired by Red Hat in September 2014.

FeedHenry claims to have global infrastructure on all major clouds and support for on-premise, back-end deployment. The FeedHenry online environment integrates directly with GitHub for collaboration and version control.

FeedHenry 3 supports native SDKs for iOS, Android, and Windows Phone 8, along with hybrid apps using Apache Cordova, HTML5 mobile Web apps, and Sencha, Xamarin, and Appcelerator Titanium. The way the JavaScript interface to the FeedHenry cloud works, it



**FeedHenry includes an online editor, supporting offline tools, and a command-line interface. Here we see the mobile app, with a code editor in the middle of the screen and a preview at right. You can configure the back-end service in another pane of the online interface.**

would be hard to find a JavaScript framework that isn't compatible.

When writing for FeedHenry in JavaScript, you include the feedhenry.js script in your HTML, initialize it with `$fh.init`, then call cloud functions from the `$fh` namespace. FeedHenry can import existing apps from a Zip file or Git repository.

The FeedHenry build service, which functions along the same lines as Adobe PhoneGap Build, can turn an HTML5 app into binaries for Android, BlackBerry, iPhone, iPad, iOS (universal), and Windows Phone. Each binary can connect to one of your MBaaS instances, and it can be built for development, distribution, release, or debugging, depending on the platform.

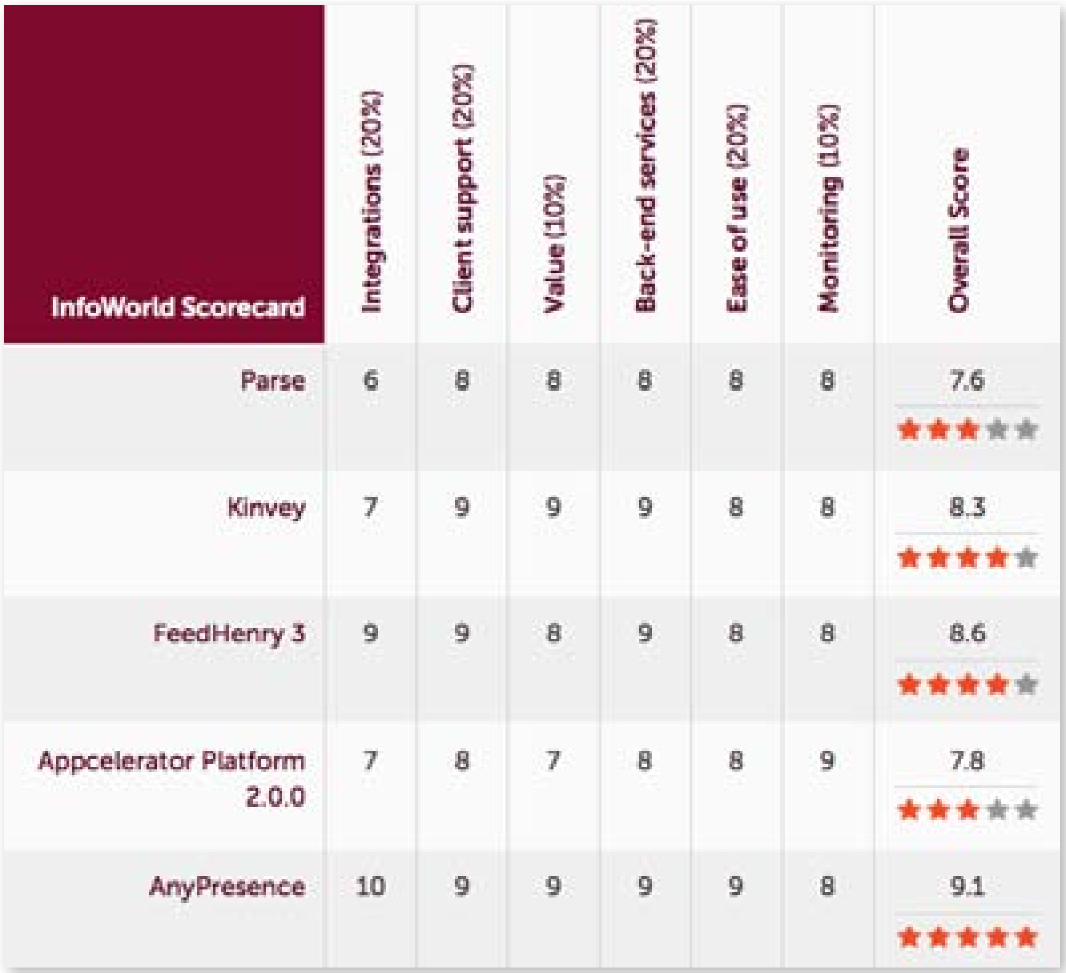
FeedHenry has a drag-and-drop form builder with a good assortment of templates to use as starting points. However, at the time I reviewed FeedHenry, it had few full-fledged app templates.

FeedHenry lists more than 50 Node.js plug-ins in its curated modules list. That list includes interfaces to most major relational and NoSQL databases. Should the curated list not include what you seek, the much larger list of Node community modules is likely to yield a match.

FeedHenry runs on all major public and private clouds, and on a wide range of IaaS and PaaS infrastructures. FeedHenry operates a HIPAA-compliant cloud and live clusters in both Europe and North America.

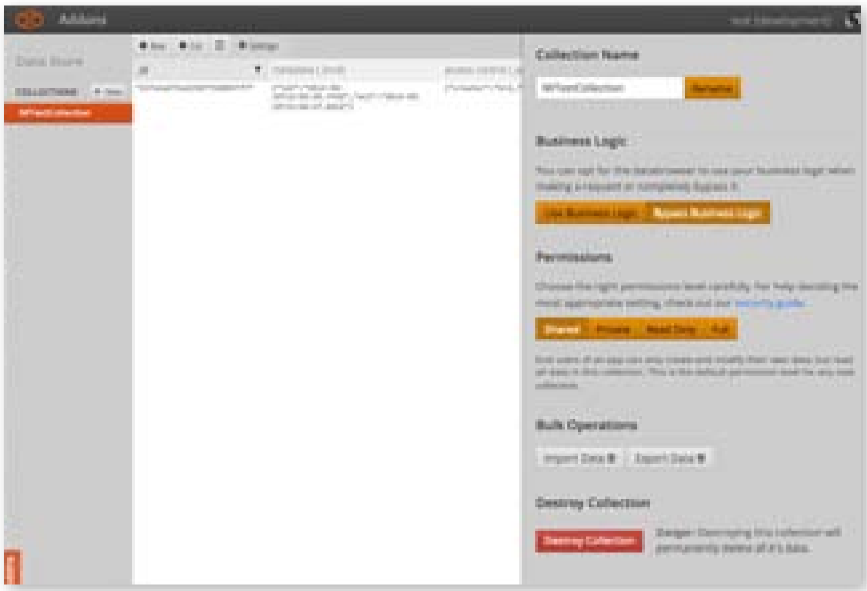
## Kinvey

Kinvey bills itself as a complete mobile and Web app platform. It has extensive client support, integrates with the major enterprise databases, and offers a back-end data store, a file store, push notifications, mobile analytics, iBeacon support, and the ability to run custom code on the back end.



| InfoWorld Scorecard | Integrations (20%) | Client support (20%) | Value (10%) | Back-end services (20%) | Ease of use (20%) | Monitoring (10%) | Overall Score |
|---|---|---|---|---|---|---|---|
| Parse | 6 | 8 | 8 | 8 | 8 | 8 | 7.6 ★★★☆☆ |
| Kinvey | 7 | 9 | 9 | 9 | 8 | 8 | 8.3 ★★★★☆ |
| FeedHenry 3 | 9 | 9 | 8 | 9 | 8 | 8 | 8.6 ★★★★☆ |
| Appcelerator Platform 2.0.0 | 7 | 8 | 7 | 8 | 8 | 9 | 7.8 ★★★☆☆ |
| AnyPresence | 10 | 9 | 9 | 9 | 9 | 8 | 9.1 ★★★★★ |

Kinvey sells to IT as its primary customer because it provides an enterprise platform, not for one or two apps but for tens or hundreds of apps for an enterprise. However, it also engages and supports the developer community app by app.



**Kinvey collections use MongoDB, which provides a schema-less, no-SQL database for use by your apps. This screen lets you create and design collections (only the creation step is necessary) and choose whether to enable or bypass your database business logic.**

Kinvey supports native, hybrid, and HTML5 apps. It has native toolkit support for iOS and Android. In addition, it supports Angular, Backbone, Node.js, Apache Cordova/PhoneGap, and Appcelerator Titanium, and it provides a REST API. Kinvey integrates with apps through libraries and API calls, and expects you to edit your app locally.

Kinvey cloud code is written in JavaScript, although not Node.js, and edited online. In addition to using standard JavaScript and external services, it can use Kinvey APIs for logging, accessing collections, sending push notifications, sending email, validating requests, date and time functions, asynchronous processing, rendering a Mustache template, and obtaining the back-end context. Cloud code can live in hook processing functions and custom endpoints. Cloud code is versioned internally in Kinvey.

Kinvey supports deploying on almost any cloud, including private clouds. That includes deploying to HIPAA-compliant facilities and to facilities located entirely in the EU. Even Kinvey's multitenant cloud is considered secure enough for most apps, as the company does end-to-end encryption, and customers that use data links can keep their data in databases behind their own firewalls. If you have a Google App Engine server, you can link it to your Kinvey back end.

Authentication can be done internally by Kinvey or through LDAP or Active Directory in the business and enterprise versions. Kinvey also supports Facebook, Twitter, Google+, and LinkedIn identities through OAuth.

Kinvey data links connect to Kinvey's MongoDB data store. In most cases, customers forward the CRUD requests directly to the real back end, but some cache the data in MongoDB. Kinvey currently has data links for Microsoft Dynamics CRM, Salesforce CRM, Oracle Database, and Microsoft SQL Server.

Kinvey has an automated control setup for offline data synchronization, in which data is automatically pulled from the cache if the application is offline. If the application is online, data is pulled from the network and stored in the cache. Using automated control, your Kinvey app will attempt to synchronize any locally stored data when the device goes online again, but if the server data has also changed you'll have a conflict. You can set your conflict resolution policy to `clientAlwaysWins`, `serverAlwaysWins`, or a custom conflict resolution function.

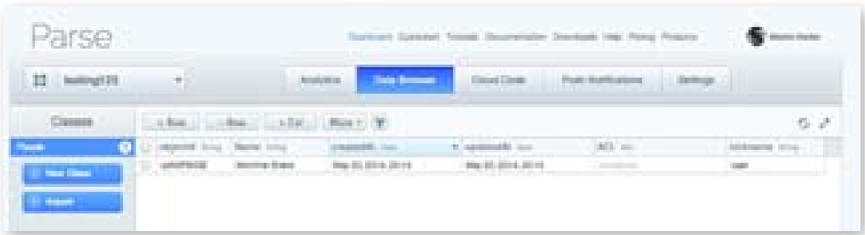
## Parse

Parse was once the poster child for MBaaS, and despite its acquisition by Facebook, it is still a viable, low-friction mobile back end for limited-volume consumer apps. On the plus side, it is well-documented, with good native client support and a JavaScript client SDK based on Backbone. Parse also runs JavaScript code on the back end, which offers developers the option of an all-JavaScript application stack. On the minus side, Parse is missing big pieces necessary for business apps, such as data integration, offline operation, and online/offline synchronization. At the same time, its pricing seems geared to lower-volume apps.

Parse supports native mobile, JavaScript, and desktop apps. On the mobile side, it has native support for iOS, Android, and Windows Phone 8. On the desktop, it has support for OS X and Windows 8 (.Net), as well as Unity games.

Parse lets you run JavaScript code in the cloud using the same Parse JavaScript SDK as the client. Rather than have you routinely edit your cloud code in a browser, as FeedHenry and Kinvey do, Parse supplies a command-line tool for managing code in Parse Cloud and allows you to use your favorite JavaScript editor on your computer. However, you can view your code and your logs in your dashboard. The command-line tool is an app scaffold generator, app deployment tool, log printer, app rollback tool, and self-updater.

Parse can send Push notifications to iOS, Android, Windows 8, and Windows Phone 8. In each case, you'll have to provision your push server, then provide the certificate or credentials to your app.



The Parse Cloud data browser lets you import bulk data; add classes, columns, and rows; and view filtered data.

Parse has a fairly complete user system predefined, including the usual sign-up mechanism with email verification and a provision for anonymous users. A system of ACLs controls what data individual users can read and write. For more complicated use cases, Parse supports a hierarchy of roles, with a separate layer of ACLs for the roles.

Parse has nine integrations with other services. Three of them -- Mailgun, Mandrill, and SendGrid -- are for sending email. Stripe is for charging credit cards. Twilio sends SMS and voice messages. Third-party modules are available to integrate Parse with Cloudinary, Instagram, and Paymill.

As far as I can tell, implementing enterprise data integration with Parse requires writing a REST Web service wrapper for the data source and a JavaScript module for Parse. I haven't seen any options for hosting Parse other than using its own multitenant cloud.

## Mo' better MBaaS

As you can see from the scores listed at the bottom of the first page of this article, AnyPresence earned the highest marks: a combined score of 9.1 and an Editor's Choice badge. I feel that AnyPresence offers more value than the others for enterprises that need to integrate their existing systems with mobile applications, as it generates customized SDKs, along with apps and back ends, from your model and design. Costing a "low six figures" per year, however, it won't fit into every company's budget.

FeedHenry, which earned an overall score of 8.6, is also an enterprise-oriented MBaaS. FeedHenry has a nice integration with Git for collaboration and version control, and I like its hosted app build service, its Node.js back end and curated Node modules list, and its drag-and-drop form designer. Like AnyPresence, FeedHenry may not fit into every company's budget.

Kinvey, with an overall product score of 8.3, engages as a company with the developer community, as well as with corporate IT departments. I like the way Kinvey does enterprise data links through its internal NoSQL database API, and I appreciate the way it has structured its hooks for back-end business logic.

I criticized Appcelerator for its apparent lack of effort to curate data integration modules, and considered that its high price relative to FeedHenry and Kinvey may diminish its overall value, giving it a net score of 7.8. However, Appcelerator as a company only recently pivoted into the MBaaS space. It may yet fill in its product's missing functionality and adjust its pricing to be more competitive.

Finally, I consider Parse suitable for building and operating back ends for consumer-facing mobile apps, and not business apps, given its lack of any data connectors other than a basic REST client. My other major reservation about Parse is its usage-based pricing, which lets a developer get started easily but could potentially bite an underfunded startup that suddenly had a viral hit on its hands without a real business model. Its score is 7.6, the lowest in this group.

That isn't to say you shouldn't use Parse. It's a viable, low-friction way to get started with back end as a service. However, if you choose to use it, go in with your eyes open, monitor your costs, and be prepared to throttle or eliminate service calls that are running up bills you can't afford.

For business apps, AnyPresence and FeedHenry lead the pack in both ease and capabilities. Kinvey is not far behind, and its pricing is more favorable for smaller businesses. ■

Deep Dive

# AnyPresence aces enterprise mobile apps

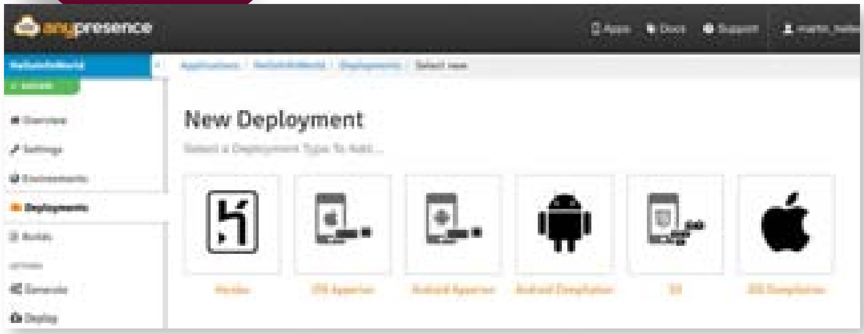
*AnyPresence combines broad client support, useful code generation, and a rich set of options for data storage and enterprise integration*

AnyPresence has an online designer that not only generates back-end and mobile app code, but also customized mobile API code.

**AnyPresence is designed** "to meet the evolving needs of mobile-enabling a developer ecosystem with fully portable run-time source code and zero platform lock-in." OK, that's a mouthful of marketing-speak. What it means, however, is that the goal of AnyPresence is not only to help enterprises build back ends for their mobile apps; AnyPresence combines app building, back-end services, and an API gateway.

AnyPresence has an online designer that not only generates back-end and mobile app code, but also customized mobile API code. All the generated code can be downloaded, edited, and run on compatible platforms. To go with one of AnyPresence's favorite customer examples, MasterCard has used AnyPresence to enable partners to easily build mobile apps against MasterCard's Open API services.

Now, it isn't necessary for you to have partners developing against your APIs for AnyPresence to make sense for your business. If you think about the generated app code as "app UI starter kits" (the term AnyPresence likes) or test apps for your back end, then the generated mobile API classes would be building blocks for your developers' "real" apps.

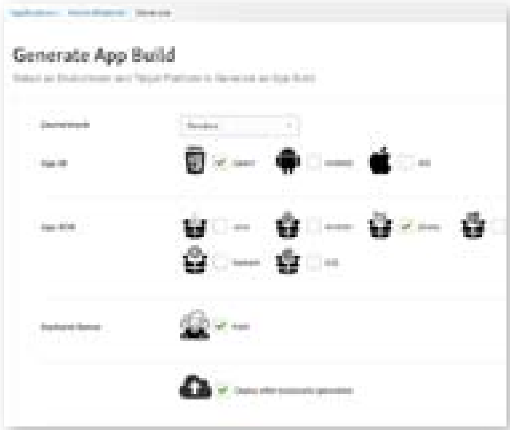


**Deployment options in AnyPresence include Heroku (usually for Rails apps), Amazon S3 (usually for HTML5 apps), and native iOS and Android apps with or without Apperian security.**

## Client support

AnyPresence generates App UIs (or starter kits, if you wish) for jQuery, Android (XML layout), and iOS (storyboard), and it generates App SDKs for Java, Android, HTML5, Windows Phone, Xamarin, and iOS. The design environment refers to the generated JavaScript/HTML5 SDK as "jQuery." In fact, what AnyPresence actually generates is CoffeeScript that uses the Underscore, Backbone, and jQuery libraries. For example:

```
class AP.auth.Authentication
   _.extend @, Backbone.Events
# if server ever responds with
401, assume the session expired
$.ajaxSetup
   complete: _.debounce ((xhr,
result) => @destroySession() if
xhr.status == 401 and result ==
'error'), 150
```



**AnyPresence's app build selection screen. Note the wide assortment of SDKs that can be generated and the small assortment of prototype app UIs that can be generated.**

AnyPresence generates back-end servers for Ruby on Rails, and in the future will also generate Node.js back ends, which will be a good development. If the developer folklore and publicly reported cases (such as LinkedIn) are to be believed, Node tends to scale better than Rails for many applications.

The AnyPresence environment can generate deployments to Heroku (usually for a Rails back end) to Amazon S3 (usually for HTML5 apps) to native iOS and Android apps with or without Apperian security. You aren't limited by AnyPresence's deployment choices, however. The generated code can always be downloaded and deployed elsewhere, assuming you have compatible deployment environments.

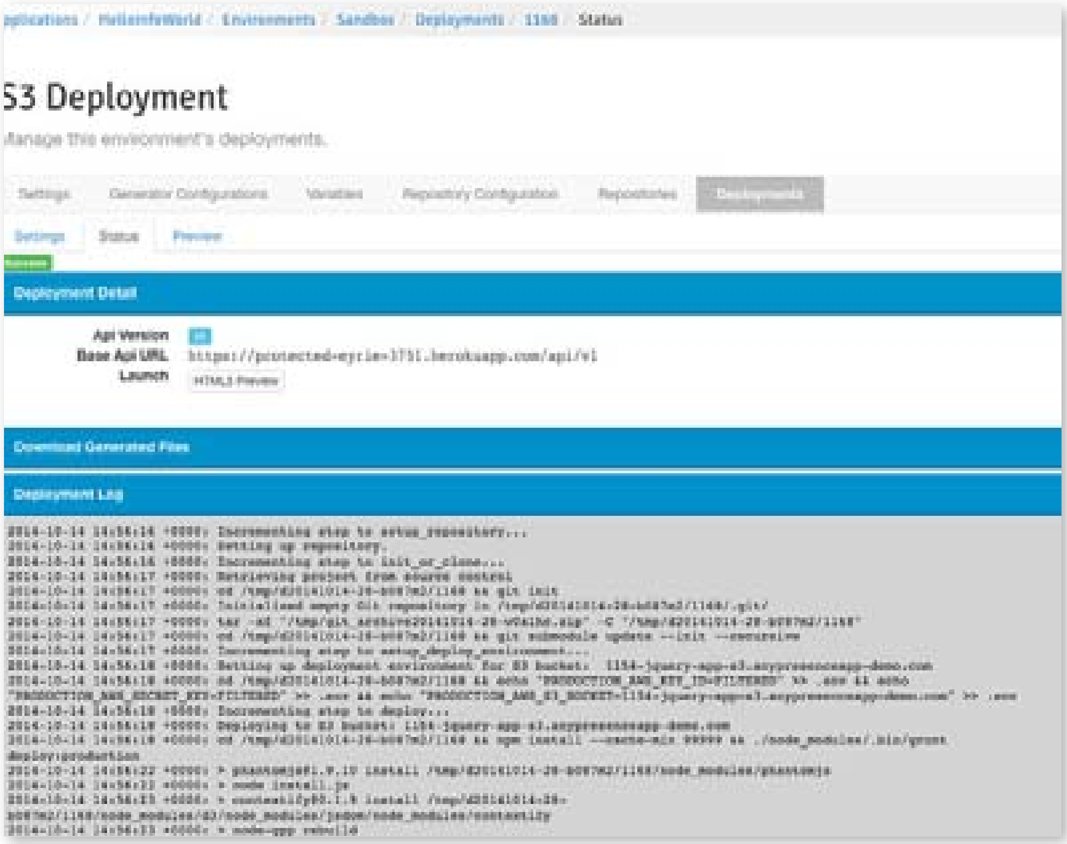
Deployment options in AnyPresence include Heroku (usually for Rails apps), Amazon S3 (usually for HTML5 apps), and native iOS and Android apps with or without Apperian security.

## Design and deployment environment

The AnyPresence design environment exists online and runs in most browsers. According to the company, the environment supports older browsers that are still in use at enterprise customers, but I didn't break out Windows XP to test IE6. I tested in the most current version of Chrome running on OS X. For the one day when Chrome "broke" the site (and many similar sites), I used Safari 7.1.

The design environment has a dashboard; a settings screen; screens to create and monitor environments, deployments, and builds; screens to generate and deploy apps, back ends, and SDKs; screens to add and manage data sources and data objects; screens for authorization, roles, and authentication strategy; screens for stock and custom extensions; the interface designer; and a customizable set of themes.

The data sources screen is where you add integrations with enterprise applications (such as Salesforce.com) and databases (such as Oracle). I found the selection of data sources to be good, and the implementation of the provided MongoDB data store to be on par with other MBaaS systems. What sets AnyPresence apart is the way the data model integrates throughout



S3 Deployment

Manage this environment's deployments.

Api Version: v1
Base Api URL: https://protected-eyrie-3751.herokuapp.com/api/v1

```
2014-10-14 14:56:16 +0000: Incrementing step to setup_repository...
2014-10-14 14:56:16 +0000: Setting up repository.
2014-10-14 14:56:16 +0000: Incrementing step to init_or_clone...
2014-10-14 14:56:17 +0000: Retrieving project from source control
2014-10-14 14:56:17 +0000: cd /tmp/d20141014-28-b087m2/1168 && git init
2014-10-14 14:56:17 +0000: Initialized empty Git repository in /tmp/d20141014-28-b087m2/1168/.git/
2014-10-14 14:56:17 +0000: tar -xf "/tmp/git_archive20141014-28-w5a1bd.zip" -C "/tmp/d20141014-28-b087m2/1168"
2014-10-14 14:56:17 +0000: cd /tmp/d20141014-28-b087m2/1168 && git submodule update --init --recursive
2014-10-14 14:56:17 +0000: Incrementing step to setup_deploy_environment...
2014-10-14 14:56:18 +0000: Setting up deployment environment for S3 bucket:  1154-jquery-app-s3.anypresenceapp-demo.com
2014-10-14 14:56:18 +0000: cd /tmp/d20141014-28-b087m2/1168 && echo "PRODUCTION_AWS_KEY_ID=FILTERED" >> .env && echo
"PRODUCTION_AWS_SECRET_KEY=FILTERED" >> .env && echo "PRODUCTION_AWS_S3_BUCKET=1154-jquery-app-s3.anypresenceapp-demo.com" >> .env
2014-10-14 14:56:18 +0000: Incrementing step to deploy...
2014-10-14 14:56:18 +0000: Deploying to S3 bucket: 1154-jquery-app-s3.anypresenceapp-demo.com
2014-10-14 14:56:18 +0000: cd /tmp/d20141014-28-b087m2/1168 && npm install --cache-min 99999 && ./node_modules/.bin/grunt
deploy:production
2014-10-14 14:56:22 +0000: > phantomjs@1.9.10 install /tmp/d20141014-28-b087m2/1168/node_modules/phantomjs
2014-10-14 14:56:22 +0000: > node install.js
2014-10-14 14:56:23 +0000: > contextify@0.1.9 install /tmp/d20141014-28-
b087m2/1168/node_modules/d3/node_modules/jsdom/node_modules/contextify
2014-10-14 14:56:23 +0000: > node-gyp rebuild
```

**Amazon S3 deployment detail for an HTML5 app. From here we can view a preview and download the generated files.**

the design environment and into all the generated code.

The place you add most monitoring integrations, such as Airbrake and New Relic, is hidden deep in the Deployments/Addons tab, since it's dependent on the runtime environment, and AnyPresence is designed to be environment-agnostic. For Splunk integration, you have to enable syslog output on the back end to push all the logs/events into Splunk systems for reporting and monitoring.

## Generated code

I downloaded and examined the generated code for an AnyPresence "hello, world" application that I had built in the design environment -- specifically a Rails back end, HTML5 mobile app starter kit, and JavaScript SDK. I didn't look at all the other possible generated code, as the number of combinations was daunting.

In the mobile app, I found most of the interesting CoffeeScript code under app/javascripts/app, and it was divided into application, model, view, and controller directories. I didn't have any trouble reading the code once I found it, although the excessive spacing gave away the fact that it was generated. There seems to be a mechanism for adding your own code to "custom" subdirectories to allow it to be automatically incorporated into the build, and to avoid having it overwritten by future code generation from the design environment.

The generated code used nine JavaScript libraries: Backbone, jQuery, jQuery Mobile, large-local-storage, NVD3, Offline.js, Q, the generated SDK (holding a generated AP class), and Underscore (needed for Backbone). Comments in the code indicated where to switch included file sources around for local development. A fairly full set of grunt commands is provided for building the app, along with basic test scripts.

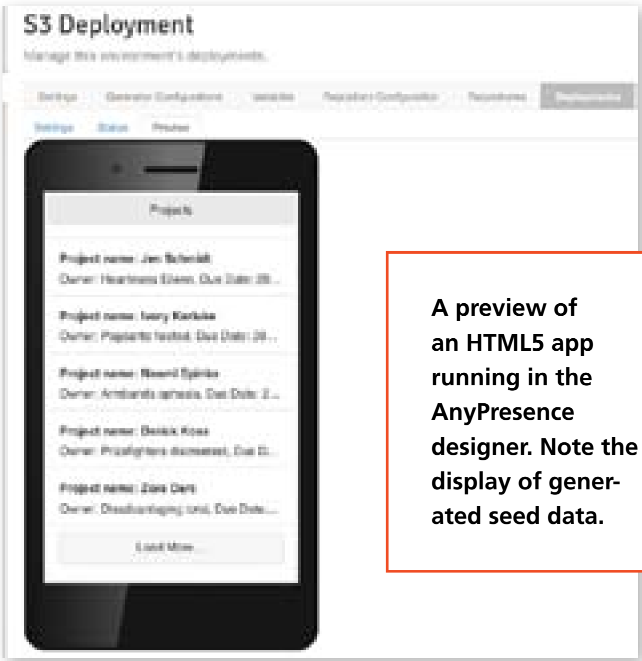
In the SDK, the Backbone, jQuery, large-local-storage, Q, and Underscore libraries were in use, and the SDK class library was generated. Logic in the SDK implements collections, relationships, authentication, and an MVC (model view controller) model, and generated HTML documentation is provided. Again, a fairly full set of grunt commands is provided for building the SDK, along with basic test scripts to exercise, for example, the CRUD (create, read, update, delete) functionality.

In the Rails API, I found the MVC model divided the way it's supposed to be, with the field definitions in the model, for example. I found a good set of tests, especially for the CRUD, and the seed data declared in the Ruby code but excluded from the test environment -- again, as it's supposed to be structured.

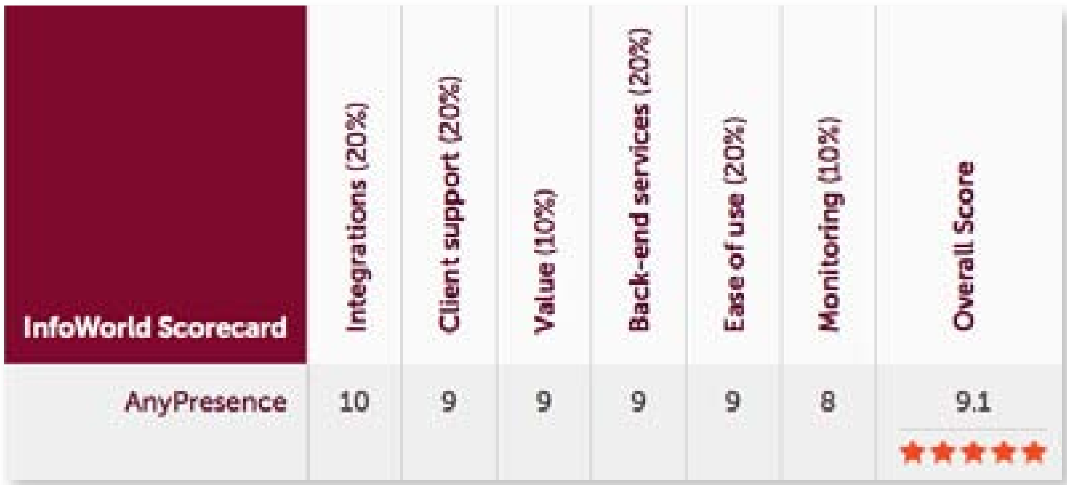
In general, I found the generated code I examined to be in line with best practices and decently commented, although not always formatted as I would have liked. While I was initially concerned that the generated code would not be maintainable without the design environment, my fears were laid to rest when I reviewed the source code.

I initially felt that the absence of a built-in monitoring service might be a competitive weakness, but I can't argue with AnyPresence's integration with the best-of-breed monitoring systems, many of which are already in use by its customers. Also, I didn't delve deeply into the offline app capabilities of AnyPresence for different platforms. Given its generation of customized SDKs for multiple clients, generating offline/online database synchronization for each client seems like a lot to ask.

AnyPresence, while priced a bit higher than the entry level of the other MBaaS systems I reviewed for this series, offers more value than these competitors for enterprises that need to integrate their existing systems with mobile applications. It is especially valuable for enterprises that wish to expose their APIs to partners who can in turn use them in their own mobile applications. ■



**A preview of an HTML5 app running in the AnyPresence designer. Note the display of generated seed data.**



| InfoWorld Scorecard | Integrations (20%) | Client support (20%) | Value (10%) | Back-end services (20%) | Ease of use (20%) | Monitoring (10%) | Overall Score |
|---|---|---|---|---|---|---|---|
| AnyPresence | 10 | 9 | 9 | 9 | 9 | 8 | 9.1 ★★★★★ |

Deep Dive

# Appcelerator is a mobile cloud platform in progress

*Appcelerator Platform 2 combines rich mobile client support with the advantages of Node.js, but lacks pre-built integrations and full sync support*

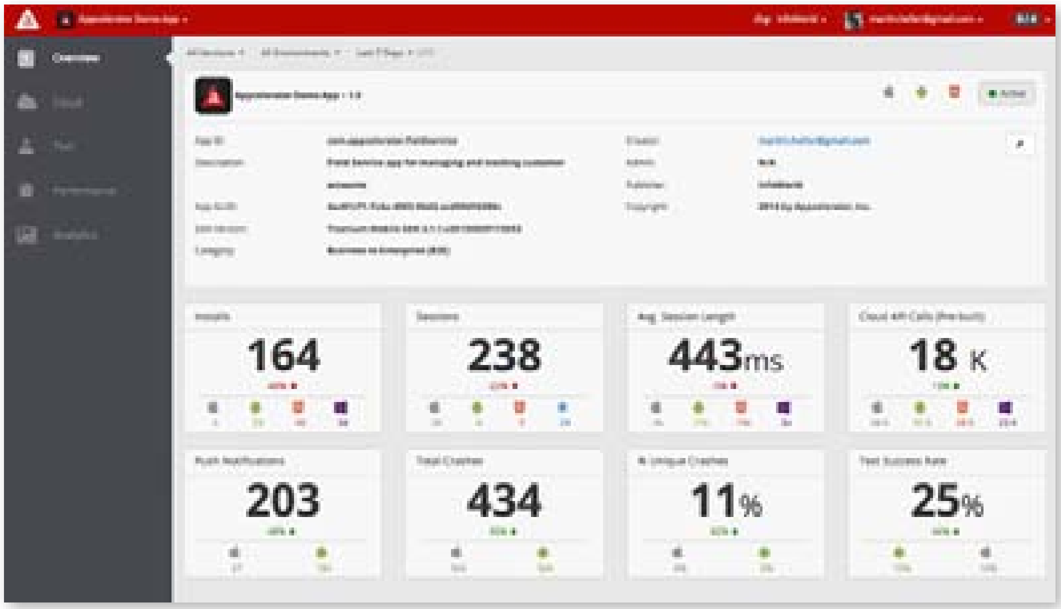
**Appcelerator Titanium has been** a player in the mobile development space for several years, with a JavaScript-based development environment that compiles to native code for iOS, Android, and other targets. With the release of Appcelerator Studio 3.3 and Appcelerator Platform 2.0 in July, the company added a mobile back end as a service (MBaaS) accompanied by about 25 APIs, Node.js support, and online analytics. Also, Appcelerator has published interfaces to its MBaaS that developers can add to apps built with native SDKs, though it hasn't yet supported native SDKs in its own Studio IDE.

"Studio" is short for any edition of Appcelerator Studio, the company's enterprise-oriented IDE; Titanium Studio, the company's free IDE; and Aptana Studio, the free Eclipse plug-in on which the other two products are based. Aptana was one of my favorite open source JavaScript and Rails IDEs in its day, before it was bought by Appcelerator, though my affections have since moved on to more modern IDEs and editors.

## Services and APIs

The principal focus of this release of Appcelerator is its MBaaS and analytics. There are more than 25 prebuilt APIs, including most of the usual suspects such as push notifications and pair storage, and extending to users, places, photos, and social integrations. These cloud service APIs are in addition to Appcelerator's interfaces to native device capabilities such as local storage, media services, geolocation, contacts, and accelerometer support. Furthermore, Appcelerator now includes a layer on top of Node.js, called Node.ACS, for building custom cloud services. (ACS refers to Appcelerator Cloud Services.)

Early in Appcelerator's development, some of its Android APIs were coerced into looking like



Appcelerator Platform's dashboard overview for the demo Field Service application. The crashes were deliberately coded into the app.

their iOS equivalents. Not surprisingly, that didn't always work well. I'm not sure when these APIs changed, but now the Appcelerator wrappers for Android APIs seem to follow the Android model. That's good because they actually work reliably; it's bad because you have to include different modules for Android builds than for iOS builds. Appcelerator still supports BlackBerry, but I have lost interest in BlackBerry development over the last couple of years, as its market share has essentially disappeared.

## Dashboard and analytics

Developers can see a quick overview of app installs, sessions, API calls, and crashes in the online Appcelerator dashboard overview page. Other parts of the dashboard allow for cloud management, testing, performance metrics, and analytics.

The Cloud panel shows usage, exposes data management, displays API request and push notification logs, lists custom services, and allows for cloud configuration. The testing panel uses SOASTA's TouchTest as an integrated mobile testing solution. The performance panel allows you to monitor your apps; troubleshoot performance, crashes, and exceptions; view crash trends; integrate with bug tracking systems; and configure your monitoring.

Developers can define and view Appcelerator analytics online and optionally publish selected analytics to the Appcelerator Insights app for

| InfoWorld Scorecard | Integrations (20%) | Client support (20%) | Value (10%) | Back-end services (20%) | Ease of use (20%) | Monitoring (10%) | Overall Score |
|---|---|---|---|---|---|---|---|
| Appcelerator Platform 2.0.0 | 7 | 8 | 7 | 8 | 8 | 9 | 7.8 ★★★☆☆ |

the iPad, typically for use by a manager. There are five categories of analytics: real time, users, sessions, events, and event funnels.

Appcelerator Insights allows you (or your manager) to view published analytics on an iPad. The most recent improvement in Insights is the ability to view event funnels. If properly constructed, event funnels can tell you how users are progressing through your app, whether they are getting to the key functionality, and if not, where they are bogging down.

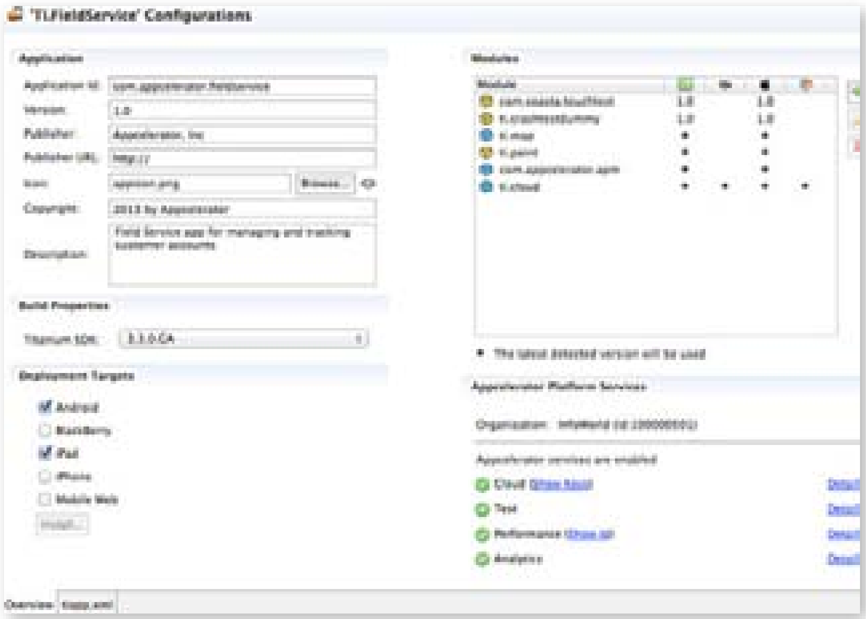
Appcelerator Platform allows you to build custom back-end services using Studio and Node.ACS. Node.ACS combines Node.js and Express with interfaces to Appcelerator Cloud Services and a model-view-controller (MVC) framework. Appcelerator also allows you to run plain Node.js applications on its cloud platform.

The Node.ACS support in Studio requires you to have Node.js installed on your development machine, and it tries to install acs with Node. On my iMac, this failed. I eventually had to run this rather scary command in Terminal:

```
sudo npm --unsafe-perm -g install
acs
```

Some developers at Appcelerator found that for me, and it worked -- but using the unsafe permissions flag to Node's package manager didn't exactly make me feel warm and fuzzy. Appcelerator claims this is a known but uncommon Node.js bug; Node.js claims it's a bad install script. Finger-pointing doesn't make me feel warm and fuzzy, either, especially finger-pointing at a free, open source software project to which you're adding commercial software.

Appcelerator Studio's app configuration screen.



'Ti.FieldService' Configurations

Application
Application Id: com.appcelerator.fieldservice
Version: 1.0
Publisher: Appcelerator, Inc
Publisher URL: http://
Icon: appicon.png
Copyright: 2013 by Appcelerator
Description: Field Service app for managing and tracking customer accounts

Build Properties
Titanium SDK: 3.1.0.GA

Deployment Targets
[✓] Android
[ ] BlackBerry
[✓] iPad
[ ] iPhone
[ ] Mobile Web

Modules
[illegible]

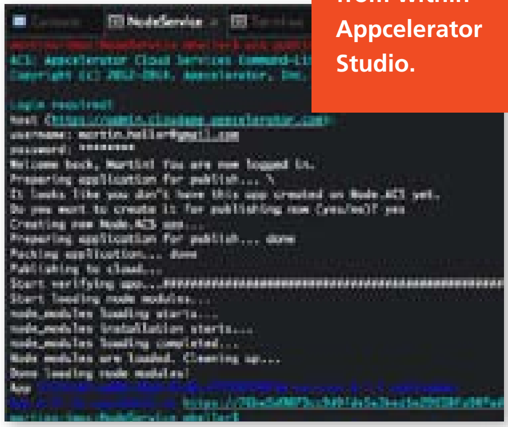
* The latest detected version will be used

Appcelerator Platform Services
Organization: InfoWorld [illegible]
Appcelerator services are enabled
Cloud
Test
Performance
Analytics

## Frameworks and API types

Appcelerator has multiple frameworks on the client side and multiple API types for the cloud. At the base level on the client,

Using the acs command to publish an app from within Appcelerator Studio.



```
[illegible]
ACS: Appcelerator Cloud Services Command-Li[illegible]
Copyright (c) 2012-2013, Appcelerator, Inc.

Login required!
host: (https://admin.cloudapp.appcelerator.com)
username: martin.heller@gmail.com
password: ********
Welcome back, Martin! You are now logged in.
Preparing application for publish... \
It looks like you don't have this app created on Node.ACS yet.
Do you want to create it for publishing now (yes/no)? yes
Creating new Node.ACS app...
Preparing application for publish... done
Packing application... done
Publishing to cloud...
Start verifying app...##########################################
Start loading node modules...
node_modules loading starts...
node_modules installation starts...
node_modules loading completed...
Node modules are loaded. Cleaning up...
Done loading node modules!
App [illegible]
[illegible]
```

Appcelerator offers the Titanium SDK, which provides an interface between JavaScript and native services. At a higher level, Appcelerator offers the Alloy Framework, which is based on the MVC architecture and contains built-in support for Backbone.js and Underscore.js. When you create a new client app from Studio, you'll typically generate one that uses Alloy.

On the cloud side, you can reach the Appcelerator Cloud Services using a REST API, via bindings to the Titanium SDK, via Node.ACS as discussed above, and via native SDK support as discussed below. The REST API will always work, though it's the least convenient option. You'll mostly want to use REST calls to reach new services that don't yet have bindings to the Titanium SDK.

## Native SDK support

Appcelerator used to make a big deal about Titanium's ability to trans-compile JavaScript into native code for iOS and Android, and pooh-poohed any need for Native SDK support. As part of Appcelerator's transition into having an MBaaS, it now includes Native SDK support for iOS and Android.

On iOS, you'll have to add the Appcelerator framework code to your Objective-C project, import Appcelerator.h from your code, and call various Objective-C classes starting with APS. On Android, you'll copy the Appcelerator framework JAR file to your Java project, import the APSServiceManager class, and call various Java classes starting with APS. In both systems, working from the provided examples and instructions to add MBaaS functionality in the native SDK tools is straightforward. However, none of this is included in Appcelerator Studio. Rather, you use Eclipse or Xcode for this editing. In the future, as I understand it, Studio or some other Appcelerator component might support actions to add the appropriate stuff to Java and Objective-C projects.

## Data import and offline/online synchronization

Appcelerator can call REST and even SOAP services using HTTPClient and its built-in parsing routines. If you've set up a REST wrapper for a database query, you can get the JSON data into your app fairly easily. That wrapper might be implemented on Node.js or on another server, as in the case of a Web service extension to the database server.

According to Appcelerator, from the MBaaS side, the most common scenario is writing REST services on Node.ACS that talk to the customer's data sources. Depending on the scenario, Appcelerator sometimes discourages customers from accessing their databases directly from a mobile middle tier. (Usually those types of customers immediately agree and recognize their security team would shut down the idea.)

That begs the question of enterprise integrations, however. A more serious MBaaS would already have tested, integrated modules set up to easily map the major databases to a form consumable by its apps, certainly for Oracle, SQL Server, MySQL, and Postgres. I view leaving this as an exercise for the developer as a cop-out, though writing RESTful database wrappers isn't rocket science, especially on Node.js.

Appcelerator says it has a few enterprise connectors it sells on the MBaaS layer, such as for SAP and Salesforce.com. And one of the advantages of Node is the supply of community-developed modules for many other sources such as MySQL, SQL Server (which works on a Windows server with Node.js), Postgres, and several NoSQL databases.

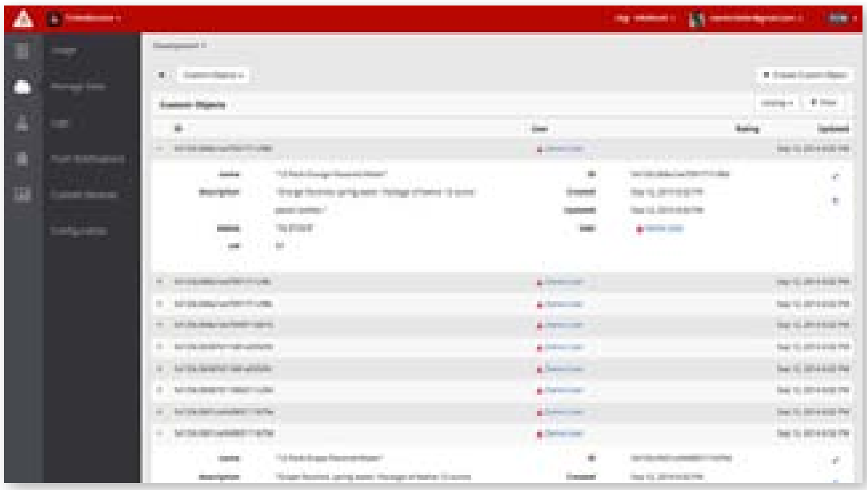
Similarly, Appcelerator can use a local SQLite database on a device, use pair storage, cache in-memory, and detect when the device is online. However, it has no complete framework in place for handling intermittently connected apps, especially not conflict resolution. According to the company, most of its customers use Alloy models to handle some of these duties.

Synchronization is admittedly a difficult problem, but it's one area where most mobile developers need help from the vendor, and it's important in the real world. A more serious MBaaS would at least provide hooks for code that handles data entry conflict resolution, but Appcelerator hasn't done that yet.

Appcelerator notes that it has a commercial sync server available with a Microsoft Dynamics connector attached, and it will expand this to all supported data sources later this year. At least the company is working on the problem.

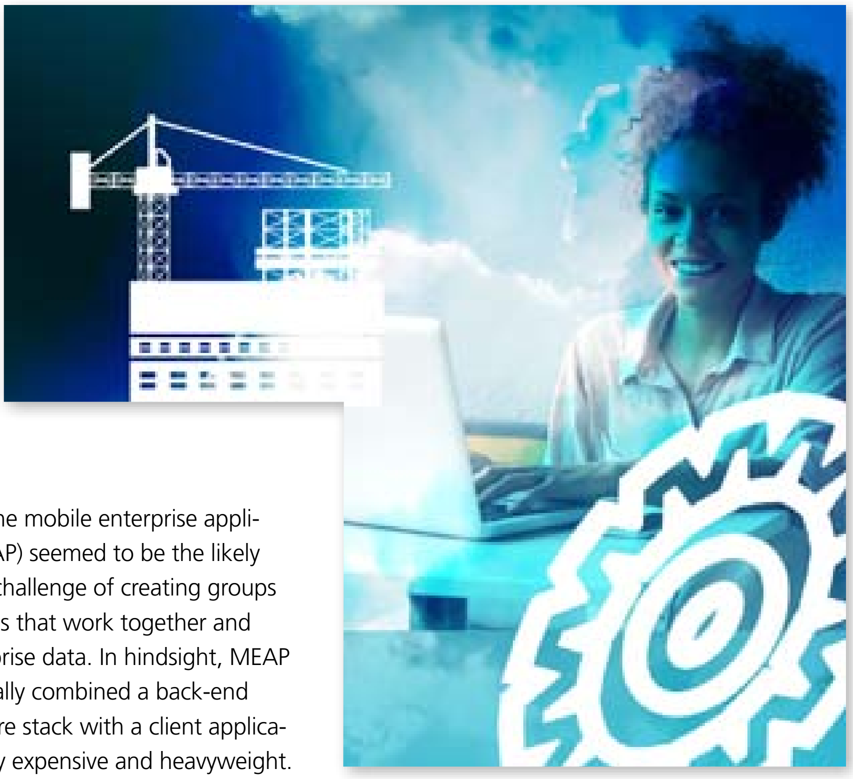
Overall, I see Appcelerator as a mobile app development tool company just beginning to pivot into the MBaaS space. Most of the basic pieces are in place, but Appcelerator hasn't quite finished the hard parts. ■

**Examining the custom objects in the MongoDB database for the Field Service demo app.**



# FeedHenry uses Node.js to fortify mobile apps

*FeedHenry boosts enterprise mobile applications with rich client and tools support, as well as fast, scalable, Node.js-based back-end services*



**A few years ago,** the mobile enterprise application platform (MEAP) seemed to be the likely answer to the huge challenge of creating groups of mobile applications that work together and integrate with enterprise data. In hindsight, MEAP systems, which typically combined a back-end server and middleware stack with a client application, seem excessively expensive and heavyweight.

The current trend is toward MBaaS (mobile back end as a service) platforms, loosely coupled with native, Web, and hybrid mobile applications. An MBaaS -- which might be focused on business applications, consumer applications, or both -- places much of the logic onto the mobile device, while enforcing security and managing the data at the back end. Even traditional MEAP vendors, such as Kony, are now offering MBaaS platforms.

FeedHenry is a Node.js-based, enterprise-oriented MBaaS and mobile application platform with a wide array of integrations, both online and offline development options, collaborative app building, and a drag-and-drop form builder. FeedHenry was spun off from the Irish Research Institute in 2010. The company describes its offering as a cloud platform for building mobile-first solutions, both B2C and B2E, with a focus on enterprise line-of-business apps. FeedHenry claims to have global infrastructure on all major clouds, as well as support for on-premise back-end deployment.

Note that FeedHenry is priced only for enterprise customers. The company does not currently sell to independent developers or small to midsized businesses.

FeedHenry has impressive customer apps in its catalog. For example, Aer Lingus built a multi-platform app for mobile check-in, flight search, real-time status updates, and flight bookings. The original app had a 10-week time to market. The app integrates with eight back-end systems through the FeedHenry cloud.

## Built on Node.js

The FeedHenry back end is built on Node.js. That was a bit ahead of its time in 2010, but is quite fashionable now. Today Node.js is often used for building fast, scalable network applications. Node has an event-driven, nonblocking I/O model that makes it lightweight and efficient compared to, say, Java Server Pages or ASP.Net. Node lets developers make code asynchronous without the hassle of threads and synchronization. The Node community is growing fast, with more than 70,000 public Node modules in the ecosystem. Enough Node modules deal with back-end data integration that FeedHenry can boast lots of integration points without having to build many of them internally.

On the downside, Node.js can be tricky to debug. As a weakly typed dynamic language, JavaScript doesn't give you much in the way of bug checking prior to deployment, although the various JavaScript linting tools can help.

With FeedHenry 3, the online environment integrates directly with GitHub. This carries a number of meanings, all of them good. You can get access to your source code from your own computer and develop offline when it's convenient; check your local code back in, and it will be reflected in the online repository. Teams can collaborate on both the cloud and app sides of a FeedHenry project without stepping on one another's changes. You can also build binaries of an app in the cloud, as I'll discuss later.

Many proponents of agile development insist that teams need to be located in one place and have continuous verbal communication. There's a certain amount of truth to that, and I've seen it work amazingly well. On the other hand, many projects -- proprietary enterprise software as well as open source software projects, including outsourced projects -- are developed successfully by internationally distributed teams. GitHub is one of the most widely accepted version control services for globally distributed software development, and I think the way FeedHenry has integrated with GitHub is a step in the right direction. International teams often use a bug reporting and ticketing system, as well as a source code control system. It would be useful for FeedHenry to integrate with one of those, too.

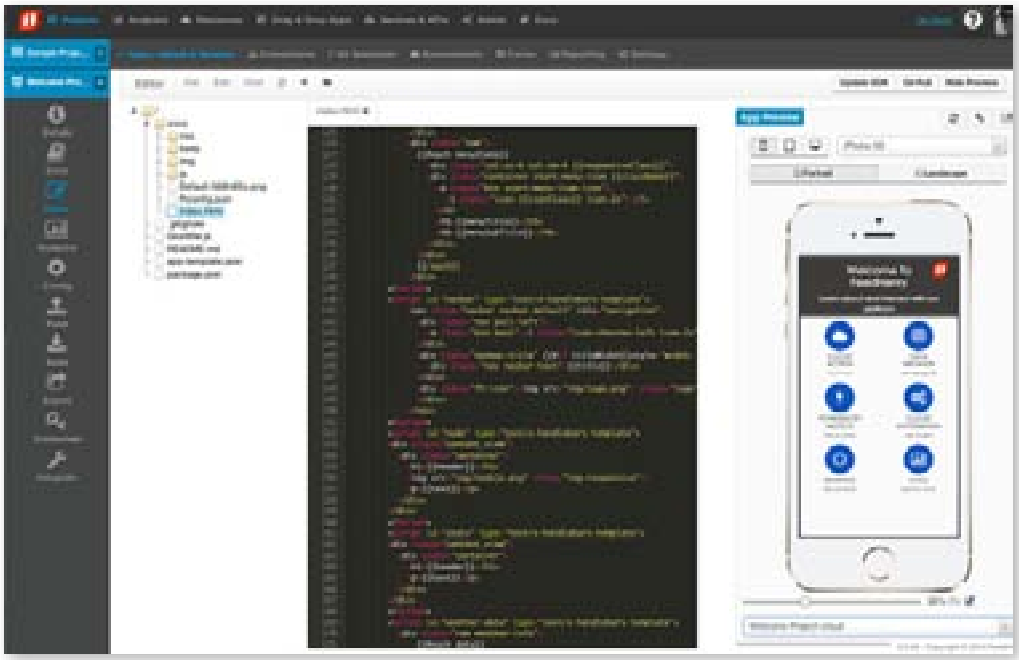
## API management and MBaaS

MBaaS is a small part of FeedHenry's bag of tricks. Nevertheless, FeedHenry offers a strong mobile back-end service. Part of what makes it scalable is the use of Node.js, which I discussed above. Further, FeedHenry uses MongoDB for its data store, which is also highly scalable. FeedHenry's servers typically don't even break a sweat under Black Friday-level loads.

FeedHenry back-end code is relatively simple, if you understand Node.js. Here's an example main (application.js) from the automatically generated Welcome app:

```
var mbaas = require('fh-mbaas-express');
var express = require('express');
// Securable endpoints: list the endpoints which you want to make securable here
var securableEndpoints = ['hello'];
var app = express();
app.use('/sys', mbaas.sys(securableEndpoints));
app.use('/mbaas', mbaas.mbaas);
app.use('/cloud', require('./lib/cloud.js')());
// You can define custom URL handlers here, like this one:
app.use('/', function(req, res){
  res.end('Your Cloud App is Running');
});
// Important that this is last!
app.use(mbaas.errorHandler());
var port = process.env.FH_PORT || process.env.VCAP_APP_PORT || 8001;
var server = app.listen(port, function(){
    console.log("App started at: " + new Date() + " on port: " + port);
});
```

You'll note the lack of explicit asynchronous code in this example, yet it's highly scalable stuff. You'll also recognize the use of the open source Express Web application framework.



**FeedHenry includes an online editor, supporting offline tools, and a command-line interface. Here we see the mobile app, with a code editor in the middle of the screen and a preview at right. You can configure the back-end service in another pane of the online interface.**

## Toolkit support

FeedHenry 3 supports native SDKs for iOS, Android, and Windows Phone 8, along with hybrid apps using Apache Cordova, HTML5 mobile Web apps, and Sencha, Xamarin, and Appcelerator Titanium. The way the JavaScript interface to the FeedHenry cloud works, it would be hard to find a JavaScript framework that isn't compatible.

When writing for FeedHenry in JavaScript, you include the feedhenry.js script in your HTML, initialize it with $fh.init, then call cloud functions from the $fh namespace. For example:

```
$fh.act({
  act: 'sayHello'
}, function(res) {
  alert("Cloud says : " + JSON.stringify(res.
say));
}, function(msg, params) {
  alert('An error occured: ' + msg);
});
```

This client code assumes that your cloud code has implemented and exported a sayHello function.

Not all apps that use an MBaaS started out that way. FeedHenry understands that, and thus provides a mechanism for importing apps, which offers a choice of native iOS, native Android, PhoneGap, basic Web app, and advanced Web app import types. Import methods include cloning an existing Git repo, uploading a Zip, and creating an app with a blank repo, then pushing the code there.

## Builders, templates, and integrations

One of the headaches that comes with mobile development, especially hybrid mobile development, is installing and maintaining the SDKs for all the mobile platforms you wish to target. Adobe PhoneGap Build offers one solution for this. FeedHenry offers its own answer.

The FeedHenry build service can turn an HTML5 app into binaries for Android, BlackBerry, iPhone, iPad, iOS (universal), and Windows Phone. Each binary can connect to one of your MBaaS instances, and it can be built for development, distribution, release, or debugging, depending on the platform. For iOS builds, you need to supply the appropriate credentials.

FeedHenry 3 adds a drag-and-drop form builder with a good assortment of templates to use as starting points. While forms are not the most flexible kind of mobile application, they are appropriate for many business applications requiring data input. Forms built via drag-and-drop, such as the one FeedHenry implements, can become simple apps very quickly -- often within hours.

On the other hand, I found only a few full-fledged app templates. According to the company, new clients currently receive some mentoring or training to help get their first real app off the ground, but FeedHenry is in the process of developing templates for industries such as health care, where there is more interest in purchasing solutions than in purchasing platforms or tools.

FeedHenry 3 supports Apache Cordova 3 and its plug-ins for building hybrid apps for iOS, Android, and Windows Phone 8. If you want to specify the plug-ins used, add a `config.json` file with a plugins key to your project. If you don't specify the plug-ins you want, FeedHenry will use a standard list of 30 or so plug-ins, which will most likely increase your app building time and the size of your app.

FeedHenry lists more than 50 Node.js plug-ins in its curated modules list. That list includes interfaces to most major relational and NoSQL databases, as well as to Amazon, Google, Rackspace, Salesforce and other SaaS providers, several messaging providers, several social networks, and assorted tools. In general, the back-end integration options look strong, although I haven't tested any of them in depth.

If the curated list doesn't contain what you need, more than 50,000 module and code snippets are available, contributed by the Node.js community.

Once you have integrated a back-end server for one project, you can expose your RESTful APIs to any of your other projects.

## Cloud deployment and management

FeedHenry runs on all major public and private clouds, and on a wide range of IaaS and PaaS infrastructures. FeedHenry has a HIPAA-compliant cloud, as well as live clusters in both Europe and North America.

The company claims its cloud portability eliminates vendor lock-in. That's probably true with respect to the cloud vendors, but not for FeedHenry itself. Cloud portability is also not unique to FeedHenry. Among competing MBaaS offerings, Kinvey can say the same.

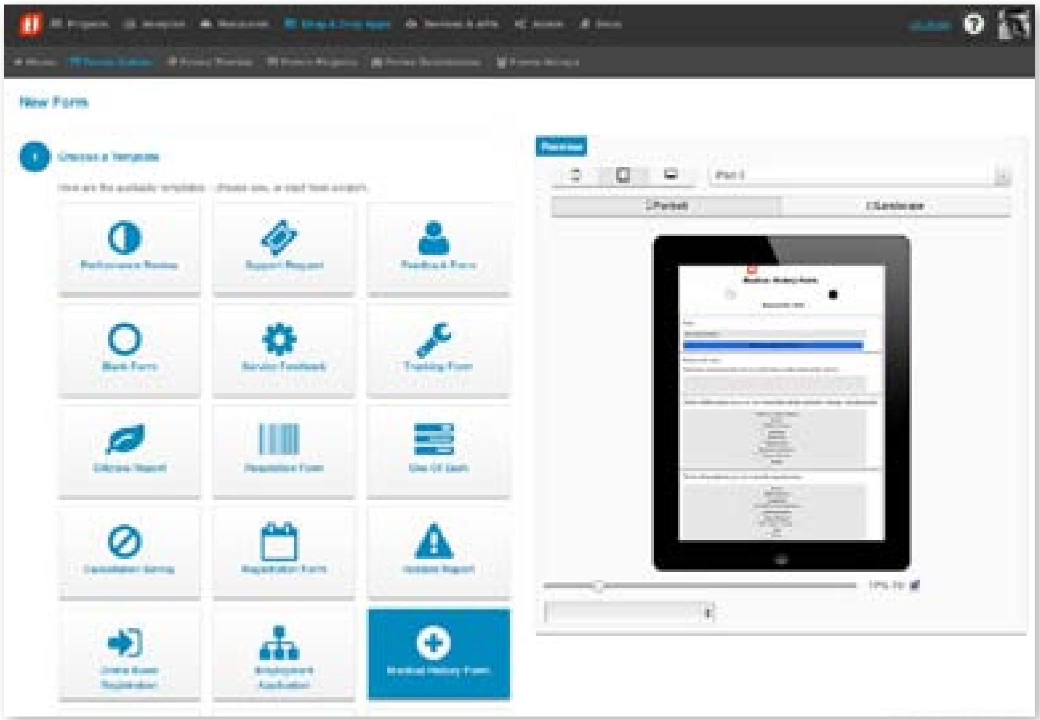
FeedHenry can encrypt locally cached data on the client with the use of AES/RSA algorithms, it supports HTTPS encryption on the pipeline between the app and the cloud, and it provides endpoint security. It has a full set of authentication and session management APIs, including support for LDAP, Active Directory, and OAuth credentials. Of course, you can find plenty of encryption plug-ins for Node.js, which can be used as needed.

Another potential security hole is between the MBaaS and the enterprise's own back-end systems. FeedHenry supports IP address punch-through, VPNs, firewalls, DMZs, approved data centers, and approved data center locations to address this issue.

Incorporating a supported third-party enterprise mobility management product, such as AirWatch or MobileIron, is a simple matter of checking a box in the FeedHenry distribution configuration.

Finally, FeedHenry does a good job of reporting app and cloud usage, including app installs, app start-ups, cloud requests, and active users by time, platform, and geography. In addition, FeedHenry monitors cloud endpoints in real time and provides full auditing. On the other hand,

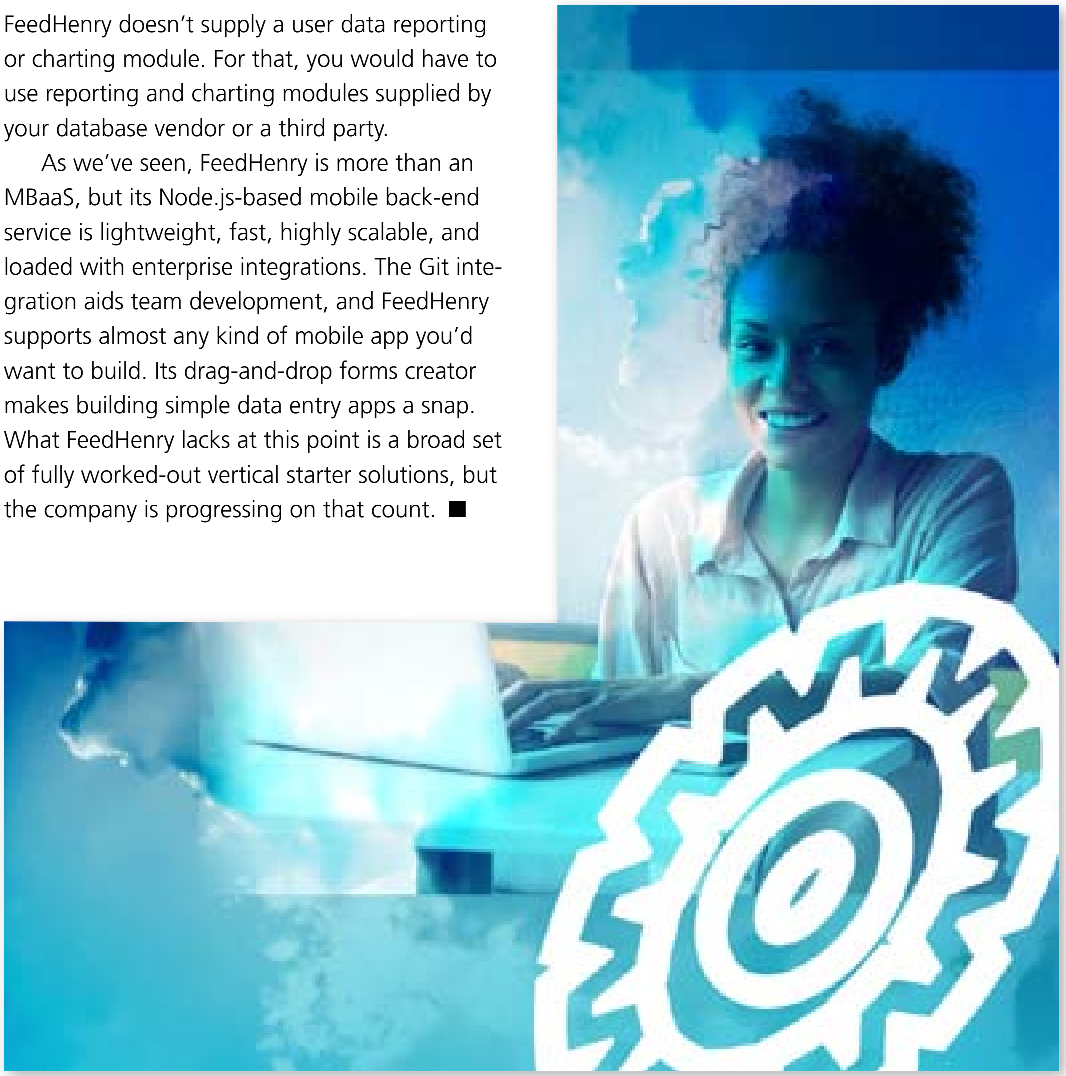
**FeedHenry 3 adds a drag-and-drop form builder. Once you have created a FeedHenry form, you can graphically edit the form, pages, fields, rules, and notifications online.**



| InfoWorld Scorecard | Integrations (20%) | Client support (20%) | Value (10%) | Back-end services (20%) | Ease of use (20%) | Monitoring (10%) | Overall Score |
|---|---|---|---|---|---|---|---|
| FeedHenry 3 | 9 | 9 | 8 | 9 | 8 | 8 | 8.6 ★★★★☆ |

FeedHenry doesn't supply a user data reporting or charting module. For that, you would have to use reporting and charting modules supplied by your database vendor or a third party.

As we've seen, FeedHenry is more than an MBaaS, but its Node.js-based mobile back-end service is lightweight, fast, highly scalable, and loaded with enterprise integrations. The Git integration aids team development, and FeedHenry supports almost any kind of mobile app you'd want to build. Its drag-and-drop forms creator makes building simple data entry apps a snap. What FeedHenry lacks at this point is a broad set of fully worked-out vertical starter solutions, but the company is progressing on that count. ■



# Kinvey boosts enterprise mobile apps

*Kinvey pairs rich mobile client and tools support with flexible back-end services, but external integrations are limited*

**Kinvey bills itself as** a complete mobile and Web app platform. It has extensive client support, integrates with the major enterprise databases, and offers a back-end data store, a file store, push notifications, mobile analytics, iBeacon support, and the ability to run custom code on the back end.

According to the company, Kinvey sells to IT as its primary customer because it provides an enterprise platform, not for one or two apps but for tens and hundreds of apps for an enterprise. However, it also engages and supports the developer community app by app. Thus pricing is available for individuals and independents, as well as for smaller businesses and large enterprises.

## Client support

Kinvey supports native, hybrid, and HTML5 apps. It has native toolkit support for iOS and Android. In addition, it supports HTML5, AngularJS, Backbone.js, Node.js, Apache Cordova/PhoneGap, and Appcelerator Titanium, and it provides a REST API. (PhoneGap is compatible with AngularJS 1.2.3 and later, and with all versions of Backbone.js.)

To set up a native iOS 6 or iOS 7 project to use the Kinvey back end, you need to download the KinveyKit framework, install it into your Xcode project, and set your project to link with the eight required libraries. You should also copy the KinveyKit documentation into Xcode's documentation directory. Then you'll need to import KinveyKit in your code and call `[KCSClient sharedClient] initializeKinveyServiceForAppKey` to connect your app with the Kinvey service, using the correct app key, app secret, and options. You can also use the Kinvey REST API in iOS projects.

To set up a native Android 2.3 or higher Kinvey project, download the latest Kinvey library and copy the JAR files into your project libs folder. To initialize the Kinvey service, instantiate a new `Client.Builder()`, with a `kinvey.properties` file present in your projects assets folder that holds the correct app key, app secret, and options.

For most HTML5 apps and JavaScript frameworks, you'll want to serve the appropriate Kinvey JavaScript library from Kinvey's content delivery network as part of your app initialization. Then you'll need to call `Kinvey.init()` or `$kinvey.init()`, depending on the framework, using the correct app key, app secret, and options.

For the REST API, you'll probably start with `GET /appdata/:appKey`, using basic authentication over HTTPS, to handshake with the service. The rest of your logic will need to start with a login request to collect an authorization token from the service, then all the other REST calls will use the authorization token.

Kinvey uses Promises to manage asynchronous JavaScript flows, simplifying the client code. Basically, a Promise is a proxy object for the result of an asynchronous operation. When you're ready to use the result, the then method can be called to process a successful operation, and the else method can be called to process an unsuccessful operation. Promises are currently implemented in mobile browsers and the Chrome, Firefox, and Opera desktop browsers. According to Kinvey, even the sort of enterprise that is still standardized on IE6 for its Web apps is able to use Promises in AngularJS PhoneGap apps.

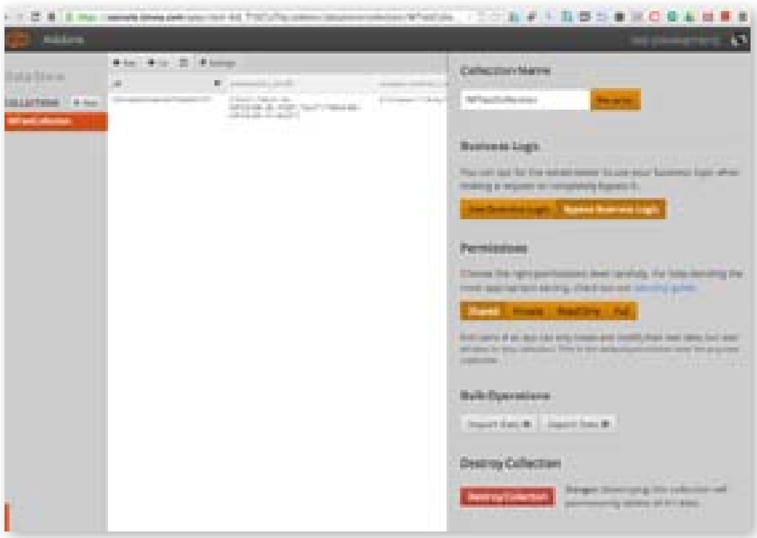
## Cloud code

Kinvey cloud code is written in JavaScript. In addition to using standard JavaScript and external services, it can use Kinvey APIs for logging, collection access, sending push notifications, sending email, validating requests, date and time functions, doing asynchronous processing, rendering a Mustache template, and obtaining the back-end context. Cloud code can live in hook processing functions and custom endpoints. Cloud code is versioned internally in Kinvey.

As a simple example, the following custom cloud endpoint handler responds to any request by asking for the real-time information feed for the Boston MBTA Red Line subway, parsing the JSON response, and sending that JSON to your client.

```
    function onRequest(request,response,mod
ules){
        var req = modules.request;
        req.get('http://developer.mbta.com/Data/
Red.json', function(error, resp, body){
            if (error){
                response.body = {error: error.
message};
                response.complete(400);
                return;
            }
            response.body = JSON.parse(body);
            response.complete(resp.status);
        });
    }
```

Scheduled code can run against your custom endpoints. Scheduled code is commonly used to aggregate, archive, and clean up data; to pull data from a third-party API into Kinvey; or to send out a batch of emails or push notifications.

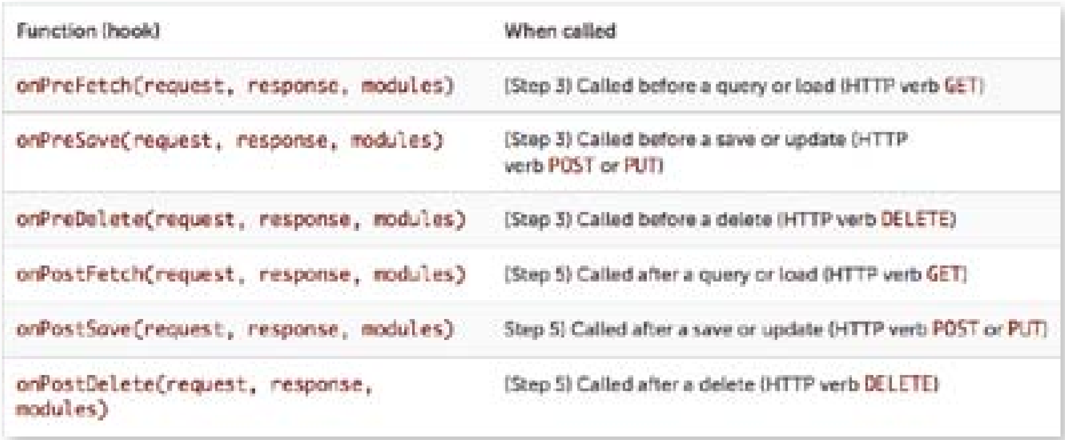


**Kinvey collections use MongoDB, which provides a schema-less, no-SQL database for use by your apps. This screen lets you create and design collections (only the creation step is necessary) and choose whether to enable or bypass your database business logic.**

Kinvey has flexible, hook-based, back-end business logic. The hook-processing pipeline runs as follows:

1. Request from the client to the Kinvey back end
2. Authentication of the client (verify that client has access to this back end)
3. Pre-processing hooks
4. Database actions
5. Post-processing hooks
6. Clean-up and response formatting
7. Response returned to the requesting client

Kinvey has six hook-processing functions that you can implement:



| Function (hook) | When called |
|---|---|
| `onPreFetch(request, response, modules)` | (Step 3) Called before a query or load (HTTP verb GET) |
| `onPreSave(request, response, modules)` | (Step 3) Called before a save or update (HTTP verb POST or PUT) |
| `onPreDelete(request, response, modules)` | (Step 3) Called before a delete (HTTP verb DELETE) |
| `onPostFetch(request, response, modules)` | (Step 5) Called after a query or load (HTTP verb GET) |
| `onPostSave(request, response, modules)` | Step 5) Called after a save or update (HTTP verb POST or PUT) |
| `onPostDelete(request, response, modules)` | (Step 5) Called after a delete (HTTP verb DELETE) |

Kinvey only guarantees that the preprocessing hooks will be called, as an error in the database actions will cause the postprocessing hooks to be skipped.

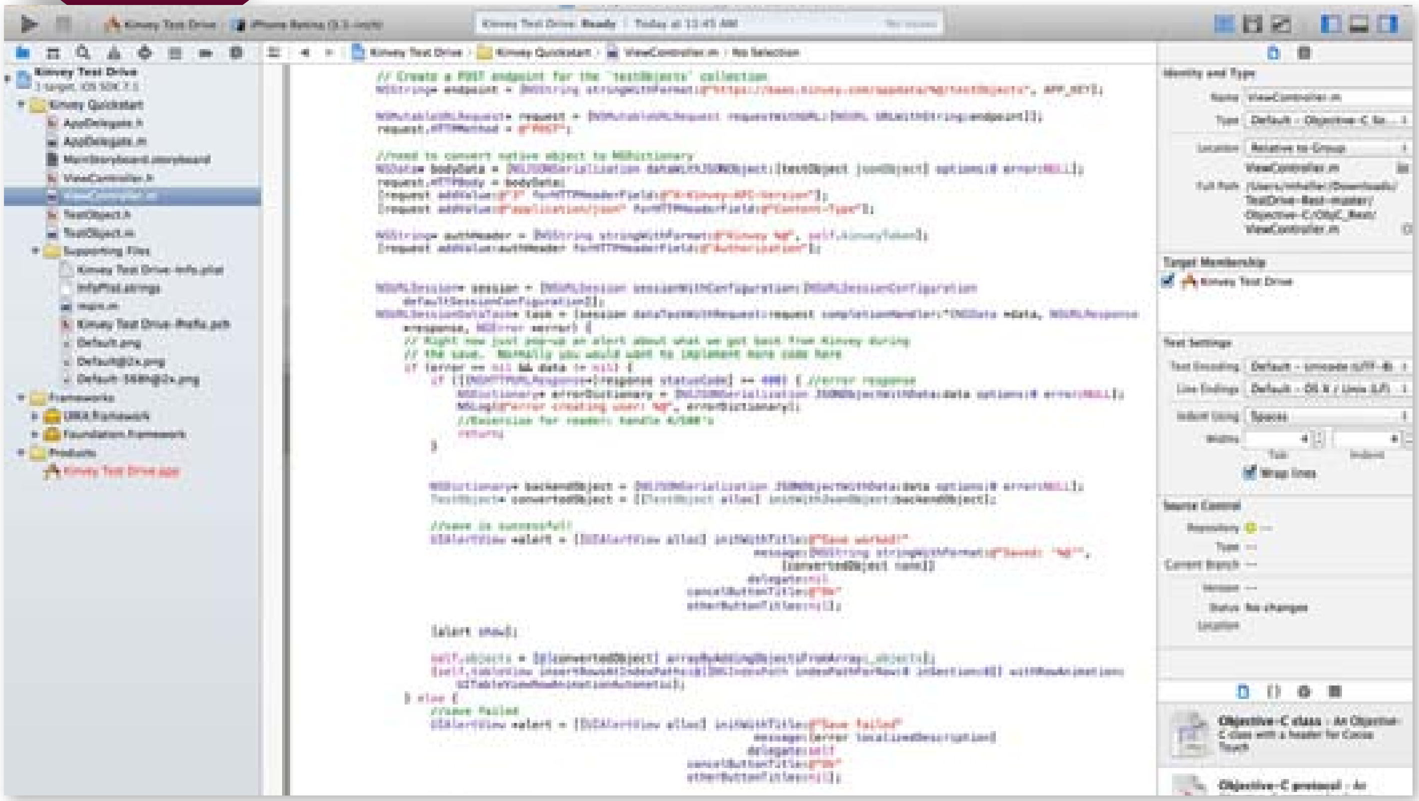
## Cloud database, data links, and integrations

Kinvey implements a simple JSON key-value data store with entities and collections, using MongoDB. In the business and enterprise versions, Kinvey also has data links for Oracle, SQL Server, Salesforce, and Microsoft Dynamics CRM, plus a generic connector. The data link architecture is defined as a REST API.

Data links connect a MongoDB collection specifically marked for data integration with your database, and forward CRUD (create, read, update, and delete) requests to the database. (A few customers have used a data integration collection to cache the data for performance reasons, but mostly the data integration collection does not hold onto the data.)

Kinvey currently supports two classes of integrated services: location services like Google Places and Foursquare, and social services like Facebook Open Graph. That doesn't mean you're restricted to public APIs, but it does mean that you can't use other classes of service, except for the data and auth links we've already discussed.

Push messaging support in Kinvey requires you to perform external configuration in Google Cloud Messaging for Android or Urban Airship for iOS. Once you've done that and connected the service to your Kinvey cloud app, Kinvey takes care of all push messaging requests through its own APIs. Note that push messages will fail in the iOS emulator, but should succeed in real devices and in an Android emulator with the Google API enabled.



```
// Create a POST endpoint for the `testObjects` collection
NSString* endpoint = [NSString stringWithFormat:@"https://baas.kinvey.com/appdata/%@/testObjects", APP_KEY];

NSMutableURLRequest* request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:endpoint]];
request.HTTPMethod = @"POST";

//need to convert native object to NSDictionary
NSData* bodyData = [NSJSONSerialization dataWithJSONObject:[testObject jsonObject] options:0 error:NULL];
request.HTTPBody = bodyData;
[request addValue:@"3" forHTTPHeaderField:@"X-Kinvey-API-Version"];
[request addValue:@"application/json" forHTTPHeaderField:@"Content-Type"];

NSString* authHeader = [NSString stringWithFormat:@"Kinvey %@", self.kinveyToken];
[request addValue:authHeader forHTTPHeaderField:@"Authorization"];


NSURLSession* session = [NSURLSession sessionWithConfiguration:[NSURLSessionConfiguration
    defaultSessionConfiguration]];
NSURLSessionDataTask* task = [session dataTaskWithRequest:request completionHandler:^(NSData *data, NSURLResponse
    *response, NSError *error) {
    // Right now just pop-up an alert about what we got back from Kinvey during
    // the save.  Normally you would want to implement more code here
    if (error == nil && data != nil) {
        if ([(NSHTTPURLResponse*)response statusCode] >= 400) { //error response
            NSDictionary* errorDictionary = [NSJSONSerialization JSONObjectWithData:data options:0 error:NULL];
            NSLog(@"error creating user: %@", errorDictionary);
            //Excercise for reader: handle 4/500's
            return;
        }


        NSDictionary* backendObject = [NSJSONSerialization JSONObjectWithData:data options:0 error:NULL];
        TestObject* convertedObject = [[TestObject alloc] initWithJsonObject:backendObject];

        //save is successful!
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Save worked!"
                                                        message:[NSString stringWithFormat:@"Saved: '%@'",
                                                                 [convertedObject name]]
                                                       delegate:nil
                                              cancelButtonTitle:@"OK"
                                              otherButtonTitles:nil];

        [alert show];

        self.objects = [@[convertedObject] arrayByAddingObjectsFromArray:_objects];
        [self.tableView insertRowsAtIndexPaths:@[[NSIndexPath indexPathForRow:0 inSection:0]] withRowAnimation:
            UITableViewRowAnimationAutomatic];
    } else {
        //save failed
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Save failed"
                                                        message:[error localizedDescription]
                                                       delegate:self
                                              cancelButtonTitle:@"OK"
                                              otherButtonTitles:nil];
```

**Here we see an example Kinvey app for iOS, using the REST API, opened in Xcode. This is a bare-bones demo that does nothing other than save and load some data to and from the cloud-based Kinvey data store.**

Kinvey handles offline data synchronization in a basic way. You can enable sync in your application initialization options and tell Kinvey when your application goes online and offline.

```
Kinvey.init({
    appKey : 'App Key',
    appSecret : 'App Secret',
    sync : {
      enable : true,
      online : navigator.onLine // The initial application state.
    }
});
// Switch application state when the on- and offline events fire.
$(window).on({
    offline : Kinvey.Sync.offline,
    online : Kinvey.Sync.online
});
```

Kinvey has an automated control setup for offline data synchronization, in which data is automatically pulled from the cache if the application is offline. If the application is online, data is pulled from the network and stored in the cache. You can turn this off and handle it yourself. You can also set important parameters such as the cache `maxAge`, which determines when a cached value expires and must be refreshed from the network.

Synchronization of offline data requires care in conflict resolution. Using automated control, your Kinvey app will attempt to synchronize any locally stored data when the device goes online again, but if the server data has also changed you'll have a conflict. You can set your conflict resolution policy to clientAlwaysWins, serverAlwaysWins, or a custom conflict resolution function.

## Cloud deployment and management

Kinvey supports deploying on almost any cloud, including private clouds. That includes deploying to HIPAA-compliant facilities and facilities located entirely in the EU. Even Kinvey's multitenant cloud is considered secure enough for most apps, as the company does end-to-end encryption, and companies that use data links can keep their data in databases behind their own firewalls. If you have a Google App Engine server,

Overall, Kinvey is a worthy competitor to FeedHenry for enterprises.

you can link it to your Kinvey back end.

Authentication can be done internally by Kinvey, or through LDAP or Active Directory in the business and enterprise versions. Kinvey also supports Facebook, Twitter, Google+, and LinkedIn identities through OAuth.

In addition to the collection-level permissions shown in Figure 5, Kinvey supports entity-level permissions, global access control, and reader/writer lists. If you are using Kinvey's User Groups in your app, you can manage group reader/writer permissions.

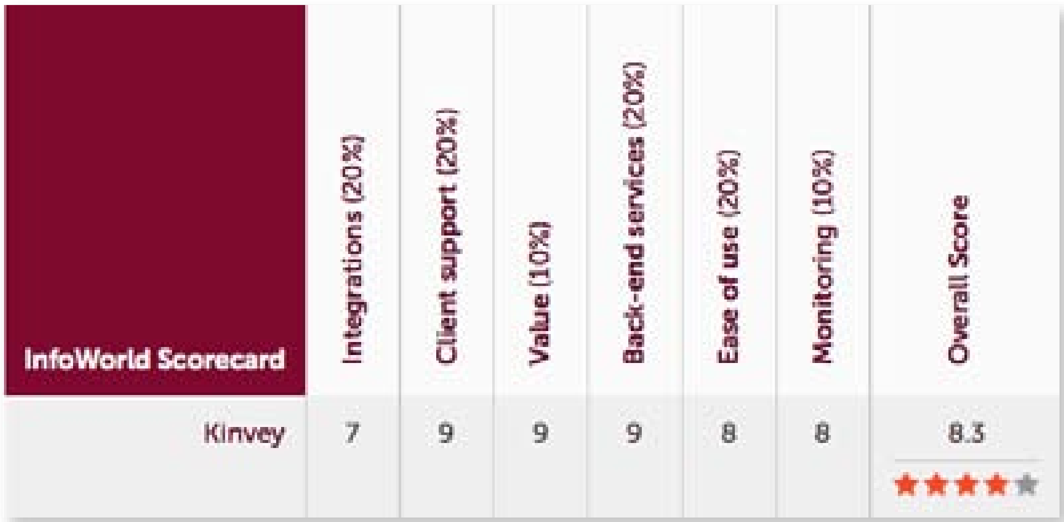
The online Kinvey documentation includes guides, tutorials, samples, and references. In general, I was able to find the information I needed, but it wasn't always easy. All the documentation is organized by the client technology. That helps to keep you from being confused by seeing different notations at once, but makes it hard to compare the capabilities and samples of different clients. According to the company, most of Kinvey's users already have a favorite client technology, which means that organizing the documentation by client is the right thing to do. However, Kinvey is now developing more documentation in the form of high-level white papers and other traditional publications, to meet the expectations of enterprises.

Kinvey offers an online app cost estimator, which is designed to illustrate the development savings you can realize from using an MBaaS. If you pick all the highest-complexity choices (five client platforms, 12 pages in the app, and so on), this estimator will tell you that a DIY app would cost you about half a million dollars, while an app using Kinvey for the back end would cost about half that. Take that with a grain of salt, and look at the formulas behind the estimates before giving the results any credence.

Bearing in mind the potential savings, Kinvey's pricing is straightforward for independents and smaller businesses, starting with free and running to $1,500 per month per back end. Note that many apps can share a single back end, so this pricing is much more favorable to the customer than per-app plans. According to the company, the $1,500-per-month business plan can be considered a starting point for negotiations of enterprise plans.

Kinvey supplies basic analytics and usage metrics for users, storage, and API calls for all plans. Kinvey's Premium Analytics add-on, which I did not test, enables you to drill down from high-level aggregate analytics, through a host of different user segments, to the behaviors and actions of individual users.

Overall, Kinvey is a worthy competitor to FeedHenry for enterprises. From the perspective of an agency, independent developer, or small business, Kinvey offers reasonably priced starter plans, while FeedHenry is laser-focused on enterprise customers. Kinvey lacks the nice online drag-and-drop forms builder found in FeedHenry, but FeedHenry forms are not useful for serious, large apps -- they are mostly for simple data collection apps, such as customer surveys or site inspections. ■



| InfoWorld Scorecard | Integrations (20%) | Client support (20%) | Value (10%) | Back-end services (20%) | Ease of use (20%) | Monitoring (10%) | Overall Score |
|---|---|---|---|---|---|---|---|
| Kinvey | 7 | 9 | 9 | 9 | 8 | 8 | 8.3 ★★★★☆ |

Deep Dive

# Parse delivers on mobile apps, but not for business

*Facebook's MBaaS shines for consumer mobile applications, but misses some must-haves for business apps*

**Parse was once the** poster child for mobile back end as a service (MBaaS), and despite its recent acquisition by Facebook, it is still a viable, low-friction MBaaS for limited-volume consumer apps. On the plus side, it is well documented, it has good native client support, and it has a JavaScript client SDK based on Backbone.js. Parse also runs JavaScript code on the back end, which offers developers the option of an all-JavaScript application stack.

On the minus side, Parse is missing big pieces that are necessary for business apps, such as data integration, offline operation, and online/offline synchronization. At the same time, its pricing seems geared to lower-volume apps. If you happen to create the next viral hit, Parse will cost you plenty as the usage of your app picks up.

## Client support

Parse supports native mobile, JavaScript, and desktop apps. On the mobile side, it has native support for iOS, Android, and Windows Phone 8. On the desktop, it supports OS X and Windows 8 (.Net), as well as Unity games.

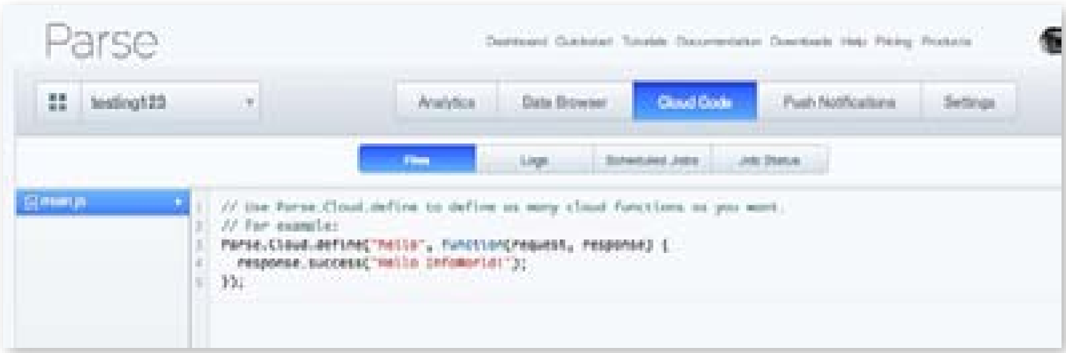
The Parse JavaScript SDK is based on the Backbone.js framework. According to the company, most existing Backbone.js apps can work with Parse, pending only minor changes. Parse also has a REST API, which can be made to work on almost any client, in any language that supports HTTP requests, including the Curl command-line utility.

In addition to its own official client API libraries, Parse has a number of third-party, community-supported client libraries. These include support for .Net, ActionScript, Appcelerator Titanium, Clojure, Corona, Java, additional JavaScript APIs, Temboo, PHP, Python, Qt, RealBasic, Ruby, and WebOS.

## Cloud code

Parse lets you run JavaScript code in the cloud, using the same Parse JavaScript SDK as its client, which is based on the Backbone.js framework. Rather than have you routinely edit your cloud code in a browser, à la FeedHenry and Kinvey, Parse supplies a command-line tool for managing code in Parse Cloud and allows you to use your favorite JavaScript editor on your computer. However, you can view your code in your Dashboard, which is also where you can view your logs.

On OS X and Linux, a single tool, called Parse, installs to /usr/local/bin/parse. On



```
// Use Parse.Cloud.define to define as many cloud functions as you want.
// For example:
Parse.Cloud.define("hello", function(request, response) {
  response.success("Hello InfoWorld!");
});
```

**A Parse Cloud application log viewed in the Parse Dashboard. The same information can be viewed from the command-line interface.**

Windows, the Parse tool installs a ParseConsole, which launches a Parse-aware PowerShell session; the first session adds Parse to PowerShell for future use. The command-line tool Parse is an app scaffold generator, app deployment tool, log printer, app rollback tool, and self-updater.

Your cloud code resides in `main.js` after running `parse new`. The default is a "hello, world" program:

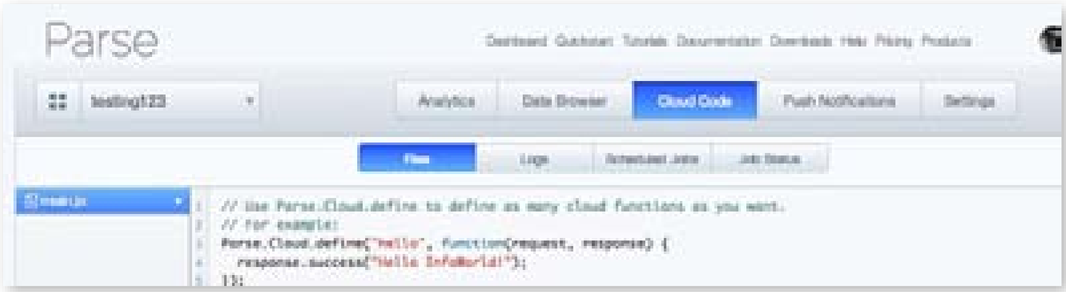
```
   Parse.Cloud.define("hello",
function(request, response) {
   response.success("Hello world!");
   });
```

You can push this to the cloud with parse deploy. Once it has successfully deployed, you can use any Parse client to run it, including a simple REST call. The tutorial in your Parse Web console will generate sample code for you with the correct credentials in several languages. Here's the Python version with my credentials obscured:

```
   importjson,httplib
   connection = httplib.
HTTPSConnection('api.parse.com',
443)
   connection.connect()
   connection.request('POST', '/1/
functions/hello', json.dumps({
         }), {
           "X-Parse-Application-Id":
"TRZxxxxxx",
           "X-Parse-REST-API-Key":
"Ajyetxxxxxx",
           "Content-Type": "applica-
tion/json"
         })
   result = json.loads(connection.
getresponse().read())
   print result
```

"Hello, world" does nothing but demonstrate that you can call code in the Parse Cloud, but Cloud functions can be useful if you pass them parameters and have them do database lookups and calculations on the data. If you need to do more complicated tasks, you'll probably want to break your application up into multiple .JSfiles and load them from `main.js` using `require()` statements.

**Parse Cloud code is written in JavaScript. Here we see version 2 of "Hello" in the Parse Dashboard, as uploaded from my computer.**



```
// Use Parse.Cloud.define to define as many cloud functions as you want.
// For example:
Parse.Cloud.define("hello", function(request, response) {
  response.success("Hello InfoWorld!");
});
```

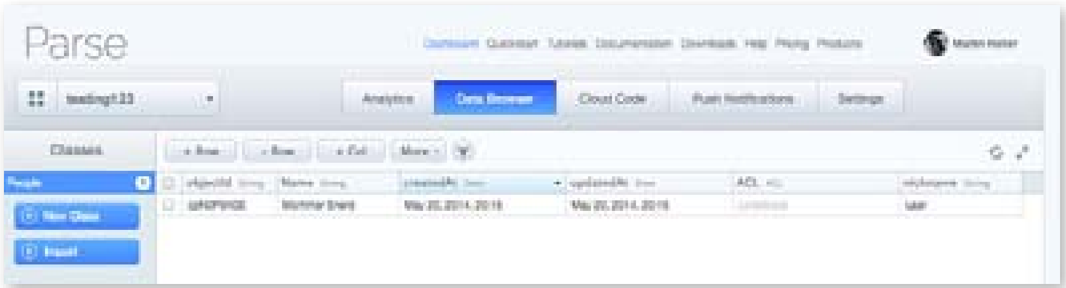
## Key/value storage

Key/value pair storage in the Parse cloud is simple to use. The details vary with the client SDK, but this Java code for Android is typical:

```
   ParseObject testObject =
newParseObject("TestObject");
   testObject.put("foo", "bar");
   testObject.saveInBackground();
```

The resulting data can be retrieved from any app with access to the data and viewed in the Parse Dashboard.

Parse uses a NoSQL data store, but it supports relationships: one to one, one to many, and many to many. These can be implemented with pointers, arrays, Parse relations, and join tables. Parse supports nine simple data types, including null, and a given column can be any data type when first stored. However, Parse will lock the type of that field to the initial type after the first value has been stored. There are two ways to store binary data in Parse: as a byte stream or as a file.



**The Parse Cloud data browser lets you import bulk data; add classes, columns, and rows; and view filtered data.**

You can define `Parse.Cloud.before-Save` handler functions to perform server-side data validation, and possibly apply data-modification rules, such as restricting the length of strings or removing forbidden characters. To take actions after data has been saved, define `Parse.Cloud.afterSave` handler functions. Similarly, you can control object deletion by handling the `Parse.Cloud.beforeDelete` event, and take action after object deletion, such as logging, with a `Parse.Cloud.afterDelete` handler. These event handlers have much the same flavor as Kinvey's hook-processing functions.

Parse Cloud functions will be killed after 15 seconds of wall clock time. The beforeSave, afterSave, beforeDelete, and afterDelete functions will be killed after three seconds of runtime. To get around these limits, you can define a background job, Parse.Cloud.job. Background jobs are terminated after 15 minutes of runtime. You can schedule background jobs from your Parse Dashboard.

Standard Parse Cloud functions take parameters in JSON. If you need to use a different format, you can write custom Webhooks and call the Express Web application framework to process input.

## Push notifications

Parse can send push notifications to iOS, Android, Windows 8, and Windows Phone 8. In each case, you'll have to provision your push server, then provide the certificate or credentials to your app. For iOS, you need to provision on the Apple Developer site. For Android, you use Google Cloud Messaging where supported by the device; otherwise the messages come from the Parse server. Windows RT and Windows Phone apps can receive push notifications from Microsoft push servers. A JavaScript client can't receive push notifications.

## Users and roles

Parse has a fairly complete user system predefined, including the usual sign-up and email verification, along with a provision for anonymous users. A system of ACLs controls what data individual users can read and write. For more complicated use cases, Parse supports roles, with a separate layer of ACLs for the roles and a hierarchy of roles.

Not surprisingly, given that Parse belongs to Facebook, it has good support for social account linking -- including Twitter. In each case, you must have an app set up on the social networking platform to enable the OAuth authentication.

Parse supports in-app purchases only on iOS. Oddly, Parse currently supports a local data store only on Android, although support for a local data store on iOS is planned.

## Integrations

Parse boasts it can do double duty as a Web host. That's nice, but it isn't exactly a compelling consideration for choosing a mobile back-end service.

Parse has nine prefab integrations with other services. Three of them -- Mailgun, Mandrill, and SendGrid -- are for sending email. Stripe is for charging credit cards. Twilio sends SMS messages and voice messages. In addition, Parse has third-party modules for Cloudinary, Instagram, and Paymill.

As far as I can tell, implementing enterprise data integration with Parse requires writing a REST Web service wrapper for the data source and a JavaScript module for Parse. That isn't hard, but it isn't convenient, either. At one point, prior to the Facebook acquisition, Parse had a Web page that talked about how you could do enterprise data integration yourself. That's gone now, and Parse isn't even pretending to have enterprise data integration.

I haven't seen any options for hosting Parse other than its own multitenant cloud. I can't see it being used for apps that need to be HIPAA-compliant or for apps restricted to data located in the European Union.

Parse now has usage-based pricing, ranging from free for low usage to $1,700 per month per app for 200 requests per second, plus five centers per 1,000 unique push messaging recipients per month over the first million. That's quite reasonable for many apps, but I can see a popular consumer app blowing through the limits, and I can't see a large business wanting those kinds of limits for its important apps. On the other hand, vendors of successful apps and large businesses are often in a position to negotiate pricing.

While I don't see Parse as the top MBaaS option for most businesses, I can see Parse as an easy, low-cost way to prototype the back end of a mobile app, especially a consumer app. The questions in my mind are whether it makes sense to start with a back end that lacks important capabilities that you may need later, and whether it makes sense to start with a back end that may become too expensive for an app that's popular but not a big moneymaker. ■

**Martin Heller** *is a contributing editor for InfoWorld reviews.*

While I don't see Parse as the top MBaaS option for most businesses, I can see Parse as an easy, low-cost way to prototype the back end of a mobile app, especially a consumer app.

| InfoWorld Scorecard | Integrations (20%) | Client support (20%) | Value (10%) | Back-end services (20%) | Ease of use (20%) | Monitoring (10%) | Overall Score |
|---|---|---|---|---|---|---|---|
| Parse | 6 | 8 | 8 | 8 | 8 | 8 | 7.6 ★★★☆☆ |