Features & Technology Overview

Last updated June 15, 2016

Executive Summary
PlayFab Game Services
Cross-Platform Player Accounts
Data Storage
Title data
Catalog data
Assets
User data
Scope
Permissions
Client access rights
Player group data
Statistics and Leaderboards
Leaderboards
Resettable leaderboards
Matchmaking
Segmentation
In-game commerce
Virtual currencies
Catalog
Drop tables
Stores
Inventory
Trading
Real-money transactions
Server-side game logic
CloudScript
Custom game server hosting
Photon
Marketing
Push notifications
News articles
Coupon codes
Cross-game promotions.
Targeted stores
Social
  Friends list
  Friends leaderboards
Chat
PlayStream
  Event Pipeline
  Event Debugger
Rules Engine
Real-time Player Segmentation
Webhooks
Event History
Event Replay
S3 Archive
PlayFab Game Manager
  Permissions
  Dashboards and Reporting
  Audit log
  Customer support
Developer Tools
  API
  Multiplatform SDKs
Add-on Marketplace
PlayFab Technology
  PlayFab Architecture
    API technology stack
    PlayStream technology stack
    CloudScript technology stack
    Custom game server technology stack
Scalability
Availability
  Service monitoring
  Incident management
  Limits
Security
  Data backup and restore
Development Process
  Steps for developing and deploying changes
Load Testing
Adventure Capitalist Launch
  API Performance for AdCap
Future Roadmap
Executive Summary

This white paper provides an overview of all PlayFab features, the technology used to build and operate PlayFab, and the processes followed to create, deploy, monitor, and maintain the entire platform. It’s meant to be a living document, updated frequently to keep up with the rollout of new features.

This white paper is not meant to be a technical reference, or how-to guide, however. Some features listed here as single features live across multiple APIs and are exposed in multiple places in the game manager web application. For example, “User Data Storage” spans 18 separate APIs between the client and server APIs.

For more information on any of the features listed in here, see the online documentation site, at https://api.playfab.com/ or contact the Developer Success team at “devrel@playfab.com”.

PlayFab Game Services

PlayFab offers the following game service features. These features are accessed directly via the PlayFab Web API, or through a platform-specific SDK. Many of these features are also exposed via the PlayFab Game Manager web application for game testing, configuration, or operation.

Cross-Platform Player Accounts

Player accounts are at the heart of PlayFab. Every user gets a player account, which are authenticated by and linked to the accounts in multiple third party platforms, such as:

- Steam
- iOS device identifier
- Apple Game Center
- Android device identifier
- Google Play
- PlayStation Network
- Xbox Live
- Facebook
- Amazon
- Kongregate
- Custom authentication providers (such as an existing internal account system)

By linking multiple authentication providers to a single player account, PlayFab supports cross-platform game play.
Data Storage

PlayFab provides an easy way to store custom player and game data. Data is stored as key/value pairs, where values can be plain text, JSON, or binary blobs.

Title data

Title data which can be accessed by all players and game clients. This data is typically used for game configuration information, such as game difficulty settings or level descriptions.

Catalog data

Every item in the catalog (see section on in-game commerce) can have custom properties associated with it, such as rate-of-fire for a weapon, or mana points for a collectible card.

Assets

PlayFab provides support for uploading asset files, and then delivering them to game clients via an integrated CDN.

User data

Data stored per player account. Player information is broken down as follows:

Scope

- **Publisher-level** -- data for a given user account shared across all titles from that publisher, such as cross-game achievements.
- **Game-level** -- player account data for a specific game title, such as save state, controller settings, or avatar image.
- **Character-level** -- each player can have multiple characters (or vehicles) in a single game, each with their own data for things such as tank configuration, or health remaining.
- **Inventory-level** -- data for individual items from a player’s virtual goods inventory.

Permissions

- **Private** -- can only be accessed by the player.
- **Public** -- can be accessed by other players (e.g. description of a base)

Client access rights

- **Read/write** -- the game client can read and write the data.
- **Read-only** -- the game client can read the data, but only a game server can write it.
- **Internal** -- the data is invisible to the client; only the server can read or write it.
Player group data

Data may be shared among a specific group of players, such as a guild, or players in a lobby. Examples include guild name, or group message of the day. Players can be added and removed from a shared group; all members in the group can read or write shared group data.

Statistics and Leaderboards

Statistics are a special subcase of player and character data, used to represent numerical properties. They have string names and integer values, and they are set according to one of the following aggregation methods:

- Last: overwrites the statistic with the specified value
- Min: overwrites the statistic if the specified value is lower than the current value
- Max: overwrites the statistic if the specified value is higher than the current value
- Sum: adds the specified value to the current value

Besides tracking numeric properties, statistics can be used for the following purposes.

Leaderboards

Any statistic can be used to define a leaderboard comprised of a list of players, ranked by their value for the statistic. There are API options for retrieving the top ranked players overall, the players surrounding a given player on the leaderboard, or a player’s ranked list of friends (see Social).

Resettable leaderboards

Statistics can be configured to reset on an hourly, daily, weekly or monthly schedule. When a statistic resets, all associated leaderboards and matchmaking rankings reset as well. Archives of leaderboards and all players’ values for previous versions of the statistic are retrievable after the reset. This can be useful for events or tournaments.

Matchmaking

Stats can be used as part of the multiplayer matchmaking logic (e.g. find a match based on player’s level or weapon rating).

Segmentation

Statistics can be used to define player segmentation rules (e.g. players with XP > 5 and < 10). More information in the PlayStream section.
In-game commerce

PlayFab provides an in-game commerce system for managing and selling in-game entitlements with real or virtual currency. These can range from virtual items (e.g., a sword, a new tank) to content unlocks (e.g. unlock level 5, unlock an additional character slot).

The following features work together to form a complete in-game economy, with PlayFab handling the logic to securely verify real-money purchases and grant the associated entitlements, to process transactions for earning and spending virtual currency, to manage the lifecycle of each virtual good, as it is granted, stacked, consumed or expired, and more.

Virtual currencies

Each game can define multiple virtual currencies, with optional initial balances, recharge rates, and maximums. Each player and character can have separate balances for each virtual currency, and virtual currency can be exchanged between characters and players.

Catalog

The game catalog defines the master list of all items in the game, with optional usage counts, expiration times, custom properties, stacking behavior, price in one or more virtual currencies, and price in one or more real-world currencies. Bundles allow multiple items or currencies to be purchased together. Containers are like bundles, but must be opened before their contents are granted. Containers can be locked, in which case the player must have a matching key to open the container.

Drop tables

Drop tables are weighted lists of items that can be used with bundles or containers to randomly grant items according to the desired probabilities. Examples include card-packs, treasure chests, or lucky draws.

Stores

A store is a subset of a catalog, with optional price overrides. Stores allow items to be discontinued from sale, yet still exist in the catalog. They also allow certain segments of players to be targeted with special prices (see Targeted Stores).

Inventory

Each player and character has an inventory of items that is maintained by PlayFab as items are purchased, granted, consumed, or expire. Items in the inventory can have custom per-instance properties.
Trading

Bundles of items can be traded between players. Trades are executed using an escrow and are fully transactional.

Real-money transactions

PlayFab supports a number of payment mechanisms for real-money transactions, including platform-specific systems such as Steam, PlayStation, and Xbox, Facebook, PayPal and Amazon. PlayFab also supports server-to-server receipt validation with protection from replay attacks for mobile platforms such as Google and Apple.

The specifics of each payment provider integration are abstracted behind consistent purchasing and receipt validation APIs.

Server-side game logic

PlayFab provides several mechanisms for securely hosting custom game logic, without having to manage server infrastructure.

CloudScript

Server-hosted JavaScript which define custom functions that execute in response to requests from the game client or to filtered PlayStream events. Cloud script code executes within the security context of the current player and can call the privileged server API, which cannot ordinarily be called from the client. It can also make web requests to arbitrary HTTP endpoints, enabling integration with other services or existing backend systems.

The results of each Cloud Script function execution, along with log statements and diagnostic information are returned to the game client and/or published as PlayStream events, making it easy to debug and troubleshoot.

CloudScript provides process-level isolation between titles and utilizes the V8 JavaScript engine for high performance. Code revisions can be uploaded to the Game Manager or pulled from a Git repository, using the GitHub Marketplace Add-on.

CloudScript functions can be called in several ways:

- From a game client via the client API.
- From a game server, via the server API.
- As a triggered action in response to a particular PlayStream event.
- Manually, from the Game Manager, for a particular player.
Custom game server hosting

PlayFab can host custom game servers built with any language and engine. After a package containing the server executable and dependencies is uploaded, PlayFab automatically deploys and scales game servers based on load to maintain sufficient capacity for hosting new game matches. Game servers can optionally be hosted in many different regions around the world, providing for low latency connections between game clients and servers.

Built-in matchmaking supports assigning players to matches based on parameters such as min/max players, region, game mode, player statistics, and friend list. Developers may also provide custom matchmaking logic.

Games servers can write local files, including logs, replays, or crash dumps, which are automatically archived and indexed for later analysis or replay. Game servers authenticate connecting players using a ticket system, and can call the privileged game server API.

Photon

PlayFab’s partnership with Exit Games makes it especially easy for games to provision multiplayer game rooms using Photon Cloud. Rooms can automatically call CloudScript for events such as EnterRoom or ExitRoom, and user authentication is taken care of automatically.

Marketing

PlayFab has several features to help games market to their existing players.

Push notifications

Games can send custom mobile push notifications directly to a specific player using the server API. Push notifications can also be triggered by player segment changes or PlayStream events. The content of the push notification message can be customized with properties from the player’s profile using a template.

Customer service reps may also manually send a push notification to a player directly from within Game Manager.

News articles

Developers can publish articles using the Game Manager to be displayed in the game. Examples include launcher message-of-the-day, interstitials, and in-game notices.
Coupon codes
PlayFab can generate one-time coupon codes which may be redeemed in-game for any item in the catalog.

Cross-game promotions.
Publishers can cross-promote titles on PlayFab by enabling players in one game to unlock or be granted items in other games.

Targeted stores
The contents of a store can be replaced with those of a different store based on a player’s segmentation. This allows a game’s product manager to offer different stores to different players without having to write any code, including both A/B test scenarios.

Social
PlayFab has several social features to help promote higher engagement and retention through player-to-player interaction.

Friends list
PlayFab maintains a friends list for each player, and can automatically add players to that list by matching against existing Facebook or Steam friends. Games can also add or remove players to the list directly. The friends list can also be used by leaderboards and matchmaking.

Friends leaderboards
Leaderboards can be filtered to include only a player’s friends who have a value for the leaderboard statistic.

Chat
PlayFab supports player chat rooms via partnership with Exit Games.

PlayStream
PlayStream is an event processing system that unifies the entire data flow from a game into a single event stream. It collects built-in events generated by PlayFab game services, as well as custom events generated by a title’s game client, server, and existing backend systems. It processes the stream of events through a flexible rules and player segmentation engine in real-time, and routes them to wherever they need to go, including third party services.
Event Pipeline

The event pipeline reliably captures and stores events generated by the API, client and server, and it automatically scales to handle growing event volumes. Tracking information about each event is updated in real-time at every stage of the pipeline.

Event Debugger

PlayStream events can be viewed in near real-time as they flow through the pipeline, using the event debugger in Game Manager. This is useful for debugging during development and for customer support after launch. Events can be viewed for a particular player, or sampled across the entire title.

Rules Engine

Rules for triggering actions based on matching events are defined using the visual rule editor in the Game Manager. For example, an event indicating that a player completed a tutorial might trigger a rule that grants the player an inventory item, sends them a push notification, or even executes a custom Cloud Script function. These rules are matched to events in the pipeline in real-time, making them suitable for implementing game logic which can be revised without requiring client updates.
Real-time Player Segmentation

Player segments are defined based on profile properties such as statistics, virtual currencies, source, or login behavior using the visual segment builder in the Game Manager.

Segments can be used to change the game’s behavior based on the player’s segment. For example, using the Targeted Store Feature, a game may target different stores to different player segments. Segments can also be shared with third-party add-ons.

Actions can also be triggered automatically as players enter or exit segments, in real-time. For example, the “social maven” segment could be configured to automatically grant a gift to players as soon as they invite more than 10 friends via Facebook.

Players can also be placed into segments randomly, to enable A/B testing. Product managers can define multiple test buckets, with different weightings for each, into which players are automatically slotted. These segments can then be used to experiment with different game behavior, with the resultant KPIs broken out by test segment.

Webhooks

PlayStream events can be posted to external web services via HTTP webhooks. Webhooks make it easy to integrate with a custom analytics provider or to tie-in to a developer’s existing backend services.

Event History

A searchable event history supports filtering and viewing historical events based on properties such as player, event type, time and source. The results include delivery status for webhooks and other destinations.

Event Replay

The historical archive of events can be “replayed” against a particular webhook or 3rd party Add-on, making it possible to try out or sign up for new vendor services, and get the full benefit of historical data.

S3 Archive

A complete or filtered set of PlayStream events for a title can be written directly to an AWS S3 bucket, formatted for direct import into RedShift or for processing by Elastic Map Reduce jobs,
among other purposes. The S3 bucket can be in any AWS account, as long as it has the appropriate write permissions.

**PlayFab Game Manager**

PlayFab provides a unified web interface that gives an operations team a single destination for viewing and managing a live game. Most of the game services listed above, including PlayStream, can be managed and used via the Game Manager.

Specific features unique to the Game Manager include:

**Permissions**

PlayFab provides a robust permission model so that different users can be given permission to access different features. For example, customer service reps can be given permission to edit player properties but not change item prices, and product managers can be given permission to create or edit items, but not stop or start game servers.

**Dashboards and Reporting**

A set of basic dashboards include up-to-the-minute graphs of game KPIs, such as:

- Logins
- Installs
- Purchases
- Virtual currency transactions
- PlayStream events
- API requests
- API errors

Daily reports provide accurate tallies for:

- DAU
- MAU
- ARPU
- Retention
- Top grossing items
- Top spending players

Additional dashboards and reports will be added over time.
Audit log

A list of all configuration or data changes made via the Game Manager, along with who made them, makes it easy to track down issues or monitor support changes.

Customer support

Support representatives can investigate issues and provide service recovery by granting or revoking virtual items or virtual currency, temporarily or permanently banning abusive players, looking up login history, editing player properties, reviewing purchase history, and more.

Developer Tools

PlayFab provide a rich set of APIs and SDKs to help developers easily integrate PlayFab’s services into their game.

API

All PlayFab services are exposed via a JSON-based web API for fully cross-platform support. All API documentation is auto-generated from the code ensuring it is always up-to-date with any changes.

Multiplatform SDKs

PlayFab provides SDK’s for the following devices or platforms:

- ActionScript
- Android Studio
- C#
- Cocos 2D
- Java
- Javascript
- Lumberyard
- NodeJS
- Objective C
- Unity3D
- Unreal Blueprint
- Unreal C++
- Windows (C++)
- Xamarin
All of the SDK’s are auto-generated directly from the API code, making it easy to add new SDK’s as needed, and ensuring that the SDK stays consistent with any changes to the API. The API generator was open-sourced via GitHub so developers can build their own SDK’s if needed.

### Add-on Marketplace

PlayFab provides a marketplace of pre-integrated third party services, many of which can be activated without any additional SDK needed on the client.

Categories of add-ons include advertising, attribution tracking, community, advanced analytics, payment providers, and more.

Marketplace Add-ons are able to send and receive PlayStream events, and they can expose actions which may be triggered via PlayStream Actions. Partners are able to access player profile data, providing a single consistent view of the player across the entire system of services, and can also call into the PlayFab API to interact directly with the game.

### PlayFab Technology

Operating online games at scale is a challenge. Load is unpredictable and can ramp up very quickly. Data access patterns are very demanding and write heavy. Infrastructure can fail without warning.

PlayFab’s approach to providing a highly scalable and available service is to build with components that are 100% elastic with respect to compute and storage, and that offer multi-datacenter redundancy.
PlayFab Architecture

Architecture Overview

API technology stack
- DNS (Route53) per-title traffic routing
- HTTPS load balancer (ELB) endpoints supporting TLS1.2
- API server auto-scaling group of EC2 instances running C# / ASP.Net MVC5
- Player data stored in DynamoDB (auto-scaling read/write throughput) and S3

PlayStream technology stack
- Incoming events written to Kinesis data stream (sharded by player ID)
- Auto-scaling groups of EC2 instances read events from stream and perform real-time segmentation, action triggering and dispatch
- Events dispatched to secondary Kinesis streams for further processing (webhooks, CloudScript triggers, custom S3 bucket delivery, partner add-on integrations, etc.)
- Events and segmentation results stored in S3 (archive / replay), Redshift (reporting), Elasticsearch (search, trends), DataDog (real-time metrics), and RDS (rolled-up player profiles)

CloudScript technology stack
- All CloudScript requests forwarded to a separate pool of auto-scaling EC2 instances with extremely limited role permissions - no direct data access
• Scripts execute in the V8 JavaScript engine
• Pre-authenticated server API, log collector, and HTTP client exposed as script objects using ClearScript engine host
• Process isolation between scripts from different titles

Custom game server technology stack

• Title game server executable build with any language / engine technology
• PlayFab management service launches server processes, monitors health, and collects logs, output files and crash dumps
• Windows Server 2012 R2
• Autoscaling service ensures that there is sufficient server capacity in all selected regions

Scalability

Every component of the core architecture is capable of rapidly adjusting compute and storage resource capacity based on usage level. This meets the challenges of variable and unpredictable load, while operating efficiently and cost effectively.

The primary data store for player data is DynamoDB, which supports quickly modifiable levels of read and write throughput capacity. A resource monitoring service adjusts these levels every minute, based on usage.
Services run on groups of auto-scaling EC2 instances in three Availability Zones. Enough spare processing capacity is maintained to absorb an AZ outage with no downtime.

- Typical daily peak as of June 16, 2016: 1,200 RPS, 1.9M daily active users
- Load test peak: 22,000 RPS, 1M concurrent users

Availability

There are no “special snowflake” servers. A machine terminating is never a cause for alarm. All server roles, including core web services, matchmaking, event stream processing and report generation have instances running in multiple AZs. Storage services are all replicated across multiple data centers as well.

- All data is stored in managed AWS services with 2 or 3 redundant copies in separate Availability Zones for durability and availability
  - S3: 3 AZs
  - DynamoDB: 3 AZs
  - RDS: 2 AZs
- All API and event processing runs on machines distributed across 3 AZs.
- Single-purpose servers (e.g. matchmaking) run in primary/replica configuration in 2 AZs, with Route53 application health check triggered failover (~30s latency)
- Uptime (monitored by pingdom): 99.99% (27 min downtime) last 12 months, 100% (0 min downtime) YTD

Service monitoring

Thousands of application and machine level metrics related to the operation of the service are collected continuously and written to Datadog, a service that provides data aggregation, analysis, dashboards and alerting. These metrics provide insight into the performance of the APIs and features and how customers are using them.

The same metrics that appear on dashboards in the PlayFab offices also feed into a set of automated monitors and alerts that can trigger alarms in the face of abnormal conditions. Some examples of metrics and alerts include:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Dimensions</th>
<th>Alerts</th>
</tr>
</thead>
<tbody>
<tr>
<td>API execution</td>
<td>• API</td>
<td>• High response latency</td>
</tr>
<tr>
<td></td>
<td>• Calling title</td>
<td>• High error response rate</td>
</tr>
<tr>
<td></td>
<td>• HTTP status</td>
<td>• Sudden request rate change</td>
</tr>
<tr>
<td></td>
<td>• Error code</td>
<td></td>
</tr>
<tr>
<td>PlayStream event processing</td>
<td>• Event</td>
<td>• High processing delay</td>
</tr>
<tr>
<td></td>
<td>• Title</td>
<td>• Sudden burst of events</td>
</tr>
<tr>
<td></td>
<td>• Processing stage</td>
<td>• High processing failure rate</td>
</tr>
<tr>
<td></td>
<td>• Processing outcome</td>
<td></td>
</tr>
<tr>
<td>CloudScript execution</td>
<td>• Title</td>
<td>• High execution time</td>
</tr>
<tr>
<td></td>
<td>• Function</td>
<td>• High error rate</td>
</tr>
</tbody>
</table>
By setting dimensions on each metric for contextual information, activity can be tracked down to the individual API, PlayStream event, or game title. Issues that impact just one or a small number of customers can be quickly identified that would otherwise be lost in the noise, and then either resolved or else the customer(s) can be informed of required action. For example, customers can be quickly notified when their CloudScript functions have endless loop bugs or similar errors.

Incident management

Every member of the backend engineering team participates in the on-call rotation, so there is always someone ready to diagnose and respond to issues impacting the service 24x7x365. Since these are the same developers who build and maintain the service every day, they are best equipped to resolve incidents quickly and effectively.

VictorOps is used to manage the on-call rotation and route alerts, which are triggered by several redundant sources of service monitoring, including Datadog metrics, AWS CloudWatch metrics and Pingdom uptime monitors.

A server health check page is maintained at [http://status.playfab.com/](http://status.playfab.com/) with a list of any incidents impacting the service and their resolutions.

Limits

- No hard caps on RPS
- Configurable RPM limits for each API per player session and/or per client IP
- Configurable limits on count and size of per-player inventories, data values, leaderboard statistics etc.
- Configurable limits on other forms of resource consumption, such as CloudScript execution time
Security

Trustworthiness is one of PlayFab’s core values, and so security is taken very seriously. The most recent security audit was in October 2015 by the Veris Group. It returned no significant issues, and only five issue of ‘low’ concern. All of those issues have either since been fixed, or triaged as insignificant risks. The findings from the audit are available for review upon request.

Overall, PlayFab follows the following security-first principles:

- All communication with the API via TLS1.2
- Authentication via platform-specific methods (e.g. Google/Apple device, Facebook, Steam, XboxLive, PSN) employ server-side verification according to their specs
- Email / username / password authentication - password hashed / salted with bcrypt before storage
- All communication between machines and storage is within a VPC. The only port open to the public internet is port 443 (HTTPS) on the ELB
- Data can optionally be encrypted in transit or at rest within the AWS VPC as part of a private cloud using AWS Key Management Service

Data backup and restore

<table>
<thead>
<tr>
<th>Data</th>
<th>Backup procedure</th>
<th>Restore procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player data (DynamoDB)</td>
<td>Periodically scheduled Data Pipeline jobs copy DynamoDB table snapshots to S3 bucket in a separate region.</td>
<td>Data Pipeline job copies snapshot from S3 bucket to new DynamoDB tables.</td>
</tr>
<tr>
<td>Player profiles (SQL)</td>
<td>Daily full RDS database snapshots.</td>
<td>Restore database snapshot to a new RDS database.</td>
</tr>
<tr>
<td>PlayStream events (Kinesis)</td>
<td>Full copy of every event saved to S3 bucket.</td>
<td>PlayStream replay job loads events from S3 bucket and posts to stream or arbitrary HTTP endpoint.</td>
</tr>
</tbody>
</table>

Development Process

PlayFab relies heavily on automated build and test processes to achieve high reliability while still making frequent updates and improvements (often several times a week). Phabricator is used as the task management and code review tool, and Jenkins is the automated build and
test tool. Nearly every gate for the code submission process has been automated, because a lightweight process results in higher compliance from engineers - it's easy to do the right thing.

Steps for developing and deploying changes

1. Code changes for a new feature or bug fix are prepared in a Git feature branch by one or more developers.
2. Unit and integration tests are developed in parallel accompanying the change, with a minimum of 80% branch code coverage.
3. The code is submitted to Phabricator, which automatically schedules the change to be built and tested on Jenkins.
4. After the code change passes the complete test run of all unit and integration tests, it is published for peer code review.
5. Once the change is accepted by the reviewers, it is committed to the master branch, built, and deployed to a staging environment in AWS that mirrors production.
6. A final suite of acceptance tests is run against the staging environment. If all tests pass, the commit is considered ready to deploy to production.
7. A completely new set of machines, load balancers, etc. is launched in the production environment running the new build, using AWS CloudFormation Templates.
8. Using weighted DNS records, live traffic is gradually shifted to the new production environment, which is continuously monitored for errors or performance issues. If any issues are detected, traffic is reverted back to the previous environment.

The hardest part of the PlayFab code base to test automatically has until recently been the suite of SDKs. 14 SDKs are supported, across more than 6 devices including iOS, Android, PC, web browsers, Xbox, and Playstation. SDK bugs are especially frustrating for game developers, since they can grind development to a halt, yet the only way to test them in most cases is to run tests on the target devices.

Manual testing is not an option, since SDKs are auto-generated from API definition files. Each time a change is made to the backend platform, the SDKs are rebuilt.

To solve this problem, PlayFab has rolled out (and released to open source) an extension to Jenkins dubbed the “Jenkernaught” which sends new SDKs to remote devices, runs tests on the devices, then gathers up the logs and sends the results back to the central testing platform for analysis. This new process has resulted in a more stable SDK, with less time spent manually testing and deploying.

Load Testing

Load tests are run frequently to ensure that the service scales predictably. In addition to routine test to verify the performance of new features, stress tests are run ahead of new title launches
that are expected to stress the PlayFab platform in new ways. These runs simulate player sessions that match the usage patterns of the particular title.

The largest such test was recently run for a large multiplayer game company who wanted to know that PlayFab service could handle up to 1 million concurrent players.

Details:
- Simulated 1,000,000 concurrent players
- Provisioned 1 million unique player accounts, each with 600 items in the inventory

Results:
- 32 c4.4xlarge server instances, with 65% average CPU usage
- Results in 21K requests per second

<table>
<thead>
<tr>
<th>API call</th>
<th>Median response time</th>
<th>95% response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginWithAndroidID</td>
<td>63ms</td>
<td>185ms</td>
</tr>
<tr>
<td>GetTitleData</td>
<td>6ms</td>
<td>15ms</td>
</tr>
<tr>
<td>GetCatalogItems</td>
<td>1ms</td>
<td>5ms</td>
</tr>
<tr>
<td>GetUserInventory</td>
<td>388ms</td>
<td>577ms</td>
</tr>
<tr>
<td>GetStoreItems</td>
<td>1ms</td>
<td>7ms</td>
</tr>
<tr>
<td>AddUserVirtualCurrency</td>
<td>24ms</td>
<td>113ms</td>
</tr>
<tr>
<td>ConsumeItem</td>
<td>40ms</td>
<td>99ms</td>
</tr>
<tr>
<td>ValidateGooglePlayReceipt</td>
<td>119ms</td>
<td>282ms</td>
</tr>
</tbody>
</table>
Adventure Capitalist Launch

The following chart of player logins depicts how the service performed during the launch. The elastic architecture automatically scaled to handle the sudden jump in load without any issues, and without any sort of manual intervention.
API Performance for AdCap

- Average request execution time measured at load balancer: 70 ms
- Average end-to-end request round-trip latency observed in North America: 110 ms
- API endpoint uptime for last 12 months (measured by Pingdom): >99.99%

Since the AdCap launch, there have been other days in which a new title added more than 1M daily active players to the load without any prior warning, and the system has absorbed the load with no issues.

Single-Tenant Deployment Option

By default, games hosted on the PlayFab platform are hosted on a multi-tenant architecture. Great care is taken to ensure each game is isolated from other games, but the possibility does exist that a game’s performance may be impacted temporary by an especially heavy load on another game.

For customers demanding the highest level of security and isolation, PlayFab does offer a single-tenant deployment, in which the customer gets a fully independent instance of PlayFab running in a private cloud.
Advantages of this include:

- Full isolation from other games
- Control over timing of updates (e.g., no updates during key events)
- Direct access to all data.
- Direct access to full operational metrics (via DataDog or other tool), including many more metrics than are shared via the Game Manager.
- Full visibility into server & bandwidth costs, as the private cloud costs are passed through directly.
- Easier to deploy custom features or plugins.

Future Roadmap

A rich set of new features and services are planned to be rolled out over the next year. These include:

- New marketplace add-ons from prominent third party services, including Community Sift, Delta DNA, Scientific Revenue, Ironsource, Multiplay, Unity ads & analytics, and more.
- CloudScript support for C#, so server code can be easily developed alongside client code in Unity.
- Game client log and error collection, to help detect issues and debug games during soft-launch and release.
- A Localization module, to help operate games in multiple geographies.
- Batch actions across all players in a segment, such as giving a gift, or sending a message.
- Publish/subscribe support for PlayStream events by game clients, so clients can be notified in real-time about changes on the backend.
- Override Title Data for individual player segments, for more powerful A/B testing, and more targeted gameplay experiences.
- A Live Events module, to make it easier to setup and then run events in a game, such as a holiday event, or a weekly challenge.
- Synchronization of all PlayFab configuration data into Git, to facilitate easier Dev/Test/Live deployments of changes to a game.
- A Guild service, with guild data, statistics, leaderboards, and inventory/bank.
- Game Manager extensibility via “Canvas App” style plug-in apps and pages.
- Deployment of PlayFab into China, behind the great Chinese firewall, so games seeking to move to China may be operated in China by local partners without having to rip out and replace the backend platform.