

# Mobility Modernization

# Table of Contents

## **Introduction / 3**

## **The Challenges of Migrating to a Modern Mobile Architecture / 4**

Before you begin / 5

1. Assess your structure / 5
2. Assess your skills / 6

## **Strategies for Migrating to Mobile / 8**

Mobile Application Approaches / 8

Mobile Application Architecture / 10

The Bottom Line / 11

## **Strategies for Migrating to the Cloud / 12**

Data Architecture / 13

Addressing Your “Monolith” / 13

API Design / 14

Service Modularization & Granularity / 15

## **Data & Mobile Applications / 17**

Security / 18

App and API Authentication and Authorization / 18

Data Encryption / 19

Compliance / 19

Choosing the Right Cloud / 20

## **Future-Proofing Your Modernization Efforts / 21**

Performance Considerations / 21

Monitoring & Management Considerations / 22

Design with the Future in Mind / 23

## **Next Steps / 24**

## **Progress Reference Architecture / 25**

Going cloud-native with Progress / 25

# Introduction

As a technical architect, you play a critical role in stewarding the legacy systems that power the core of your business. Keeping these systems running well is essential to your organization's present, and future.

As a forward-thinking technologist, you also know that there are a number of trends in business and technology worth paying attention to. Mobility, cloud computing, and even emerging areas like Chatbots, AR/VR and the like have the potential to open up new paths to customers, and unlock new innovations within your organization.

Your challenge—**one that you are uniquely suited to tackle**—is how to keep your legacy systems stable, while moving your technology stack into the future, starting with mobile and the cloud. The purpose of this playbook is to provide you with concrete, actionable guidance that you can use when planning a mobile modernization effort.

We'll start this guide with a brief discussion about the challenges unique to mobile modernization, and tips for assessing your organizational design when considering mobility. Then, we'll follow-up with strategies for migrating to mobile, migrating to the cloud and approaches for future-proofing your efforts so that you can better embrace emerging trends when the time is right. By the end of this playbook, you'll be fully equipped with tips, strategies, visuals and checklists you can use to plan, discuss and execute a mobile modernization effort for your organization.

# The Challenges of Migrating to a Modern Mobile Architecture

Modern mobile applications are becoming the preferred window through which your employees and consumers interact with enterprise data. Whether email, calendars, and chat, or accessing CRM, ERP or financial data, your organization has likely already “gone mobile” as employees put their work in their pockets. While mobility projects may have existed on the POC fringe in your organization just a few years ago, their prevalence, and your customers’ and employees’ appetite for them means that these apps must be architected with the same care we’ve taken for traditional systems in the past.

What’s more, because modern workforces are increasingly physically mobile--the walls of an office and a modern workday no longer constrain a desire or need to work--there’s a need not just to create mobile interfaces, but mobilize access to essential data. Modern mobile apps need to connect to your monolithic or legacy systems in order to be useful. But it can be challenging to mobilize the data in those monoliths, not to mention move them, in entirety, to the cloud.

Finally, while mobile applications and cloud migration efforts share many common threads with traditional enterprise systems, both introduce unique performance, scale and security challenges that you need to be prepared for.



## Mobile Workforce

Mobile-applications are increasingly the preferred interaction channel for your customers and employees



## Beyond the Firewall

Modern workforces are moving beyond the walls of the office, yet still need access to critical enterprise data



## Challenges of the Cloud

Mobile and cloud modernization projects present unique performance, scale and security challenges that you need to plan for

These challenges, collectively, can be daunting, but need not be overwhelming.

By combining traditional architecture with a few modern practices, you can easily tackle those challenges, enabling you to modernize your systems and unlock new opportunities for your organization.

## Before you begin, consider the architecture of your organization.

First and foremost though, you should assess your organizational architecture. When undertaking a mobile modernization effort, it can be tempting to jump right into system design. But we recommend making sure you understand how the structure of your organization and skills of your teams will impact and influence your modernization efforts. By taking a few moments to consider your current state using the checklists below, you'll be better prepared to make key technology decisions down the road.

First and foremost though, you should assess your organizational architecture. When undertaking a mobile modernization effort, it can be tempting to jump right into system design. But we recommend making sure you understand how the structure of your organization and skills of your teams will impact and influence your modernization efforts. By taking a few moments to consider your current state using the checklists below, you'll be better prepared to make key technology decisions down the road.

### 1 Assess your structure

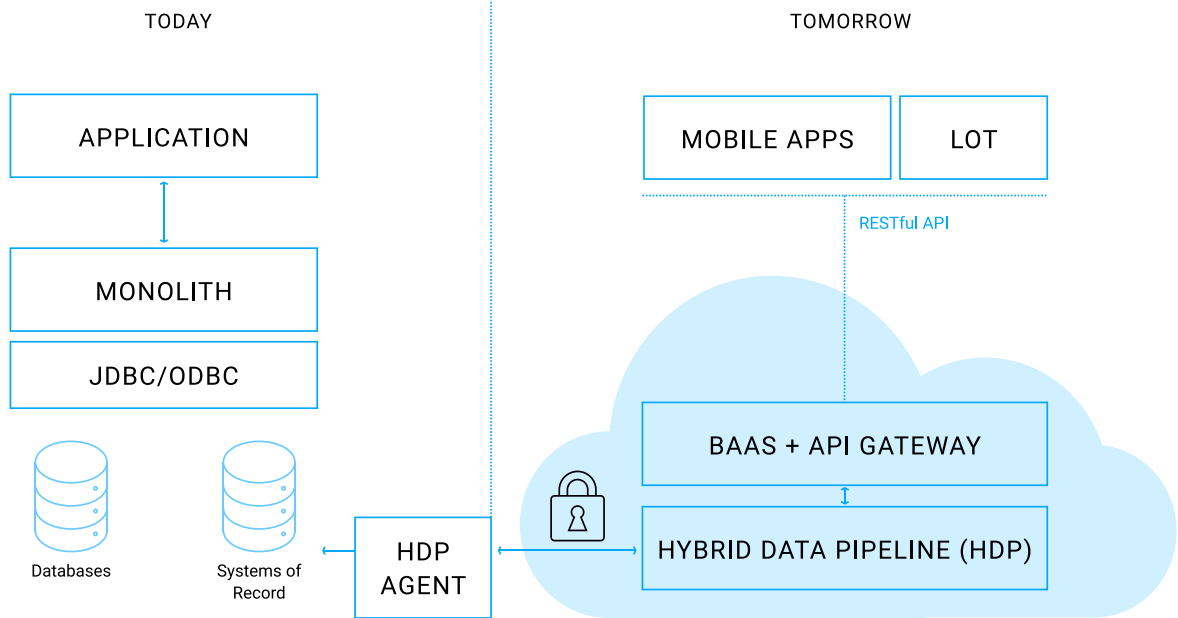
First, you'll want to pay close attention to how the makeup of your organization impacts your modernization efforts. The following are some key questions to ask, and advice for fitting your modernization effort into your organizational context.

Mobility team makeup	We are forming a new mobile team and either retraining existing staff or hiring mobile developers.	We are asking existing front-end teams to take on upcoming mobile projects.
Data & backend teams	We are forming a new API team that will work alongside our existing data & backend teams.	We are asking our existing data and backend teams to build a new API, while maintaining legacy interfaces
API roadmap	We are building a new API that wraps our legacy sources for use by modern applications.	Our new API will completely replace legacy interfaces from day one, affecting all applications, old and new.

**Table 1.** Personnel considerations for mobility modernization

It's important to note that modernization doesn't need to be an all-or-nothing approach, for your mobile apps, or your backend. Whether creating APIs and a microservices architecture or adding mobile projects to your existing slate of apps, we recommend taking a piecemeal approach that favors keeping your core running and stable, while building out new capabilities and services that wrap the core, without disruption.

Wherever possible, it's best to favor adding new services and apps alongside what you have in place, as opposed to wholesale replacement.



**Fig 1.** Moving monoliths to the cloud does not have to be “all or nothing.” Layering BaaS on top of existing legacy systems, and using secure pipes to access existing on prem data, minimizes potential disruption when taking the first step towards the cloud.

## 2 Assess your skills

Next, evaluate the skills of the teams that will be responsible for the development and management of key pieces of your mobilization effort, from the mobile apps themselves to cloud services, and APIs. In the realms of both mobile and data, consider the following questions before you begin your effort.

Regardless of how you answer the questions below, if there is some close overlap of skills across teams, we suggest unifying skill sets across the development stack for front-end, back-end and even mobile work. While there are several options, we believe unifying around JavaScript is an ideal choice. Both because of presence of well-supported, stable JavaScript runtimes and libraries across the stack, and the size of the JavaScript community, which presents a solid talent pool from which to grow your development teams, over time.

### Key Considerations

- Make sure you're aware of how the structure of your organization, especially with mobile, data and backend teams might affect your modernization efforts.
- Take a piecemeal approach to modernization that favors keeping your core running and stable, while building out new services that wrap the core.
- It's best to favor adding new services and apps alongside what you have in place, as opposed to wholesale replacement.

<b>Mobile Teams</b>	Our mobile teams will be/are staffed by front- and backend developers who are most comfortable with JavaScript.	Our mobile teams will be/are staffed by developers experienced with native mobile technologies.
<b>Data/Services Teams</b>	Our Data teams will be/are staffed by developers who are comfortable with Node.js.	Our Data teams will be/are staffed by developers who are comfortable with legacy frameworks, such as C#.NET or Java.

**Table 2.** Skill considerations for mobility modernization



# Strategies for Migrating to Mobile

Any mobile modernization effort will consist of two discrete chunks of effort, the mobile apps and associated systems, and mobilizing data through cloud-based APIs and services. In this section, we'll discuss strategies for your mobile app migration, first via the types of apps available to you, and then architectural considerations for the apps themselves.

## Mobile Application Approaches

The first architectural decision you need to make for a modernization effort, whether you're creating a single mobile app or several, is the development approach you want to take. In particular, there are four major approaches you'll want to consider:

- **Native Mobile App:** An app purpose-built for mobile operating systems, using native languages, tools, and distributed as a native, installable package. Has full access to native device APIs and features.
- **Cross-platform Native Mobile App:** A native mobile app built from a single cross-platform codebase. Has full access to native device APIs and features, while leveraging cross-platform languages (i.e. JavaScript) and tools to support the creation of a single app.
- **Hybrid Mobile App:** A mobile app built with web technologies, hosted in a native app shell. Hybrid technologies (Cordova, PhoneGap, Ionic) embed full-screen browsers and use plugins to access native device APIs.
- **Mobile Web App:** A responsive web app tailored to mobile browsers (Safari, Chrome, Firefox). Can leverage native features of these browsers (Geolocation, Local Storage, etc.), but do not have access to native device APIs and features.
- **Progressive Web App (PWA):** PWAs are mobile web apps that leverage emerging mobile web standards to look, feel and behave more like native apps. They are an evolution of traditional hybrid and mobile web apps. Currently, PWAs have more support in the Android ecosystem, but coming improvements to Safari on iOS make this an option worth monitoring and evaluating in 2018.

### NativeScript

NativeScript is an open source framework from Progress for creating cross-platform native mobile applications with JavaScript.

This is **not** hybrid.

NativeScript produces truly native apps, complete with native performance, while enabling complete code reuse across iOS and Android. It does this while still integrating deeply with the technologies and concepts web developers already know: JavaScript/TypeScript, CSS, Angular, Vue, NPM and WebPack.

When used with Angular, a significant portion of an app's code can even be shared between the native mobile clients and the web.

Learn more at [nativescript.org](https://nativescript.org)



Each choice has pros and cons, and you should make your selection based on your organizational context (like skills), as well as the needs of your app or apps. And while you may wish to standardize on one approach for most or all of your apps, you should also consider allowing one-off cases where the needs of the app are uniquely suited to a non-standard approach.

The table below summarizes some key factors to consider when choosing an approach based on your organizational context and the needs of your app.

Factor	Approaches to consider
I need access to native device features and APIs not available in mobile browsers	Native, Cross-Platform Native, Hybrid
Our app needs to function as a desktop web app and a mobile web app from the same deployed location	Mobile Web, PWA
I need to be able to quickly deploy updates to my apps without relying on a 3rd party approval process	Mobile Web, PWA, Cross-Platform Native*, Hybrid
Our app is targeting a single mobile OS	Native, Cross-Platform Native, Hybrid
I need to standardize on an approach that matches the existing JavaScript skills of my web or backend teams	Cross-Platform Native, Hybrid, Mobile Web, PWA
I need 60fps performance across devices	Native, Cross-Platform Native
I need to be able to encrypt and decrypt local data on the device	Native, Cross-Platform Native, Hybrid
I need maximum access to device APIs and the best app performance	Native, Cross-Platform Native

**Table 3.** Mobile Application Factors

\* Only true if you chose a cross-platform framework that supports these types of updates. NativeScript, for example, supports this kind of hot patching.

## Mobile Application Architecture

Once you've determined your mobile app approach, you'll want to spend some time planning the architecture of the apps. Like traditional enterprise apps, this includes things like source control, branching procedures, coding standards, and other factors. While these are in your purview as an architect, they are out of scope for this article.

That said, mobile apps are architecturally unique in many ways, and introduce new considerations that fall within the application architecture heading, and which you'll want to consider. Table 4 below summarizes many of these, as well as options for native mobile and cross-platform or web projects:

### Note on hybrid

While hybrid has been a popular option for building cross-platform mobile apps, as an approach, it is rapidly fading. With the rise of PWAs and native cross-platform frameworks, there are simply better options for cross-platform mobile apps today. New projects should generally avoid hybrid.

Category	Native	Cross-Platform Native	Mobile Web, PWA
<b>Application patterns and practices</b>	MVC, MVP, MVVM. Largely driven by the patterns of vendor development stacks	MVVM, often referred to as MV*. Largely driven by best-practices in the broader web community and popular JavaScript frameworks, like Angular, Vue and React.	
<b>Device Application Runtime</b>	Native	Native w/ crossplatform interop (i.e. NativeScript)	Mobile Browser
<b>UI Frameworks</b>	UIKit (iOS), Material (Android)	Access to native UI frameworks via crossplatform interop.	Kendo UI, Bootstrap, Foundation, etc.
<b>Application Frameworks</b>	iOS, Android	Angular, Vue, React (via JavaScript)	Polymer, Ionic (PWA)
<b>Code-Sharing Approaches</b>	N/A; Not possible at the native code (UI, business logic, etc)	Business logic, networking and state management code can be shared between native mobile and web apps; some frameworks support UI code sharing	Business logic, networking, state management and UI components can be shared between web apps/PWAs

<b>Deployment &amp; Management</b>	Public or Internal Store Deployment	Public or Internal Store Deployment; Exception for JavaScript-based runtimes, which can live update nonnative code without full deployments	Server-based deployments
<b>Plugin &amp; SDK Management</b>	Cocoapods (iOS) and/or Gradle (Android)	npm, Cocoapods and/or Gradle	N/A; Native plugins and SDKs are not available with these approaches; Limited to browser APIs

**Table 4.** Mobile Application Architecture Considerations

## The Bottom Line

Given the considerations above, unless outweighed by other, contextual factors, we suggest a strategy that favors **a mix of cross-platform native apps and PWAs**, depending on the specific needs of each app. Not only does cross-platform native provide skills reuse that extends to web, cross-platform frameworks like NativeScript make it easy to share code across mobile app platforms (iOS and Android) and with your PWAs/web apps.

Once you've determined the approach for your app, and planned for your app architecture, you can turn your attention to the other side of the modernization coin: mobilizing the data your app needs.

# Strategies for Migrating to the Cloud

In the context of this playbook, we are using the term “migrating to the cloud” to encompass strategies for making existing on premises data and assets accessible to mobile apps, both in and beyond the office. This does **not**, however, mean that mobile modernization requires a wholesale migration of your legacy data sources into public or private clouds. Quite the opposite. In fact, it’s rare to find legacy systems that need to be completely modernized. Instead, we believe that a successful cloud migration is one that is piecemeal, value-based and Interface-driven.

- **Piecemeal:** Data and core functionality should be moved to the cloud in the smallest units possible, as opposed to moving entire systems, databases, or even services.
- **Value-based:** What moves to the cloud should be driven by the highest need and value, as opposed to a sequential list or backlog.
- **Interface-driven:** In many cases, “migration” means leaving core data in place, while using the cloud to expose targeted interfaces to that data via RESTful APIs, Microservices and the like.

In this section, we’ll discuss some cloud migration strategies, first through how you architect data services and systems, then through securing cloud data access.



## Piecemeal

Move the smallest units of functionality



## Value-based

Move based on need, not sequential order



## Interface-driven

Use RESTful APIs & Microservices to keep core data in place

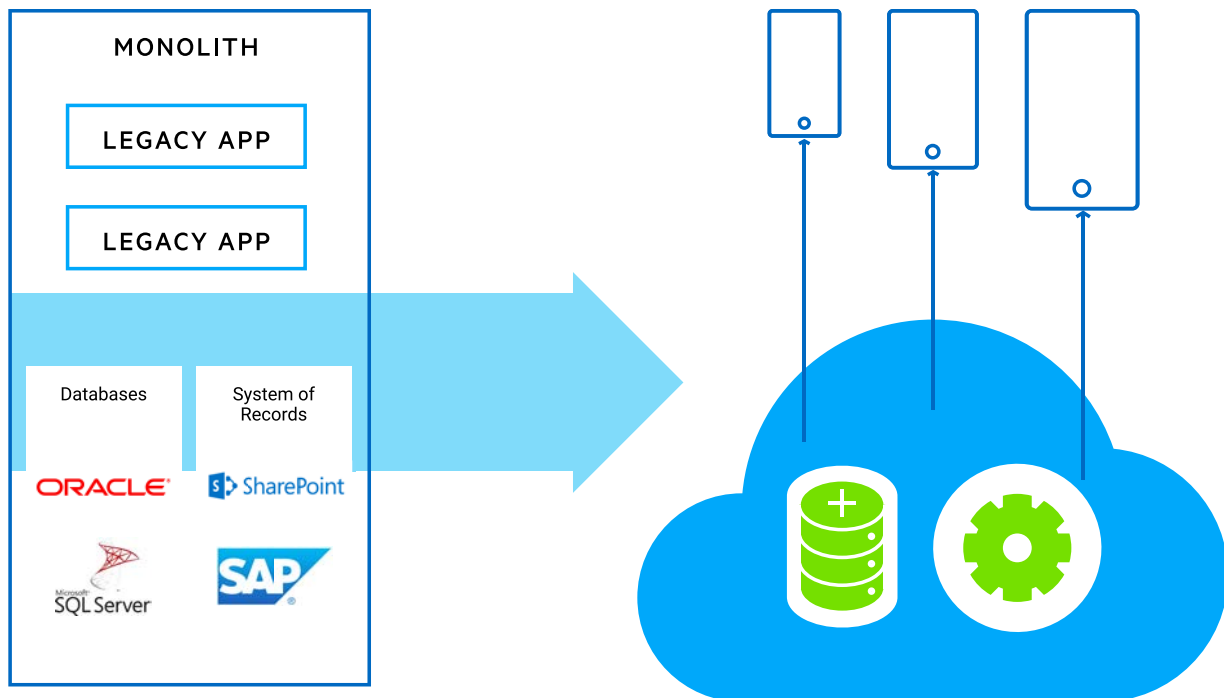
## Data Architecture

When planning the data architecture for mobile modernization, you should start with a clear picture of the role your monolith systems play in the effort, before determining how to migrate, expose and extend these into the cloud.

### Addressing Your “Monolith”

For mobile modernization, data architecture starts with identifying and putting a virtual boundary around your “monolith.” This is the collection of databases and systems of record that are essential to the apps you plan to mobilize. They could be custom-built systems or off-the-shelf systems for things like CRM, ERP, etc. No matter the source of the system, you should create an inventory of the systems that make up your monolith and the core data and required for your mobilization effort.

As we’ve said already, we **do not** recommend a modernization effort predicated on the wholesale migration of one or more monoliths to the cloud. Even if your intention is to eventually replace a monolithic system with a modern one, this should be planned out over time, using one or more of the approaches below.



**Fig 2.** Successful cloud migration strategies start with an inventory. It is possible to move parts of a monolith to the cloud and leave other parts unchanged while still gaining advantages the cloud has to offer.

## API Design

When planning the data architecture for mobile modernization, you should start with a clear picture of the role your monolith systems play in the effort, before determining how to migrate, expose and extend these into the cloud.

The first step is to provide access to monolithic data via a modularized, RESTful API. Not only does this approach provide an interface to your monolith that adheres to modern data access standards and practices (as opposed to native-driver-centric approaches of the past), but creating a RESTful API provides a blank slate upon which to define a clear migration path for enterprise data and core functionality.

One aspect of this approach that you'll need to plan for is how to facilitate the connection between the RESTful API and your monolithic systems. There are a few approaches to choose from:

- **BaaS + API Gateway:** An approach that wraps your monolithic sources and exposes data via mappings from source data to targeted, mobile-friendly data sources. Often facilitated by a BaaS or mBaaS product, such as Kinvey. Caching for increased data access performance.
- **Store-And-Forward:** An approach that insulates your monolith from direct access, while providing mobile access to core data. Requires sophisticated sync and conflict-resolution capabilities, especially in cases where mobile data is read-write.
- **Secure Data Pipeline:** An approach that provides secure access to on-premises data sources behind your firewall. One example is the Progress Hybrid Data Pipeline (HDP), which provides self-hostable data connectivity.

The goal, ultimately, is to create a reusable API layer that can power the mobile apps of today and tomorrow, while minimizing potential disruptions to existing legacy apps. With the right mix of these techniques, the traditional risks of migrating to the cloud can be significantly reduced.

### Kinvey

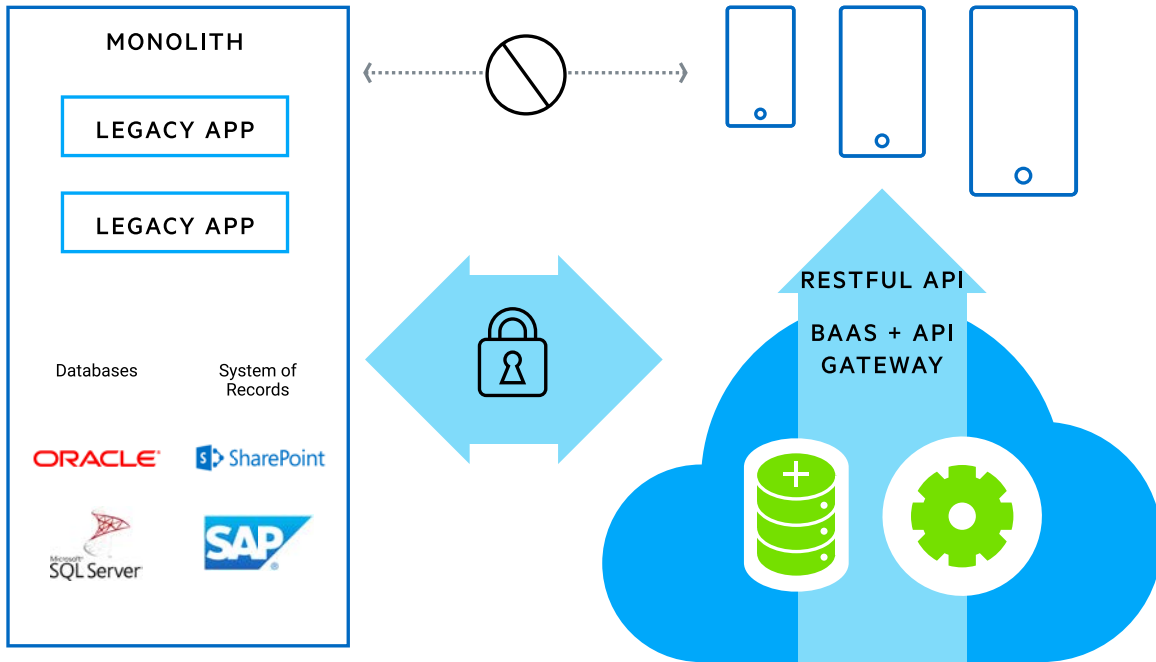
Kinvey is a complete serverless cloud platform from Progress that powers mission-critical apps. It provides everything required to deliver a high-performance, secure and compliant backend for every mobile project.

Named a Leader in the Forrester "Wave" for "Mobile Development Platforms," Kinvey specializes in helping development teams get to market over 75% faster.

**If you're trying to mobilize legacy systems or reuse existing auth providers, start with Kinvey.**

Kinvey can be run on any cloud, but unlike AWS, Azure or Google Cloud, which provide a "bag of parts" that development teams must assemble and maintain, Kinvey provides a seamless, integrated platform focused on compliance and productivity.

Learn more  
[progress.com/kinvey](https://progress.com/kinvey)



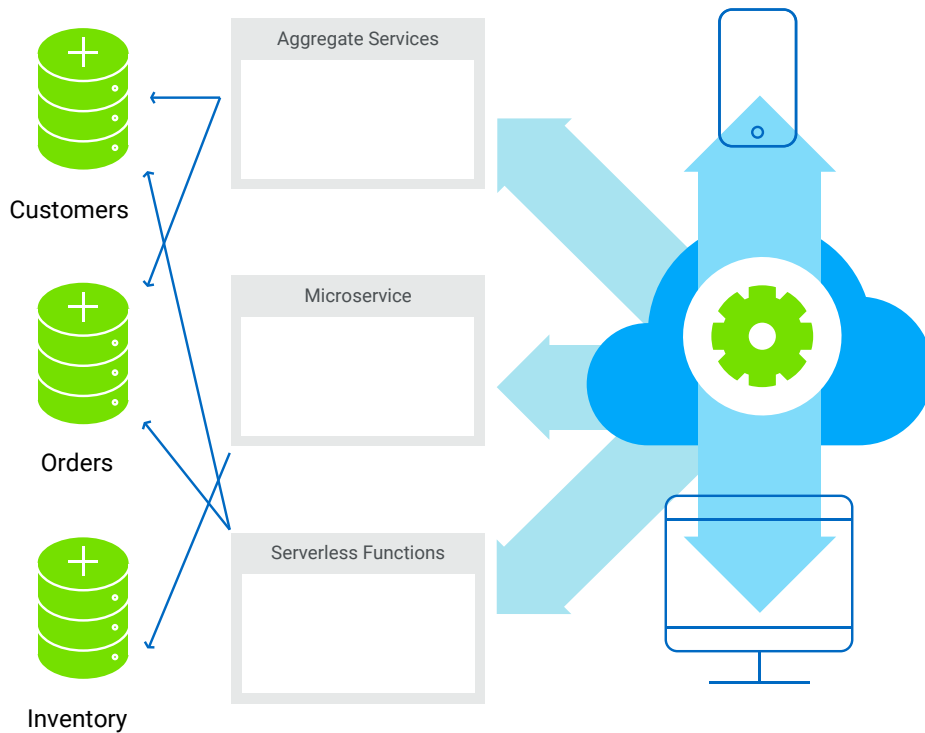
**Fig 3.** Using a secure data pipeline to expose existing data to the cloud makes it easy to build scalable, RESTful APIs for mobile clients without disrupting existing legacy apps, and avoids the pitfalls of connecting mobile clients directly to legacy

### Service Modularization & Granularity

Once you've identified one or more data access approaches for your API, you'll want to consider how to define the surface of your API. When creating a secure API for your effort, granularity is key. Because of the unique constraints of modern mobile applications (networks, device data plans, etc) it's rarely the case that your core data and services will map one-to-one to what's needed on mobile.

Often, you'll need to decompose a single service into multiple services to ensure that mobile devices are only pulling data needed for an app over a network. In many other cases, you may want to combine multiple services into a single API endpoint that provides a mobile-friendly payload. Very often, you'll use both of these approaches, together.





**Fig 4.** Use microservices, serverless functions and aggregate web services tactically. Rarely will a monolith become “all serverless” or “all microservices.” Migrating slices of functionality to these new paradigms is often the best balance between maximizing cloud performance and minimizing risk as you transition.

The table below highlights a few of the common service modularization approaches for you to consider.

Approach	What it is
Service Aggregation & Orchestration	Composing existing or related services into mobilefriendly APIs. Meant to minimize network calls and traffic in mobile apps by “normalizing” certain types of data. One example could be an Order API that pulls order and customer details from multiple monolith sources and combines these into a single payload.
Microservices deployed location	Decomposition of services into small, targeted units. May be shared, or specific to a single application or use. Driven by the needs of the app or apps.
Serverless	Event driven units of business logic or functionality as shared cloud services. Examples could be a unique email verification service or a CRM customer lookup services. Meant to be highly scalable and reduce operational costs compared to “always on” cloud services.

# Data & Mobile Applications

Finally, when planning your data architecture, you'll want to consider a few data-related design choices that are particularly unique to mobile.

- **Offline Storage:** Unlike the computers on our desktops, mobile devices can and often do go without a consistent network connection. A modern mobile application should have a plan for offline storage, especially for use cases where users will consistently find themselves without a steady connection.
- **Caching:** As discussed above, many BaaS systems help manage data access speed and mobile performance by caching certain types of data. In addition, mobile apps may choose to cache data on device for improved performance.
- **Queuing:** Similar to caching, many SDKs and systems help manage network access and device performance by queuing and batching data, especially when performing writes.

In all three of the above cases, you'll want pay close attention to how the systems, SDKs and tools you're using to help facilitate storage, caching and queuing handle sync schedules and conflict resolution. You'll want a solution that provides some common-sense defaults, while enabling your teams to control schedules and how conflict resolution decisions are made, at runtime. As these features are often complex to manage, we recommend using a 3rd party solution, as opposed to attempting to build all of these capabilities into your mobile apps and API, in house.

## Making Offline Data Easy

For most real-world mobile apps, the ability to work offline is critical. Yet so many apps fail to provide this capability because **offline is hard**.

It doesn't have to be with Kinvey from Progress.

Kinvey SDKs make it easy for developers to quickly add robust offline support to any mobile app, complete with automatic data sync to ensure offline changes are synchronized with the cloud.

This even works when using Kinvey to mobilize an existing database or legacy system. In fact, Kinvey's advanced caching techniques can make apps faster when interacting with slow, legacy backends.

Learn more at [progress.com/kinvey](https://progress.com/kinvey)

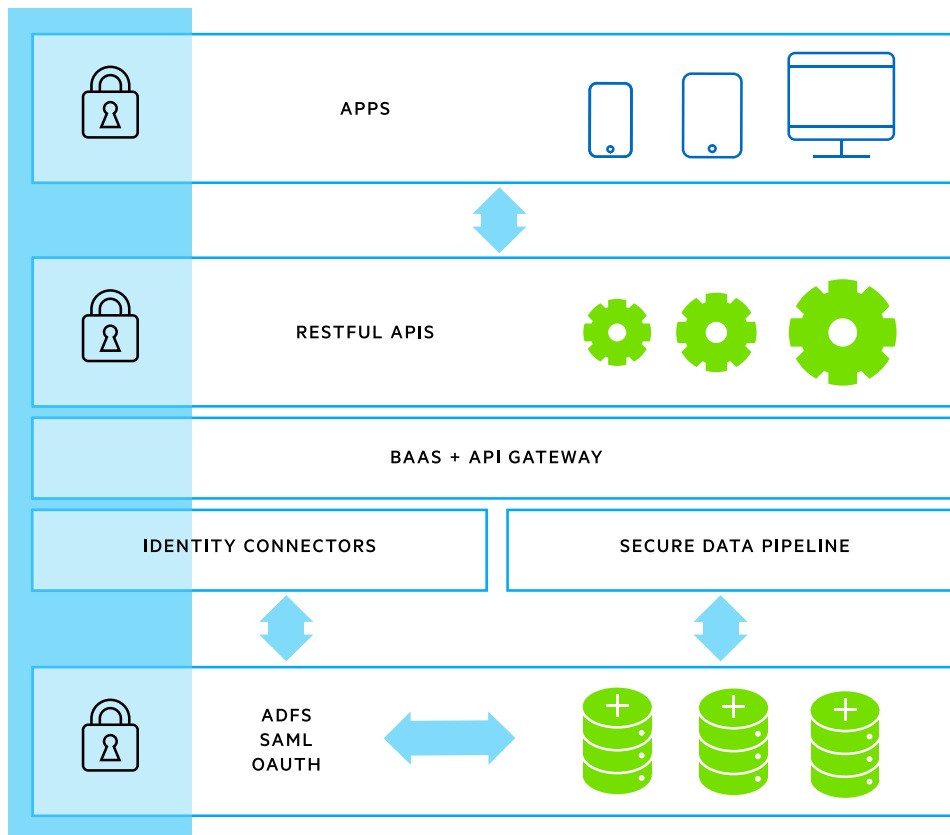
## Security

Now that we've talked about key considerations for designing your API, let's look at factors for making sure that APIs and apps stay secure, and your core data protected. In the context of security, there are three major areas you'll want to pay careful attention to:

- App and API Authentication and Authorization
- Data Encryption and Local Storage
- Compliance

### App and API Authentication and Authorization

When mobilizing and modernizing your monolith, your primary concern should be ensuring secure access on three fronts: your apps, your cloud-based API and, finally, to your internal data sources. Each of these points of authentication and authorization should work in-concert, be planned in advance, and be consistent across all apps and your entire API surface.



**Fig 5.** Securing mobile apps and RESTful API endpoints is an essential step during a cloud migration. Integrating with existing identity providers ensures consistent permissions across all apps, legacy and cloud, while allowing users to reuse existing credentials.

And while it's possible to implement your own security framework in all three of these areas, this is a complex task that is often better left to a trusted platform provider. Kinvey, for instance, provides a complete Identity Connect solution with client libraries for in-app authorization and authentication, an identity server for your cloud APIs, and no-code integration into your existing enterprise authentication sources, like ADFS, LDAP and SAML

## Data Encryption

Even if you've secured access to your apps and data sources, it's essential to use encryption as an added layer of protection. In the context of mobile modernization, you should encrypt data:

- On the pipe, in transit, using secure communication protocols like SSL
- At rest, especially when data is to be cached or stored locally apart from its source. This could be either data stored on a mobile device, or in a cloud cache.

As with authentication, handling encryption of your data, especially on device, is complex. Consider client libraries that assist in this encryption, and which can communicate securely with your token servers. Kinvey, for example, provides client libraries for all mobile app types that facilitate encryption and decryption of your sensitive data.

## Compliance

Finally, if your organization operates in an industry where compliance is always a built-in consideration, you're likely already thinking about this topic, when it comes to data and making your core business cloud-accessible and mobile. And while secure authentication and encryption are always top considerations on a compliance checklist, there's much more to consider, especially with compliance requirements like PCI, PII and HIPAA. If you're evaluating cloud platforms to accelerate your mobilization efforts, pay careful attention to what compliance certifications and guarantees each candidate provides. Progress Kinvey, for instance, considers compliance an essential core feature of its platform, and provides a set of purpose-built HIPAA-compliant services into its core products.

### Mission-Critical & Compliant

Not all apps are created equal. Some are just for fun and disposable. Some are mission-critical, working with sensitive customer data.

**Kinvey, from Progress, is the backend for your mission-critical apps.**

Kinvey is HIPAA compliant and powers the Progress HealthCloud, a tailored cloud platform for healthcare organizations, pre-integrated with EHRs and other healthcare data sources.

Connections to existing data and authentication systems happen over secure pipes, and the Kinvey SDKs make it easy to encrypt and decrypt data. When it's time to ship, Kinvey provides BAAs, SLAs, operational intelligence dashboards and industry leading support.

Learn more at [progress.com/kinvey](https://progress.com/kinvey)

## Choosing the Right Cloud

When considering a cloud provider, it's important to consider not only your needs and context, but also the strengths and specializations of the provider. While there are several cloud vendors on the market of various sizes, not every cloud is the same. Some are a loose collection of services that you assemble (like AWS, Azure or Google Cloud Platform), while others are pre-integrated or specialize around a set of use cases, like hosting or mBaaS providers.

Within specialized providers, you'll also find that some are stronger with greenfield (typically B2C) scenarios, while others are optimized for working with existing clouds and monoliths. For example, while both Kinvey, from Progress, and Firebase, from Google, are focused on adding value on top of "raw" clouds to accelerate mobile app development, Firebase tends to be better suited for consumer apps, while Kinvey provides building blocks and functionality tuned for enterprise modernization efforts.

The table below summarizes some key factors to consider when choosing a cloud based on the needs of your app and the strengths of the provider.

Cloud	Pros	Cons
"Raw" Cloud (AWS, Azure, Google Cloud)	<ul style="list-style-type: none"> <li>• Maximum access to cloud capabilities</li> <li>• Complete control over cloud resources</li> </ul>	<ul style="list-style-type: none"> <li>• Nothing is pre-integrated; some assembly required</li> <li>• Added maintenance overhead</li> <li>• Steep learning curve</li> <li>• Unpredictable pricing</li> </ul>
Consumer BaaS (Firebase)	<ul style="list-style-type: none"> <li>• Easy to get started</li> <li>• Abstracts "raw" cloud complexities</li> <li>• Great for "greenfield" apps with no need to access existing data/auth providers</li> <li>• Useful consumer-oriented capabilities (ads, public app stores)</li> </ul>	<ul style="list-style-type: none"> <li>• No support for reusing existing data/auth providers</li> <li>• No cloud portability</li> <li>• Limited compliance and SLA options</li> </ul>
Enterprise BaaS (Kinvey)	<ul style="list-style-type: none"> <li>• Easy to get started</li> <li>• Abstracts "raw" cloud complexities</li> <li>• Robust support for connecting to existing data</li> </ul>	<ul style="list-style-type: none"> <li>• Less flexible than "raw" cloud services</li> <li>• More expensive than Consumer BaaS options</li> </ul>

**Table 6.** Weighing the pros and cons of three different options

# Future-Proofing Your Modernization Efforts

After you've taken the time to fully-consider your application architecture, you'll be well on your way to a successful modernization effort. However, if you want to ensure that your organization is prepared not only for the needs of today, but can anticipate and better respond to an evolving landscape in the future, you'll want to spend some time future-proofing your current effort.

Future-proofing, in this context, does not mean that you can successfully predict every single change in technology and your industry. It does, however, mean that you should anticipate that change will occur, and that you future-proof through:

- Making performance a feature
- Baking operational intelligence into your apps and API
- Design APIs and Services that can be leveraged beyond mobile

In the final section of this playbook, we'll look at each of these, in turn.

## Performance Considerations

The phrase "performance is a feature" is meant to convey that your apps should be designed and built not just to work, but work well. They should be tuned to be as fast and lightweight as possible, to use only the data needed and to provide the best performance experience for end users. And while performance is a complex, multi-faceted topic, there are a number of high-value areas that you can focus on to ensure built-in performance for your effort:

Factor	What it is	Why it matters
<b>Payload trimming</b>	When building RESTful services, favor services that return or require only the base minimum of data needed.	Smaller payloads result in faster network operations
<b>Data &amp; Asset Compression</b>	Ensure that all data, images and other assets are compressed (for example, with gzip) on the server	Smaller assets and data payloads result in faster network operations, especially on lower bandwidth mobile connections

<b>Network latency monitoring &amp; management</b>	Be aware of and instrument how your legacy systems, cloud services and mobile apps perform in the wild. Favor offline and caching approaches for mobile apps where latency is a real consideration	High latency conditions can often render apps non-functional, and your employees or customers, frustrated
<b>Push &amp; user notifications</b>	Use mobile push notifications, SMS and email to provide critical updates or trigger an interaction	An effective notification pattern keeps users out of your apps unless they have a reason to be there, resulting in better-utilization of network requests

Table 7.

## Monitoring & Management Considerations

The second way to future-proof your modernization effort is to bake operational intelligence into your apps and API, so that you can monitor actual usage, identify trouble-spots or issues, and spot trends and opportunities that you might otherwise miss. Monitoring falls into a few categories:

Factor	What it is	Why it matters
<b>Analytics, monitoring and measurement</b>	Instrumenting the various features of your apps and API	Provides insight on actual app and feature usage
<b>Logging and auditing</b>	Crash monitoring and logging; Exception tracking; Network monitoring and logging	Can help catch bugs before they become widespread; Provides insight on device usage, network conditions in the wild
<b>End-User Feedback</b>	Whether in-app or otherwise, providing customers and end-users an opportunity to self-report issues with an app or experience	Sometimes, your own tools will miss bugs, or missing features your users identify as bugs. Feedback adds a human factor to app monitoring.

Table 8.



## Design with the Future in Mind

Finally, when it comes to future-proofing your mobility efforts, you should be aware of how the definition of mobile is changing, and how other emerging trends may affect your business in the future. In every case on this list, or even your own internal list, the key is to build flexibility into your API from day one, by creating an API that is:

- **Modular:** Decomposed into essential units of work, using a microservice or serverless approach
- **Composable:** Easy to create new services from existing services with little friction

That applies to anything in this list below, or anywhere your organization might innovate in the future.

Factor	What it is	Why it matters
<b>Chatbots</b>	Voice and Messaging as the UI of the future; Replaces traditional form-based patterns and interactions with conversational flows	Model services around objectives, not interactions. For example, consider authentication: Are your existing auth services compatible with voice or chat patterns?
<b>Progressive Web Apps (PWAs)</b>	"Installable" web apps that leverage the web, but behave like mobile apps	Embrace web technologies and cross-platform solutions, even for your native mobile apps
<b>Augmented and Virtual Reality</b>	Interaction patterns that overlay computing on the physical world (AR), or which introduce wholly-virtual environments (VR)	Look for user interactions and processes that would benefit from contextual overlays. Prepare for heavier use of computer vision as input to your apps.
<b>IoT / Industrial IoT</b>	Embedded device connectivity, from the field to the smart home, to the shop floor and beyond; An explosion of network connected and aware devices accessing, creating and acting upon your core data	Don't assume that the mobile device is the most network- and process-constrained device you'll ever need to support; Decompose APIs as much as possible to support IoT use-cases

### Future Proof with Progress

App development is always evolving, and the challenges extend far beyond mobile. In addition to NativeScript and Kinvey, Progress offers a complete app development platform that covers:

- **Web Apps:** Rapidly build modern web apps, including PWAs, using the industry leading Kendo UI widgets
- **Desktop Apps:** Make any app look better and do more with the legendary Telerik UI tools
- **Chatbots:** NativeChat, from Progress, is the easiest way to train a bot and add it to your web or mobile app.
- **IIoT:** Progress DataRPM provides an innovative solution for detecting and predicting industrial equipment failures.
- **AR/VR:** Progress Labs is hard at work creating tools to make AR/VR easy to add to any app.

Learn More at [progress.com](http://progress.com)

# Next Steps

No matter where technology goes, or how your organization evolves, creating flexible, modular and composable services increases the likelihood that you'll be able to respond quickly to change.

The world of technology continues to advance and increase in complexity, presenting you with both opportunities and challenges. As an architect, you are uniquely suited to navigate your enterprise through these challenges and opportunities both by leveraging your existing skills and patterns, and educating yourself on the unique challenges presented by mobility and the cloud.

Hopefully, this playbook has provided an overview of those challenges, and equipped you to continue the modernization conversation within your enterprise. If you're undertaking a modernization effort, Progress can help. [Click here to contact us](#) and request additional guidance from one of our mobility experts.

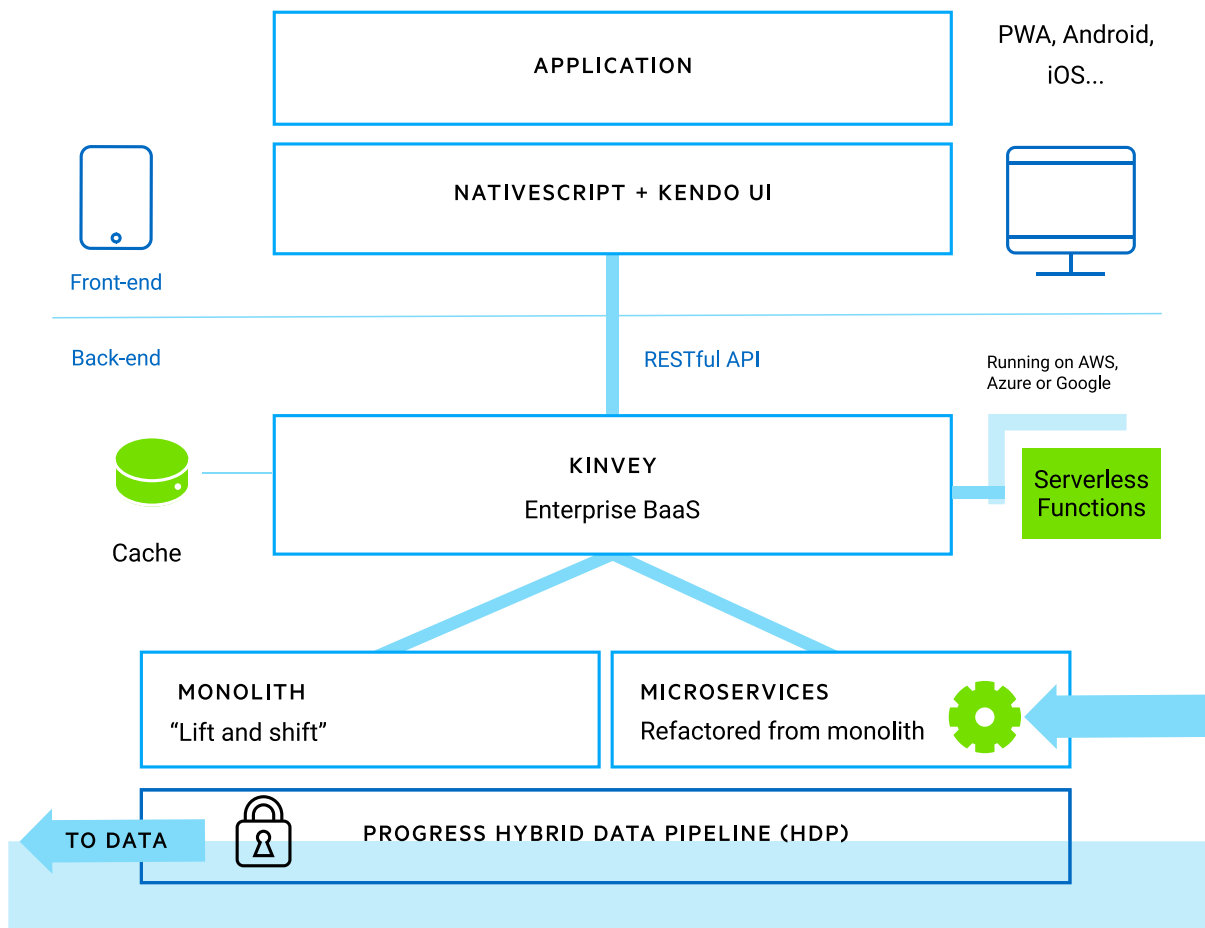
Also available when you contact us:

- **Migration architecture diagrams** in easy to copy PowerPoint format. Add them to your next internal presentation.
- **Migration “checklist”** to help guide your mobile modernization effort

# Progress Reference Architecture

While the guidance in this whitepaper is universal, Progress provides a complete app development platform that makes migrating to the cloud and mobile easy. This reference architecture shows how solutions from Progress can be used together on your journey to modernization.

## Going cloud-native with Progress



Going “cloud-native” with Progress Kinvey, Hybrid Data Pipeline, NativeScript and Kendo UI fully embraces the ideals of piecemeal migrations that minimize risk and maximize opportunity that the cloud has to offer. With this approach:

- NativeScript provides a “write once, run anywhere” application layer for creating native mobile apps for iOS and Android, while sharing code with web apps and PWAs created with Kendo UI.
- Kinvey provides a fully-integrated, serverless backend that runs on any cloud and offers out-of-the-box integrations with existing auth providers and systems of record
- Hybrid Data Pipeline helps streamline secure connections between existing data and the cloud



To learn more about mobility solutions available from Progress, including technologies mentioned in this whitepaper, such as NativeScript and Kinvey, please visit our website: [www.progress.com/solutions/mobility](http://www.progress.com/solutions/mobility)

If you have questions and would like to talk to a Progress mobility expert, please [Contact Us](#)

## About Progress

Progress (NASDAQ: PRGS) offers the leading platform for developing and deploying strategic business applications. We enable customers and partners to deliver modern, high-impact digital experiences with a fraction of the effort, time and cost. Progress offers powerful tools for easily building adaptive user experiences across any type of device or touchpoint, award-winning machine learning that enables cognitive capabilities to be a part of any application, the flexibility of a serverless cloud to deploy modern apps, business rules, web content management, plus leading data connectivity technology. Over 1,700 independent software vendors, 100,000 enterprise customers, and two million developers rely on Progress to power their applications. Learn about Progress at [www.progress.com](http://www.progress.com) or +1-800-477-6473.

© 2018 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.  
Rev 2018/02 | RITM0018846