Offline GraphQL apps with AWS AppSync

Karthik Saligrama, SDE , AWS Mobile

April, 2018

What are we doing today?

- What is GraphQL
- AWS AppSync
- Offline Application Development
- Demo

Offline/Real-time use cases

Users expect data availability offline

- Financial transactions
- News articles
- Games
- Messaging (pending chat)
- Document collaboration

Users expect data immediately

- Banking alerts
- News stories
- Multi-player games
- Chat applications
- Shared whiteboards
- AR/VR experiences
- Document collaboration

What is GraphQL?

- Application Query Language
- Agnostic of underlying Data Store
- != Graph Database
- Optimized for Performance and flexibility

How does GraphQL work?

```
type Query {
    getTodos: [Todo]
}

type Todo {
    id: ID!
    name: String
    description: String
    priority: Int
    duedate: String
}
```

Model data with application schema

```
query {
  getTodos {
    id
    name
    priority
  }
}
```

Client requests what it needs

```
{
     "id": "1",
    "name": "Get Milk",
     “priority": "1"
},
{
     "id": “2",
    "name": “Go to gym",
     “priority": “5"
},…
```

Only that data is returned

What are the GraphQL benefits?

- Rapid prototyping and iteration
- Introspection
- Client Performance First

REST/RPC

```
/posts?include=title,author
```

GraphQL

```
posts {
   title
   author
}
```

What are the GraphQL benefits?

- Delegates Power to Clients

```
/posts
```

```
/postsJustTitle

/postsWithTitleAndAuthor

/postsWithTitleAuthorAndContent

/postsWithTitleAuthorContentAndImages

/postsWithTitleAuthorContentImagesAndComments
```

What are the GraphQL benefits?

- Delegates Power to Clients

```
/graphql
```

```
posts {
   title
   author
}
```

```
posts {
   title
   authorName
   content
}
```

```
Posts (maxSize: 10) {
   title
   authorName
   content
}
```

```
Posts (maxSize: 10) {
   title
   author {
     firstName
     lastName
     imageUrl
   }
   content
}
```

```
Posts (maxSize: 10) {
   title
   author {
     firstName
     lastName
     imageUrl(size:80)
   }
   content
   comments{
     user
     text
   }
}
```

What are the GraphQL benefits?

- Include vs Endpoint & Reduction in call Volumes

REST/RPC

```
/posts?include=title,authors

/posts?include=title,authors,authors.firstname, authors.lastname
```

Hypermedia

```
{
  ...
  “author”:{
     “_links”: {
        “self”:https://api.example.com/api/author/foo
     }
  }
  ...
}
```

GraphQL

```
posts {
   title
   authors {
       firstname
       lastname
   }
}
```

What is AWS AppSync?

AWS AppSync is a managed service for application data using GraphQL with real-time capabilities and an offline programming model.

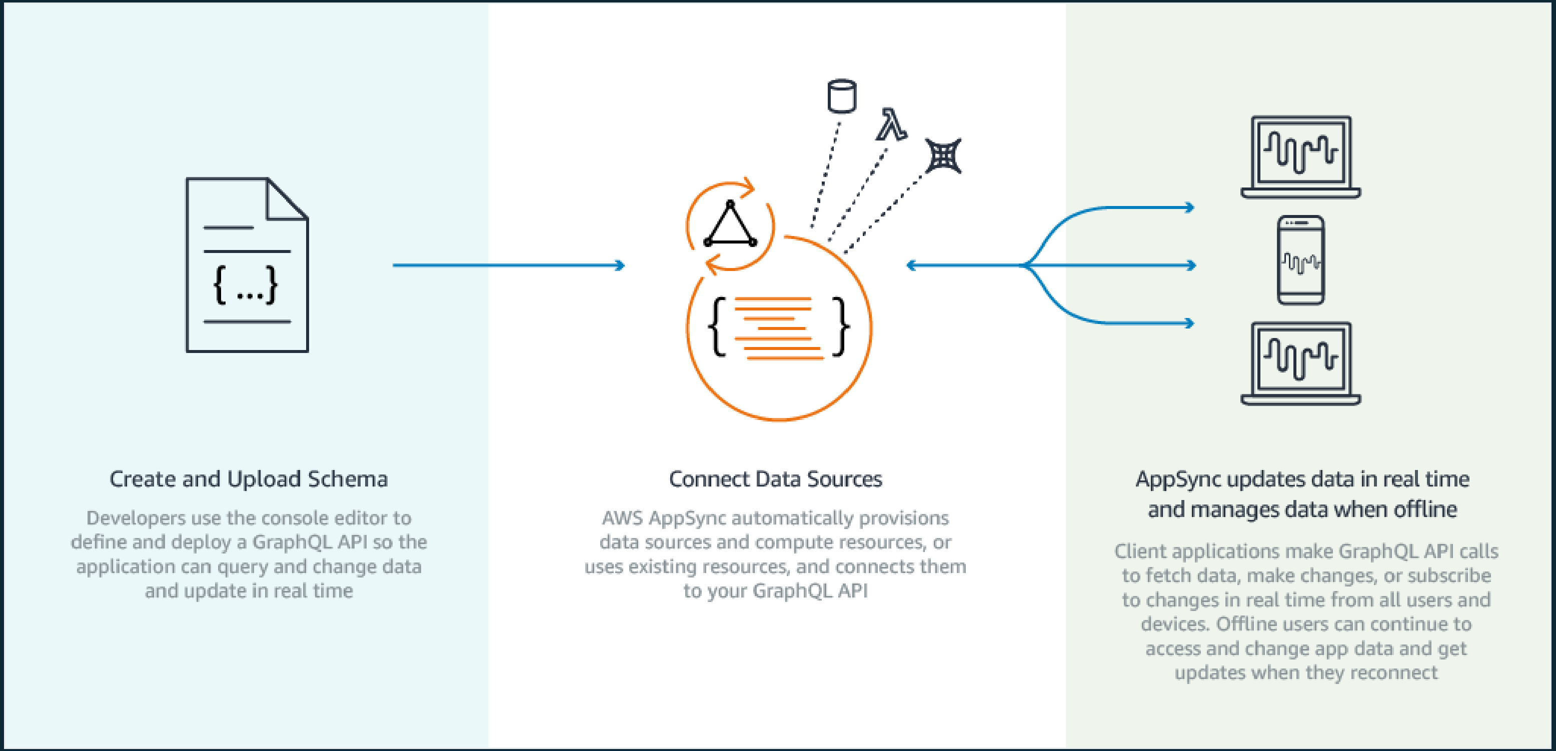
Real-time Collaboration

Offline Programming Model with Sync

Flexible Database Options

Fine-grained Access Control

How does AWS AppSync work?



Create and Upload Schema

Developers use the console editor to define and deploy a GraphQL API so the application can query and change data and update in real time

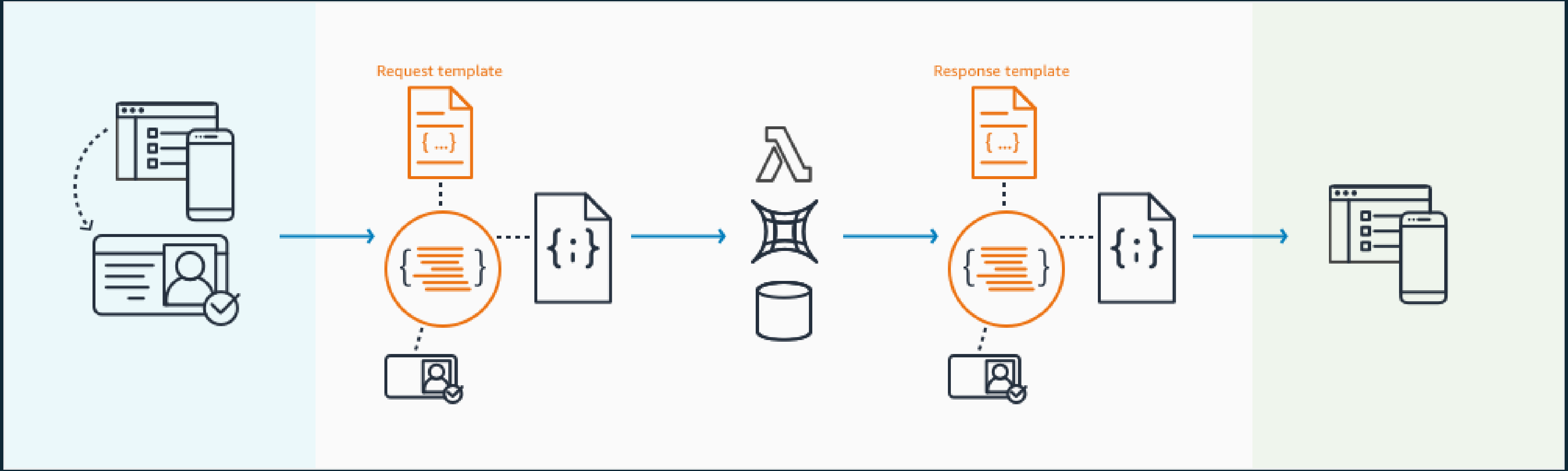
Connect Data Sources

AWS AppSync automatically provisions data sources and compute resources, or uses existing resources, and connects them to your GraphQL API

AppSync updates data in real time and manages data when offline

Client applications make GraphQL API calls to fetch data, make changes, or subscribe to changes in real time from all users and devices. Offline users can continue to access and change app data and get updates when they reconnect

GraphQL data flow in AWS AppSync



Mocking VTL

```
{
  "arguments": {},
  "source": {},
  "identity": {
    "sub": "uuid",
    "issuer": "https://cognito-
idp.{region}.amazonaws.com/{userPoolId}",
    "username": "nadia",
    "claims": {},
    "sourceIp": [
      "x.x.x.x"
    ],
    "defaultAuthStrategy": "ALLOW"
  }
}
```

➕

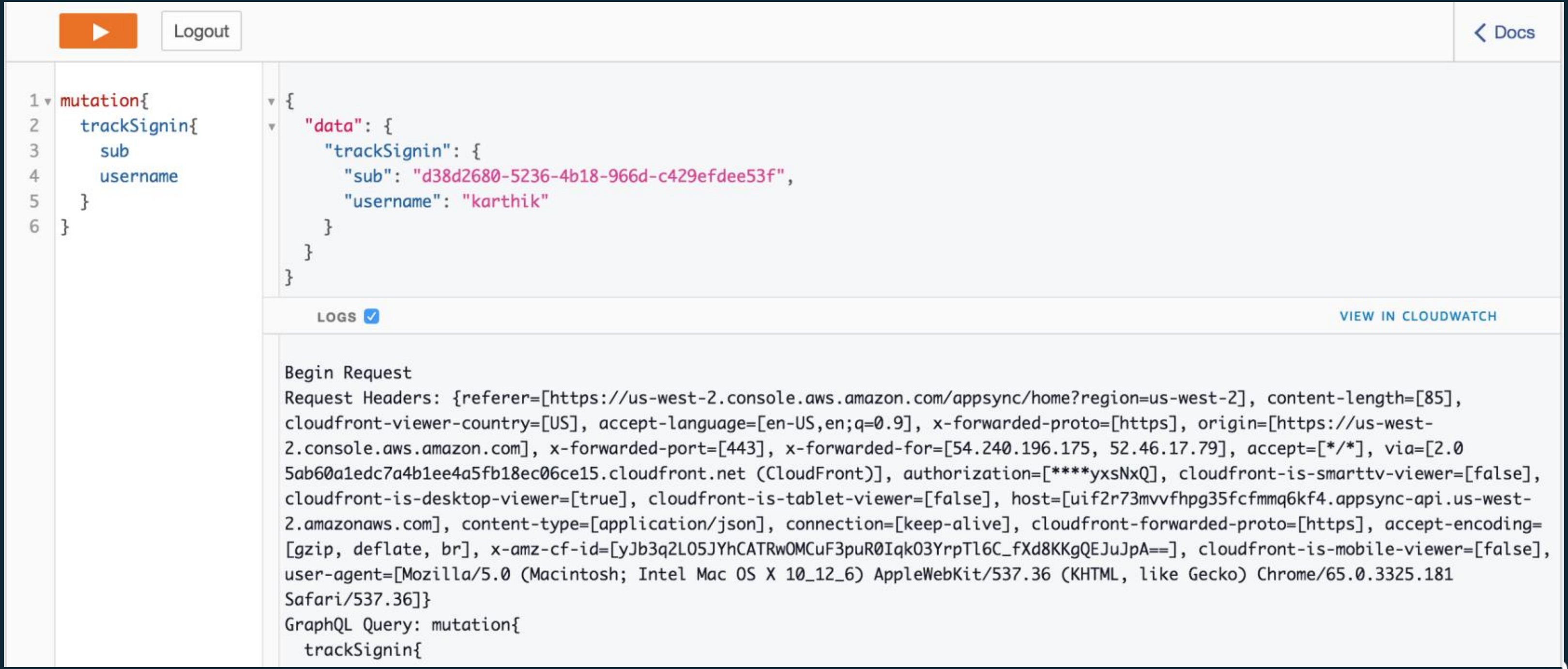
```
{
    "version" : "2018-05-29",
    "operation" : "PutItem",
    "key" : {
        "sub":
$util.dynamodb.toDynamoDBJson($ctx.identity.sub),
    },
    "attributeValues" :{
       "username": { "S" :
"${ctx.identity.username}" },
        "lastLoginTime": { "S" :
"$util.time.nowEpochMilliSeconds()" }
    }
}
```

➡

```
{
    "version" : "2018-05-29",
    "operation" : "PutItem",
    "key" : {
        "sub":
$util.dynamodb.toDynamoDBJson($ctx.
identity.sub),
    },
    "attributeValues" :{
       "username": { "S" :
"${context.identity.username}" },
        "lastLoginTime": { "S" :
"$util.time.nowEpochMilliSeconds()"
}
    }
}
```

Debug Resolver Flow

-Amazon CloudWatch logs



```
mutation{
  trackSignin{
    sub
    username
  }
}
```

```
{
  "data": {
    "trackSignin": {
      "sub": "d38d2680-5236-4b18-966d-c429efdee53f",
      "username": "karthik"
    }
  }
}
```

```
Begin Request
Request Headers: {referer=[https://us-west-2.console.aws.amazon.com/appsync/home?region=us-west-2], content-length=[85],
cloudfront-viewer-country=[US], accept-language=[en-US,en;q=0.9], x-forwarded-proto=[https], origin=[https://us-west-
2.console.aws.amazon.com], x-forwarded-port=[443], x-forwarded-for=[54.240.196.175, 52.46.17.79], accept=[*/*], via=[2.0
5ab60a1edc7a4b1ee4a5fb18ec06ce15.cloudfront.net (CloudFront)], authorization=[****yxsNxQ], cloudfront-is-smarttv-viewer=[false],
cloudfront-is-desktop-viewer=[true], cloudfront-is-tablet-viewer=[false], host=[uif2r73mvvfhpg35fcfmmq6kf4.appsync-api.us-west-
2.amazonaws.com], content-type=[application/json], connection=[keep-alive], cloudfront-forwarded-proto=[https], accept-encoding=
[gzip, deflate, br], x-amz-cf-id=[yJb3q2LO5JYhCATRwOMCuF3puR0IqkO3YrpTl6C_fXd8KKgQEJuJpA==], cloudfront-is-mobile-viewer=[false],
user-agent=[Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181
Safari/537.36]}
GraphQL Query: mutation{
  trackSignin{
```

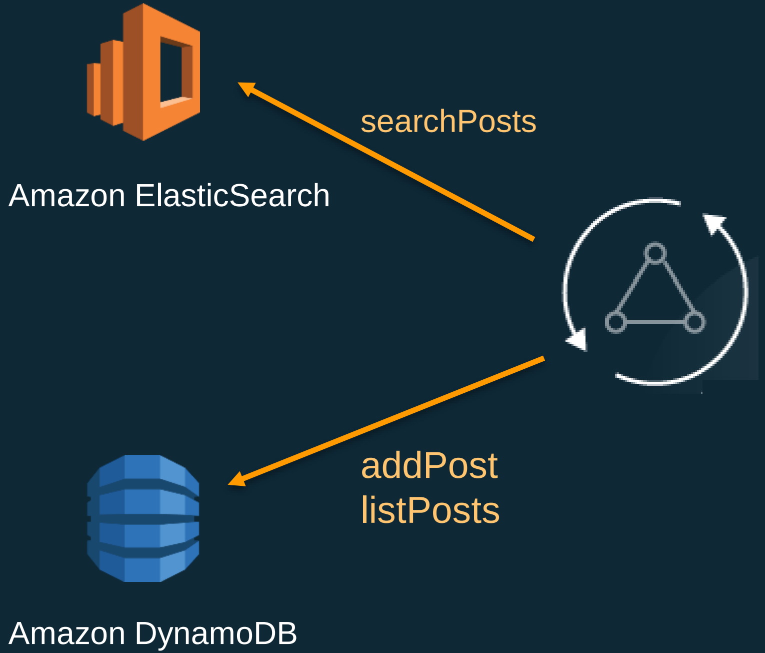
Demo

Mix/Match datasources on GraphQL fields

```
type Query {
    listPosts: [Post]
    searchPosts: [Post]
}


type Mutation {
    addPost: Post
}


type Post {
    id: ID!
    content: String
    description: String
    ups: Int
    downs: Int
}
```



Offline Application Considerations

- Local Storage (R/W)
- Order of Operations
- Network State Management
- UI Updates
- Conflict Resolution

AWS Mobile SDK + AWS AppSync

AWS Mobile SDK + AWS AppSync

iOS

```
let appSyncConfig = try AWSAppSyncClientConfiguration(
                url: AppSyncEndpointURL,
                serviceRegion: .USWest2,
                userPoolsAuthProvider: self,
                s3ObjectManager: AWSS3TransferUtility.default())

let appSyncClient = try AWSAppSyncClient(appSyncConfig: appSyncConfig)
```

Android (Kotlin)

```
val appsyncClient = AWSAppSyncClient.builder()
                .context(this.applicationContext)
                .cognitoUserPoolsAuthProvider(this)
                .region(Regions.US_WEST_2)
                .serverUrl(Constants. APPSYNC_API_URL)
                .build()
```

AWS Mobile SDK + AWS AppSync

iOS

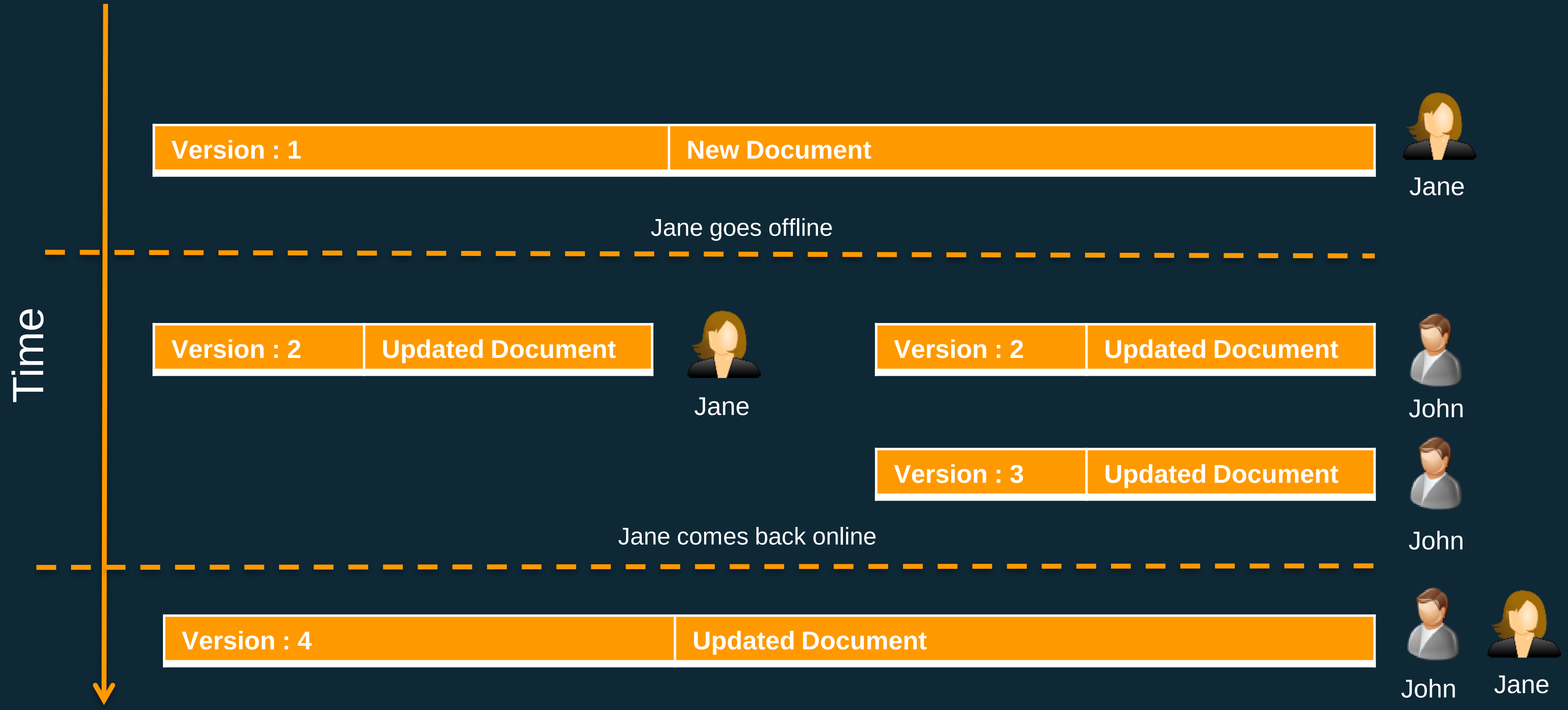
```
let trackSignInMutation = TrackSignInMutation()
self.appSyncClient?.perform(mutation: trackSignInMutation){ (result, error) in
                if let error = error as? AWSAppSyncClientError {
                    print("Error occurred: \(error.localizedDescription )")
                    return
                }
                ...//do more logic
            }
```

Android (Kotlin)

```
var trackSignIn = TrackSignInMutation()
appsyncClient!!.mutate(signup).enqueue(object : GraphQLCall.Callback<TrackSignInMutation.Data>() {
            override fun onFailure(e: ApolloException) {
                Log.e(TAG, "Failed signup mutation", e)
            }

            override fun onResponse(response: Response<SignUpMutation.Data>) {
                Log.i(TAG, response.data().toString())
                 //more business logic
            }
        })
```

Offline mutations



Conflict Resolution and synchronization

Conflict resolution in the cloud

1. Server wins
2. Silent reject
3. Custom logic (AWS Lambda)

- Optimistic version check
- Extend with your own checks

Optional

- Client callback for Conflict Resolution is still available as a fallback

Example: Check that an ID doesn’t already exist:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        "id" : { "S" : "1" }
    },
    "condition" : {
        "expression" : "attribute_not_exists(id)"
    }
}
```

Run Lambda if version wrong:

```
"condition" : {
     "expression" : "someExpression"
     "conditionalCheckFailedHandler" : {
         "strategy" : "Custom",
         "lambdaArn" : "arn:..."
     }
}
```

Images and rich content

```
type S3Object {
   bucket: String!
   key: String!
   region: String!
}
```

```
type Profile {
   name: String!
   profilePic: S3Object!
}
```

```
input S3ObjectInput {
   bucket: String!
   key: String!
   region: String!
   localUri: String!
}
```

```
type Mutation {
   updatePhoto(name: String!,
                    profilePicInput: S3ObjectInput!): Profile
}
```

GraphQL Subscriptions

Near Realtime updates of data

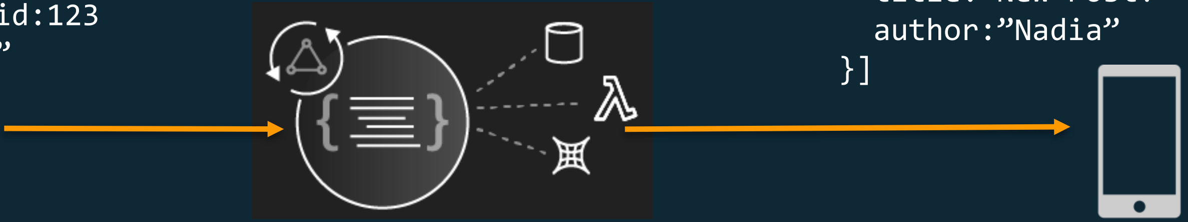
Event based mode, triggered by Mutations

- Scalable model, designed as a platform for common use-cases

Can be used with ***ANY*** data source in AppSync

- Lambda, DynamoDB, Elasticsearch

```
mutation addPost( id:123
  title:"New post!"
  author:"Nadia"){
  id
  title
  author
}
```



```
data: [{
  id:123,
  title:"New Post!"
  author:"Nadia"
}]
```

Schema directives

```
type Subscription {
    addedPost: Post
    @aws_subscribe(mutations: ["addPost"])
    deletedPost: Post
    @aws_subscribe(mutations: ["deletePost"])
}


type Mutation {
    addPost(id: ID! author: String! title:
     String content: String): Post!
    deletePost(id: ID!): Post!
}
```

```
subscription NewPostSub {
    addedPost {
        __typename
        version
        title
        content
        author
        url
    }
}
```

Demo

Best practices

- Mutations offline – what UIs actually need to be optimistic?

- Use Subscriptions appropriately
    - Large payloads/paginated data: Queries
    - Frequent updating deltas: Subscriptions
    - Be kind to your customer’s battery & CPU!

- Don’t overcomplicate Conflict Resolution
    - Data model appropriately, many app actions simply append to a list
    - For custom cases, use a AWS Lambda and keep client logic light (race conditions)

https://aws.amazon.com/appsync/