Amplify DataStore: Use Cases and Implementation

AWS Whitepaper

Amplify DataStore: Use Cases and Implementation: AWS Whitepaper

Table of Contents

Amplify DataStore: Use Cases and Implementation

Publication date: **February 3, 2021** (*Document history* (p. 18))

Abstract

Amplify DataStore is a persistent on-device storage repository for developers to write, read, and observe changes to data. This whitepaper discusses the benefits, implementation details, and use cases of Amplify DataStore. This information will help your solutions architects, tech leads, and CTOs decide when and how to utilize Amplify DataStore in their solutions.

Introduction

If you develop applications, you need think about low-latency message transmission and offline synchronization. Your applications must not only deliver the required principal functionality, but also must work under conditions where data needs to be shared in near-real-time or in locations without internet connectivity.

For example, if someone sends a message on a chat application while the device lacks internet connectivity, the application must store that message on the device. Once connectivity is reestablished, the application must transmit it. This process can be complex to achieve, given that developers need to implement the code that manages the device connectivity, stores the information, and retries every time the internet access changes.

To meet these requirements, Amplify DataStore provides a programming model for leveraging shared and distributed data without writing additional code for offline, real-time, and online scenarios. This model makes working with distributed, cross-user data just as simple as working with local-only data. It also allows your application developers to focus on features that add business value rather than undifferentiated code to handle caching, reconnection, data synchronization, and conflict resolution.

Amplify DataStore can be used as a local data store in web and mobile applications without connecting to the cloud. When paired with AWS AppSync, Amplify DataStore synchronizes the application data with an application programming interface (API) when network connectivity is available. By automatically controlling versioning, conflict detection, and conflict resolution; Amplify DataStore automatically leverages AWS AppSync to achieve near-real-time synchronization between the devices and the cloud backend.

This guide discusses practical, real-life scenarios where developers, solutions architects, and organizations can adopt Amplify DataStore in their mobile and web application development process.

Amplify DataStore overview

Amplify DataStore allows your developers to focus on modeling their data and access patterns, adding authorization rules, and business logic where necessary. The infrastructure deployment and management are a function of these developer inputs. It is automatically deployed, monitored, and managed by the Amplify Framework including code generation, command line interface (CLI) rollouts, and runtime logic.

All DataStore operations are local first. This means that when you run a query, it returns results from the local system, which can be sorted and filtered. The same is true for mutations or observations to data. So, no network latencies or constraints on the backend are present. Operations are run for your developers to integrate into your application. However, they still have control to act upon things when the local store integrates back with the rest of the system, such as reacting to data updates or write conflicts when merges take place.

Syncing takes place without any needed effort, including real-time updates when the device is online. These updates are immediately present in any of your queries running locally or on data being observed. Similarly, devices transitioning from offline to online states perform delta synchronization on your behalf. This gives you control to perform logical actions if your mutations conflict with writes that took place to the backend.

Your developers don't need to worry about "subscribing to web sockets on the server" or "querying for the latest changes". Instead, they use the DataStore API against their local data and all these things happen automatically. So, all clients in the system behave in an eventually consistent manner and will converge to the latest records that are written to your cloud database.

Being an on-device persistent repository, DataStore allows developers to interact with local data while synchronizing it to the cloud. It leverages GraphQL to facilitate the data modeling process, providing authorization rules and business logic when needed. This is offered to developers through the Amplify CLI as well as by using the GraphQL Transformer.

Once the schema is defined, domain native structures called *models*, are generated. Your developers can then use the DataStore API to save, query, update, delete, or observe changes. The models are then passed to a *Storage Engine*, which is responsible for managing a repository of models. The engine contains a *Storage Adapter*, which provides a bridge for popular implementations, such as SQLite and IndexedDB.

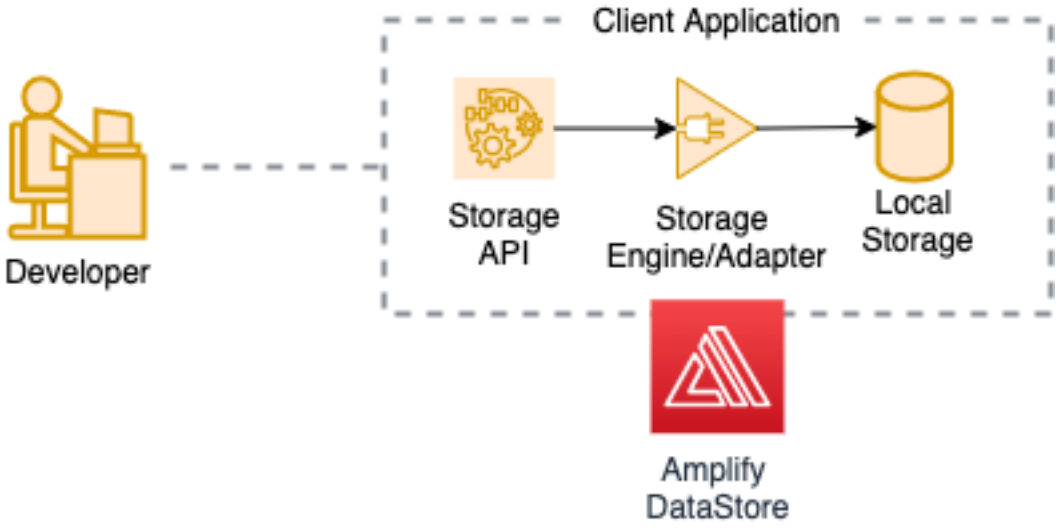

Figure 1 – Local storage model

To sync the models to the cloud, developers can use the Amplify CLI to create an AWS AppSync backend with Amazon DynamoDB tables that match the schema created at the application. Even though DynamoDB is the default database used by DataStore, the developer can choose another database to store the data by creating a custom AWS AppSync Resolver and data sources. Once the application starts interacting with the DataStore API operations, DataStore starts an instance of its *Sync Engine*, which will then interface with the Storage Engine to identify updates from the model repository.

To manage updates from the local repository to the cloud and vice versa, both the Sync Engine and the DataStore API subscribe to events published by the Sync Engine using the *Observer* pattern. This is how the Sync Engine knows how to communicate data changes generated by the application to the cloud. Similarly, this also allows developers to use the DataStore API to identify data changes that happened on the cloud, generated by other users that are manipulating applications that share the same underlying backend.

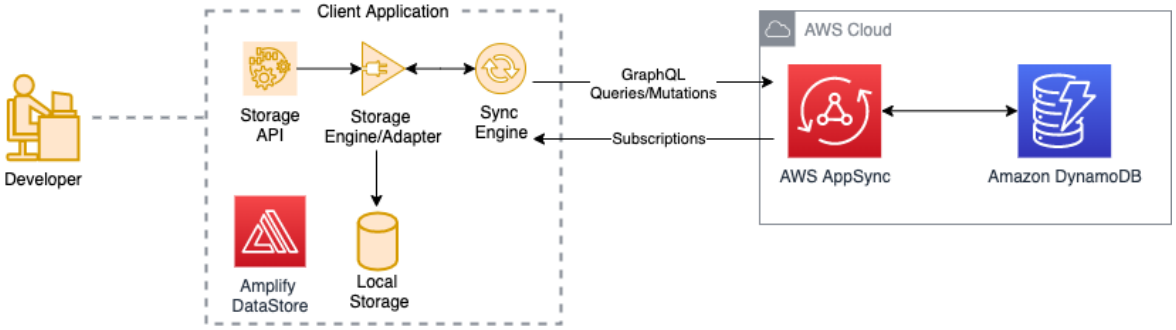

Figure 2 – Cloud sync process

If data synchronization to the cloud is enabled through AWS AppSync, multiple versions of the same object can exist on the client and on the server. This creates a need for the object to be resolved on the cloud, so a single version exists. When concurrent updates are sent from multiple clients at the same time, developers can configure conflict resolution to create strategies to define how the data is going to be stored. At this point, DataStore will apply conflict detection and resolution strategies to the data. Developers can choose between Automerge, Optimistic Concurrency, and Lambda.

Amplify DataStore supports iOS, Android, Flutter, and client JavaScript frameworks. This means that developers can build a single data schema design for multi-platform applications. From that schema, the Amplify CLI can generate models that fit solutions built in all these platforms.

Amplify DataStore architecture

Amplify DataStore relies on multiple components to deliver offline capabilities, near-real-time connectivity between devices, and conflict resolution. This section details these components and explains how they work together.

Amplify client libraries

The Amplify open-source client libraries provide purpose-built interfaces that act as a wrapper for helping the communication between the application code and the cloud backend. The libraries can be used both with new backends created using the Amplify CLI and with existing backend resources. Amplify DataStore is supported by the iOS, Android, Flutter, and JavaScript client libraries.

Amplify CLI

The Amplify Command Line Interface (CLI) is a unified toolchain to create, integrate, and manage the AWS Cloud services for your application. With the CLI, developers can create data models using the GraphQL Transform library and deploy them to AWS AppSync GraphQL API operations, and NoSQL databases. The CLI also helps developers manage the DevOps process from start to finish by facilitating the deployment of immutable cloud backends into multiple environments (such as development, QA, and production).

GraphQL

To create a data store, developers need to define a schema. A schema is a structured representation of the developer's models, containing their data types and relationships. GraphQL schema files are used to represent these models at the application level.

Once the schemas are defined, developers can use the CLI to convert the schema into CloudFormation templates. These templates are responsible for creating a cloud backend representation of the models.

AWS AppSync

AWS AppSync is a server-side managed component leveraged by Amplify DataStore to provide simplified access, querying, real-time updates, offline synchronization, caching, security, and fine-grained access control of the application data. The CLI is responsible for creating and configuring AWS AppSync based on the GraphQL schema files, defined at the application level. It provides an interface to NoSQL databases, relational databases, and AWS Lambda functions. The CLI is also responsible for data synchronization and conflict resolution for applications to sync data to the cloud.

Amazon DynamoDB

Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multiregional, multimaster, durable database with built-

in security, backup and restore, and in-memory caching for internet-scale applications. Using the CLI, developers can automatically create tables representing the schema defined at the application level on DynamoDB.

Although DynamoDB is Amplify DataStore's default database solution, developers can create AWS AppSync resolvers to use different purpose-built database offerings provided by AWS.

Amplify DataStore implementation

In this section, we cover the parts that define an implementation of Amplify DataStore, relying on platform standard data structures to represent the data schema in an idiomatic way.

Prerequisites

Amplify DataStore requires the following prerequisites for each platform:

- Installing and configuring the libraries for your development platform
- Configuring the Amplify CLI to set up the local development environment

For more information, see the Amplify documentation for JavaScript, Android, iOS, and Flutter.

Schema

The first step to create an application backed by a persistent datastore is to define a schema. DataStore uses GraphQL schema files as the definition of the application data model. The following schema contains data types and relationships that represent the application's functionality. In this example, the schema represents a whitepaper.

```
type Whitepaper @model {
  id: ID!
  title: String!
  status: PostStatus!
  pages: Int
  year: Int
}

enum PostStatus {
  DRAFT
  PUBLISHED
}
```

Idiomatic persistence

After the platform-agnostic model is created, the developer can use the Amplify CLI to generate the code associated to the model's platform standard data structure to represent the data schema in an idiomatic way. By configuring and initializing Amplify DataStore, the application is ready to start defining persistence operations using the model.

Also, leveraging platform-specific code, the persistence language is composed by data types that satisfy the model interface, generated automatically using the CLI, and operations defined by common verbs such as *save*, *query, observe,* and *delete*. DataStore also has the capability to handle relationships between models, such as *has one, has many,* and *belongs to*.

Cloud sync

To allow communication between multiple devices, there must be synchronization between offline and online data. The Amplify DataStore goal is to facilitate the process of the application when handling online and offline scenarios, handling all data consistency and reconciliation between local and remote behind the scenes. This allows developers the opportunity to focus on application logic.

Using AWS AppSync, Amplify DataStore guarantees that the local application data is synchronized with a cloud backend. DataStore can also connect to an existing AWS AppSync backend that has been deployed from another project, no matter the platform it was originally created in.

DataStore makes the process of data synchronization as transparent as possible to developers. However, considerations are required for some scenarios where local data might be out-of-sync with the backend, generating conflicts locally and on the cloud. Conflict resolution is a strategy that AWS AppSync uses to manage multiple versions of the same object on the client and server, as mentioned in Amplify DataStore Overview (p. 3).

Authorization rules

Amplify can also give developers the ability to limit which individuals or groups should have access to create, read, update, or delete data on your types by specifying an authorization directive. This limits the model access to an owner, group, make it public or private, based on AWS IAM or Amazon Cognito User Pool or a third-party provider, using an OpenID Connect (OIDC) provider.

Events

If the device has pending data to be synchronized, loses or regains connectivity, or synchronizes the local data with the cloud, DataStore publishes a set of states that can be captured by the application. The developer can then choose to add business logic to these events. For example, if the application needs to show the user that a particular model has synced with the backend, the developer can create a listener to the *modelSynced* event and notify the user, after that. For a list of events, see the documentation for JavaScript, iOS, Android, and Flutter.

Amplify DataStore best practices

Clear offline data on sign-in and sign-out

We recommend that you clear out user-specific data for shared device scenarios. This prevents security and privacy issues. If the application has authentication enabled and your schema defines user-specific data, make sure to clear the data stored during a previous session. Developers can do this by calling the DataStore.clear() API method.

Immutable models

Models in DataStore are immutable. Manually forcing a synchronization with the backend is not possible. The following example shows how to create a record:

```
await DataStore.save(
  new Whitepaper({
    title: "Amplify DataStore - Use cases and implementation",
    pages: 30,
    year: 2021
  })
);
```

To update a record, you must use the `copyOf` function provided by your library of choice to apply updates to the item's fields rather than mutating the instance directly.

```
const original = await DataStore.query(Whitepaper, "123");

await DataStore.save(
  Whitepaper.copyOf(original, updated => {
    updated.title = `title ${Date.now()}`;
  })
);
```

Start developing your application in offline mode

To avoid updating the cloud backend, it is a recommended practice to develop the application on offline mode until the developer is comfortable with the schema defined. After the model is considered stable, the cloud synchronization can be enabled, and the local data will be automatically synchronized with the backend.

Be aware of how schema changes affect offline data

If the developer decides to modify the application's schema, Amplify DataStore will evaluate at start-up to decide if these changes impact the storage structure of the application already deployed to users. If

it confirms that the schema lost its integrity, DataStore will remove the items stored and will perform a sync with the backend if cloud sync is enabled.

Sync configuration with base and delta queries

If cloud sync is enabled, at the first application start, the Sync Engine will run a GraphQL query that populates the Storage Engine from the network using a *base query*, which is the Sync Engine's most basic form of synchronization that retrieves the baseline state of your records. This will replicate the current state of your backend with your client application. If your solution relies on a limited amount of data, this solution could fit your needs and you might want to continue to replicate the backend constantly.

For cases where the solution relies on large datasets and the connectivity on the client changes frequently, the base query might become cumbersome and your application could face performance issues. For this scenario, a second query can be performed, called a *delta query*. The delta query guarantees that, once the base query runs and hydrates the cache for the first time, on every network reconnection only the changed data is captured.

Selective sync

To download and persist only a subset of the data, developers can benefit from a selective synchronization approach. This will limit the number of records that your application collects from the backend by filtering them by expressions or by limiting the number of items that will be stored. This strategy can be applied per device and user, meaning that you can have different rules for different use cases. For example, if an application needs to display multiple alert levels that groups of users can handle separately, developers can create the following:

```
DataStore.configure({
  syncExpressions: [
    syncExpression(Post, () => {
      return alert => alert.level('gt', 5);
    })
  ]
});
```

This will guarantee that only alerts with levels that are greater than 5 will be synced to the device's local storage.

Use cases and implementation

Messaging and collaboration

Current businesses require real-time collaboration between multiple parties. Messaging and collaboration applications are essential tools to enable business processes and workflows.

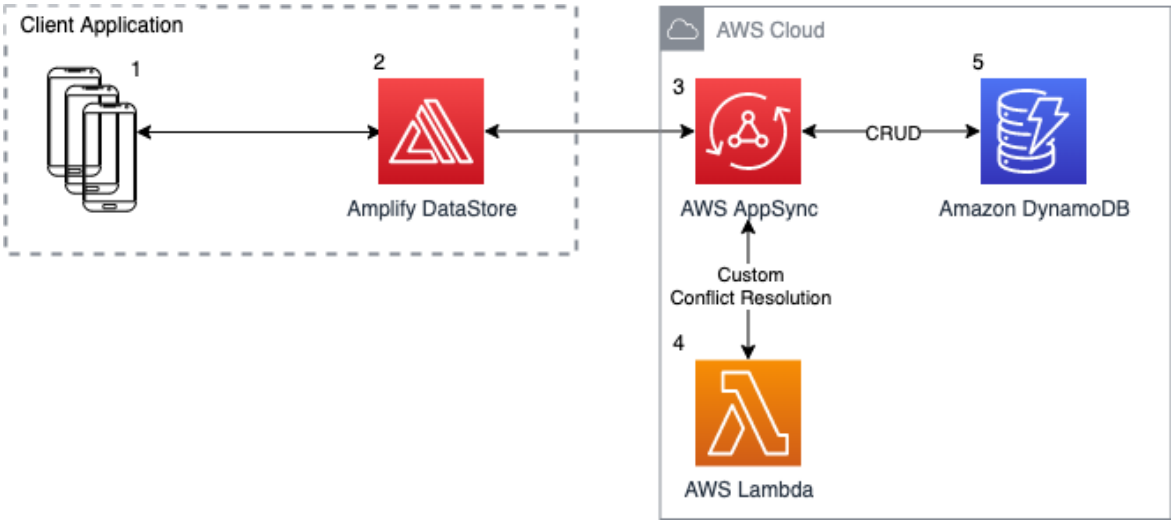

Figure 1 – Amplify DataStore messaging and collaboration use case with custom conflict resolution

1. Multiple users use messaging and collaboration applications. Each user creates, reads, updates, and deletes data at the application level. Some users can be offline (for example, on a flight or inside a subway with limited connectivity).
2. Amplify DataStore receives the data on each user's local devices, both for online and offline users.
3. AWS AppSync synchronizes data between online users. When offline users come back online, there might be conflicts between data updated by offline users and online users.
4. AWS Lambda provides custom business logic to resolve the conflict and instruct AWS AppSync resolvers to update the data based on AWS Lambda function decisions.
5. Data is updated into Amazon DynamoDB in the backend.

By default, Amplify DataStore supports Auto Merge and Optimistic Concurrency conflict resolution. Developers can also build their custom conflict resolution strategies with Custom Lambda functions. When all devices are back online, they will be sharing the same underlying local data, without conflicts.

Real-time subscriptions

Amplify DataStore supports real-time subscriptions for different use cases.

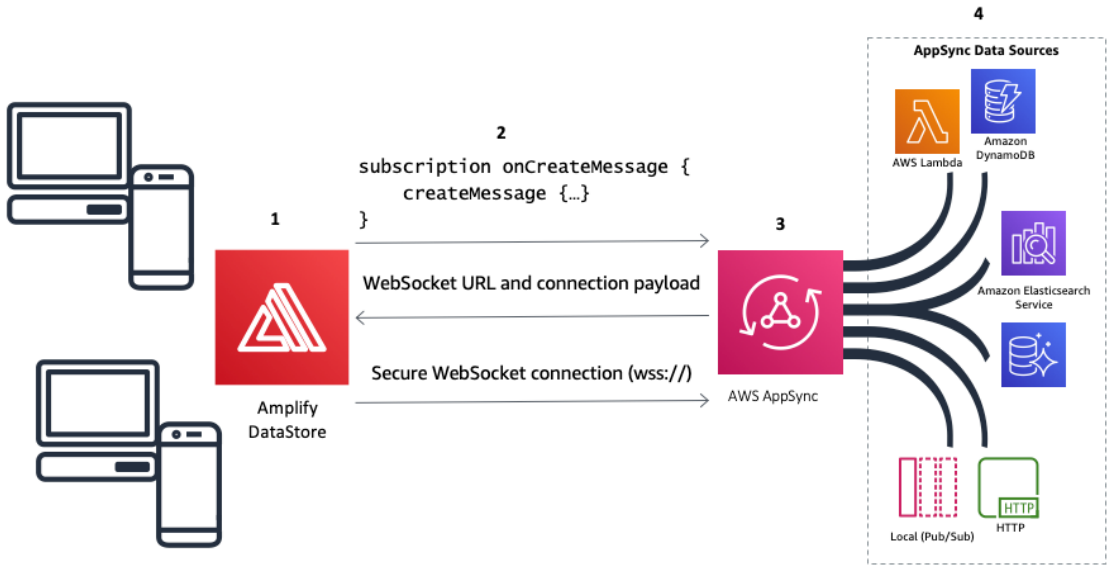

Figure 2 – Amplify DataStore and real-time subscription use cases

1. Subscription applications are used to receive updates about specific changes in data. Each user accessed the applications from web browsers or native iOS or Android applications. Amplify DataStore libraries are used to build the subscription features.
2. Amplify DataStore sync-engine creates a secure WebSocket connection to AWS AppSync. Developers don't have to write and maintain WebSocket code because Amplify DataStore libraries provide this functionality.
3. AWS AppSync synchronizes the subscription requests and provides subscribed data to Amplify DataStore.
4. AWS AppSync resolvers process the data request to different data backends, such as AWS Lambda, Amazon DynamoDB, Amazon Elasticsearch Service, HTTP endpoint, or local publish/subscribe messaging service.

Retail inventory counting

Physical inventory counting is a challenge for most organizations. It's a task that still requires manual interaction from employees and a very mature level of organizational skills to establish a successful process. A small or medium-sized organization usually stores products in display and a warehouse, spread in multiple rooms, bins, and boxes. This makes the counting process complex.

With Amplify DataStore's observe and data sharing capabilities, multiple users can perform the inventory counting process simultaneously at different places. All the information entered at the application will be shared instantly between devices, keeping all users in sync. In this particular case, the data also needs to be persisted on SAP, a popular enterprise resource planning (ERP) system.

The following example could also work with different ERP systems or databases.

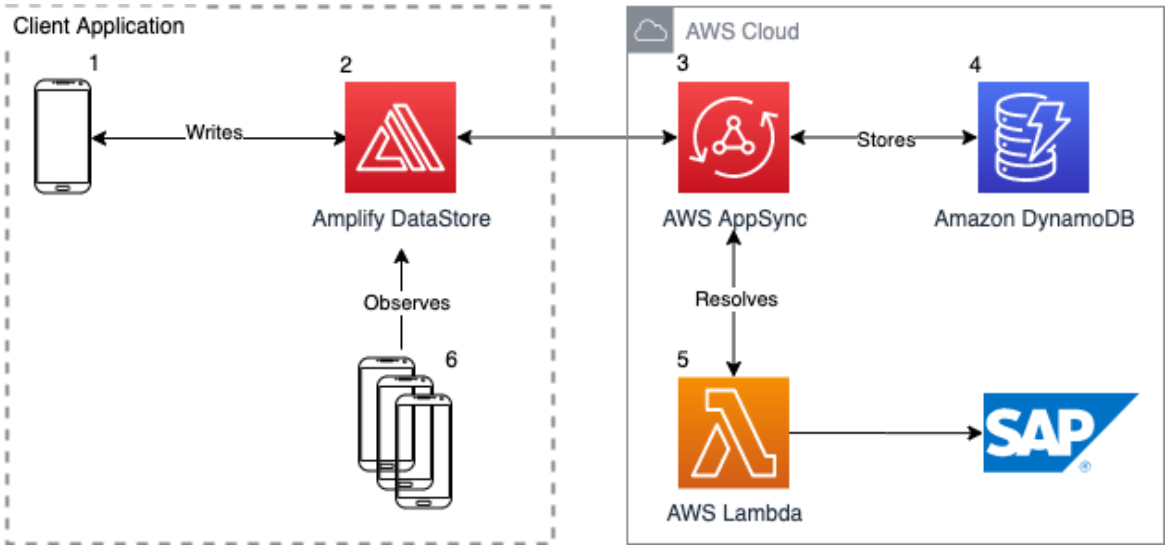

Figure 3 – Amplify DataStore retail inventory use case

1. The process begins with a user counting a particular product at a warehouse using a mobile application. This product is spread throughout many bins, so multiple people are adding up the same type of item simultaneously.
2. The data is stored locally at the device level using the Storage Engine provided by Amplify's libraries.
3. Amplify DataStore uses AWS AppSync to write and observe modifications that are made to the underlying data. AWS AppSync is mainly responsible for syncing the data stored at the device with the cloud.
4. The data is stored at DynamoDB tables, which are also managed by AWS AppSync. At this point, the data stored in DynamoDB can be further used on different applications and workloads.
5. AWS AppSync calls a Lambda Resolver to update the inventory on the ERP system, in this case represented by SAP.
6. Because AWS AppSync is also responsible for observing the underlying changes in the data stored, the devices that leverage the Amplify libraries are notified of a modification on the data that they are set to observe. At this point, all devices have the same information provided by the user during the first step.

Remote transportation tracking

Let's use an example scenario. A large company in Texas needs to manage the transportation of goods and services to multiple oil fields. Over one thousand shipments are made daily by various trucks on a 24/7 operation. This process needs to be closely monitored by the oil field managers who expect the shipment to arrive on time, the logistics manager who keeps the fleet organized, and the incident response team responsible for the security of the drivers and trucks.

An application responsible for managing these shipments needs to function in remote locations, where internet connection can be scarce. Amplify DataStore offers offline storage, maintaining the drivers' application functions when they lose cellular or Wi-Fi connectivity, and real-time updates as soon as the connection is restored. These capabilities prevent the work from being hindered because it allows the application to be fully functional in online and offline scenarios.

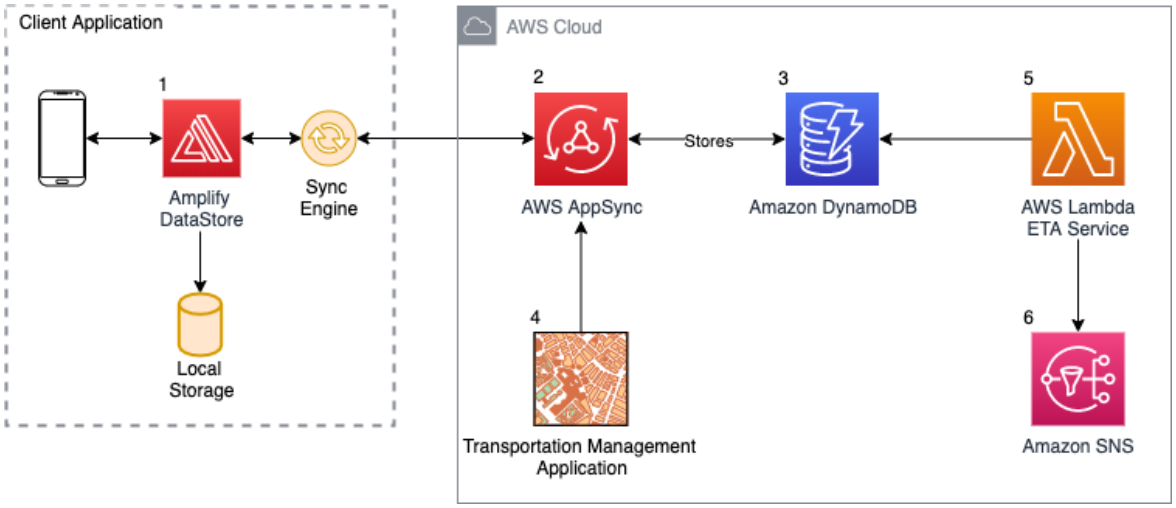

Figure 4 – Amplify DataStore transport tracking use case

1. A driver who is responsible for delivering a shipment to an oil field in West Texas uses a mobile application to track his truck's current location and the status of this delivery. Because his device is offline due to poor cellular connectivity in the area, the mobile app uses DataStore API's Storage Manager to keep that data stored at the device level, for the duration of the period where the device is offline.
2. Once the device reaches an area with cellular coverage and an internet connection is detected, Amplify DataStore instantly uses the Sync Engine to transmit the data to AWS AppSync.
3. AWS AppSync manages the underlying DynamoDB tables, where the shipment information is persisted. From there, the data can be provided to the logistics manager and the incident response team.
4. The Transportation Management Application uses the same backend created for the mobile application, thus observing changes for every trip that is synced to the cloud. This application provides dashboards for oil field managers, logistics operators, and the security response team to visualize the trips in real time and make quick decisions based on events.
5. The ETA Service collects data from the mobile devices and calculates the estimated time of arrival for each shipment. When the ETA is calculated, the value is updated to DynamoDB.
6. Events generated by the trips (change in ETA, for example) use Amazon Simple Notification Service (Amazon SNS) to notify users of the event.

Application feature update

A global retail company wants to test a new idea from its marketing team. The idea is to enable a simplified purchasing flow targeting customers from selected regions of the world.

The company already has the data about the customer's location and wants to silently enable the feature to them, without the need to keep multiple versions of the application available for download. In the end, they need to select a few customers, enable the feature, and handle the modified flow on the application side.

The company can flag certain customers on the database and, using the observe functionality, the application will detect the changes and modify itself to present the new flow, without requiring users to download an updated version of the application.

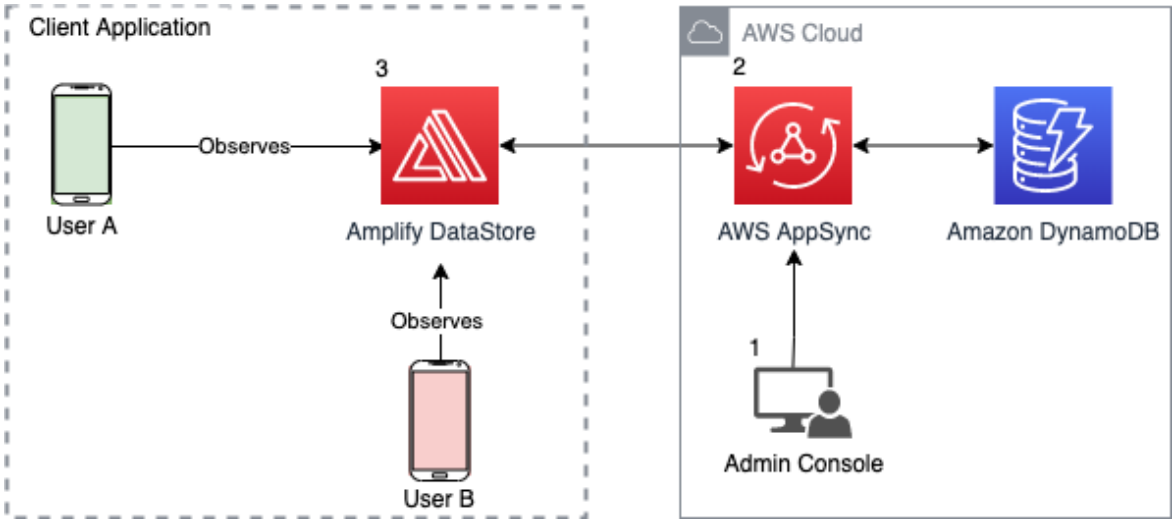

Figure 5 – Amplify DataStore application feature update use case

1. The marketing team wants to modify the purchasing flow of the user population identified as *User B* on the diagram. They use an admin console that uses the same backend that was created for the mobile application.
2. AWS AppSync manages the underlying DynamoDB tables, where the information is persisted. From there, AWS AppSync syncs the data with Amplify DataStore.
3. The devices running the mobile application with Amplify DataStore's libraries observe the changes and adapt, based on the rules provided by the market team.

Conclusion

AWS Amplify is a set of products and tools that enable mobile and front-end web developers to build and deploy secure, scalable full-stack applications. With Amplify, you can configure application backends in minutes, connect them to your application in just a few lines of code, and deploy static web applications in a few steps.

Developers can benefit from Amplify DataStore with various applications that require offline capabilities and near-real-time sync between devices. This whitepaper offers a first step to help users understand the functionalities behind DataStore, implementation practices, associated AWS services, and use cases that can help you decide if the technology is a good fit for your application and your organization.

Without the need to write undifferentiated code to manage offline and online scenarios, you can benefit from a multi-platform on-device persistence storage available for JavaScript, iOS, Android, and Flutter. DataStore automatically synchronizes data between your application and a cloud backend, powered by AWS. This makes it simple for your developers to build user-centric applications and rely on an offline-first approach.

Contributors

Contributors to this document include:

- **Fernando Rocha Silva**, Solutions Architect, Amazon Web Services
- **Ivan Artemiev,** Software Development Engineer – Mobile Applications, Amazon Web Services
- **Ashish Nanda,** Software Development Engineer – Mobile Applications, Amazon Web Services
- **Richard Threlkeld,** Principal Front-End Engineer – Mobile Applications, Amazon Web Services
- **Sigit Priyanggoro**, Sr Partner Solutions Architect, Dedicated Edge, Amazon Web Services
- **Steve Johnson**, Tech Leader – Mobile, Amazon Web Services
- **Brice Pellé**, Principal Solutions Architect – Mobile, Amazon Web Services

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

update-history-change	update-history-description	update-history-date
Initial publication (p. 18)	Whitepaper first published	February 3, 2021

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.