



AWS AppSync

GraphQL APIs for serverless webapps and real time data

Richie Schmid - Senior Developer



13.09.2018

blog.webgate.biz

Serverless Apps are cool

- No servers
- No patching
- No capacity guessing
- Pricing
- etc etc.. well, you (hopefully) already know all the advantages..



*Look, mom!
No servers!*

Serverless Apps with serverless components

Typical REST API combo for WebApps:



Übersicht Meine Absenzen Neuer Absenz Antrag

Alle Abwesenheiten heute

< << 25.10.2017 >> >

Datum	Dauer	Mitarbeiter/in	Typ
✓ 2017-10-25	Nachmittag	Natascha Walzl	Ferien
✓ 2017-10-25	Ganzer Tag	Christian Guedemann	Ferien

Abwesenheiten nach Abteilung

ENGINEERING AND OPERATIONS ▾

<< 10.2017 >>

Klaus Bild	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Philipp Pfister	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Zoran Mitrovic	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Daniel Briner	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Antonio Casu	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

WebGate Absenzen

Apps ▾ Richard Schmid ▾

Übersicht Meine Absenzen Neuer Absenz Antrag

Alle Abwesenheiten heute 25.10.2017

Datum	Dauer	Mitarbeiter/in	Typ
✓ 2017-10-25	Nachmittag	Natascha Walzl	Ferien
✓ 2017-10-25	Ganzer Tag	Christian Guedemann	Ferien

Abwesenheiten nach Abteilung ENGINEERING AND OPERATIONS 10.2017

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Klaus Bild	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Philipp Pfister	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Zoran Mitrovic	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Daniel Briner	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Antonio Casu	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Dashboard API Calls:

- Today's absences

`/getAbsencesByDate`

- List of departments

`/getDepartments`

- Team Absences, depending on user Department

`/getAbsencesByDepartment`

→ **Provider driven API, instead of consumer driven**
 → **Many REST calls instead of just one**

Übersicht Meine Absenzen Neuer Absenz Antrag

Alle Abwesenheiten heute

< << 25.10.2017 >> >

Datum	Dauer	Mitarbeiter/in	Typ
✓ 2017-10-25	Nachmittag	Natascha Walzl	Ferien
✓ 2017-10-25	Ganzer Tag	Christian Guedemann	Ferien

Abwesenheiten nach Abteilung

ENGINEERING AND OPERATIONS ▼

<< 10.2017 >>

Klaus Bild	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Philipp Pfister	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Zoran Mitrovic	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Daniel Briner	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Antonio Casu	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Consolidate many microservices into single API calls?

We could combine all info into one single REST call.

- May be ok for a small app like this

- In larger apps quickly ends in countless special cases

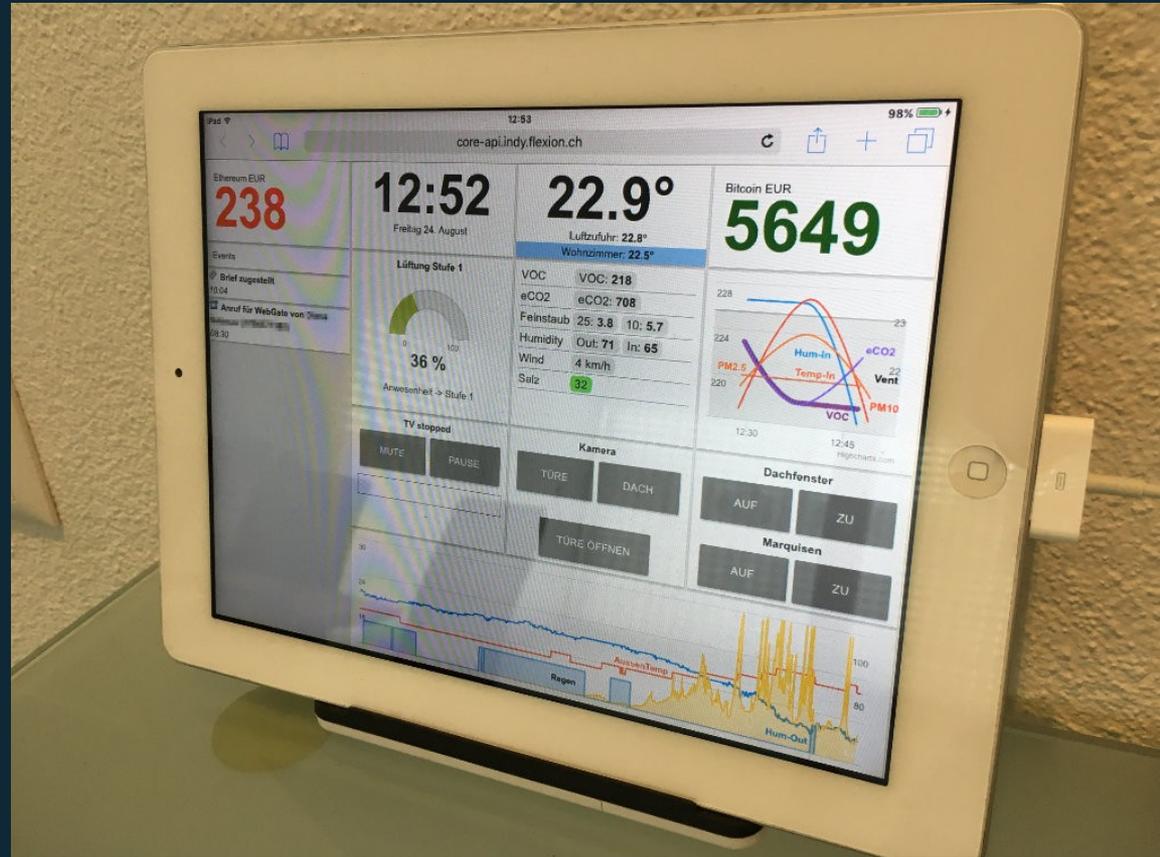
And of course requires programming and maintaining..

REST APIs for real-time data?

If you want to build apps displaying real-time data, REST calls will always show delayed data!

With REST you have to poll for updates every n minutes.

Not relevant for our absence app, but may be useful in our shift planer app, a webshop ordering queue or sensor dashboards!





WebSockets

- Good for real-time data
- Permanent bidirectional connection

Looks like we need a constantly running server, as replacement for Lambda API-GW?

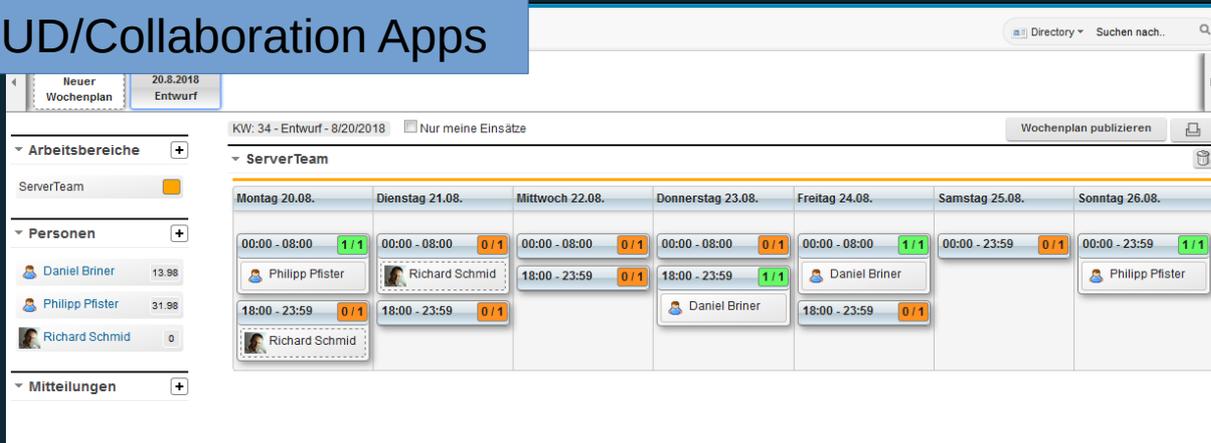
Goodbye serverless?

Sensor Dashboard



AWS IoT Hub

CRUD/Collaboration Apps



AWS AppSync



AWS AppSync Serverless GraphQL Service

- GraphQL query APIs (HTTP POST)
- WebSockets for subscription to real-time updates
- Even has offline-support!
- Seamlessly integrated into AWS serverless goodies:
DynamoDB, Lambda, Cognito..

What is GraphQL?

GraphQL

From Wikipedia, the free encyclopedia

GraphQL is an open source data [query](#) and [manipulation](#) language, and a runtime for fulfilling queries with existing data.^[2] GraphQL was developed internally by [Facebook](#) in 2012 before being publicly released in 2015.^[3] It provides a more efficient, powerful and flexible alternative to [REST](#) and ad-hoc [web service](#) architectures.^{[2][4]} It allows clients to define the structure of the data required, and exactly the same structure of the data is returned from the server, therefore preventing excessively large amounts of data from being returned.

GraphQL supports reading, writing (mutating) and subscribing to changes to data (realtime updates).^[5]

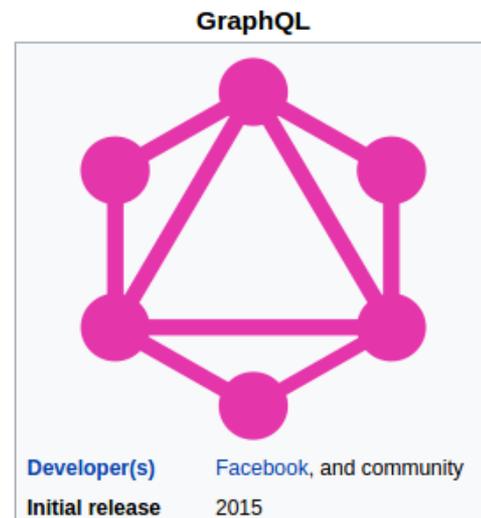
Major GraphQL clients include Apollo Client^[6] and Relay.^[7] GraphQL servers are available for multiple languages, including Haskell, JavaScript, Python,^[8] Ruby, Java, C#, Scala, Go, Elixir,^[9] Erlang, PHP, R, and Clojure.

On the 9th February 2018, the GraphQL Schema Definition Language (SDL) was made part of the specification.^[10]

See also [\[edit \]](#)

- [Query by Example](#)

References [\[edit \]](#)



- Developed by Facebook, made public in 2015, SDL added in 2018.
- Allows clients to define structure of required data, and get exactly that
- GraphQL client libraries available for many languages (Apollo)

GraphQL – Query Language for APIs



Describe your data

```
type Project {  
  name: String  
  tagline: String  
  contributors: [User]  
}
```

Ask for what you want

```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

Get predictable results

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```

- Ask for what you need, get exactly that
- Get many resources in a single request
- Single API endpoint for everything
- Describe what's possible with a type system
- Evolve API without versions

<https://graphql.org>

Ask for what you need and get exactly that

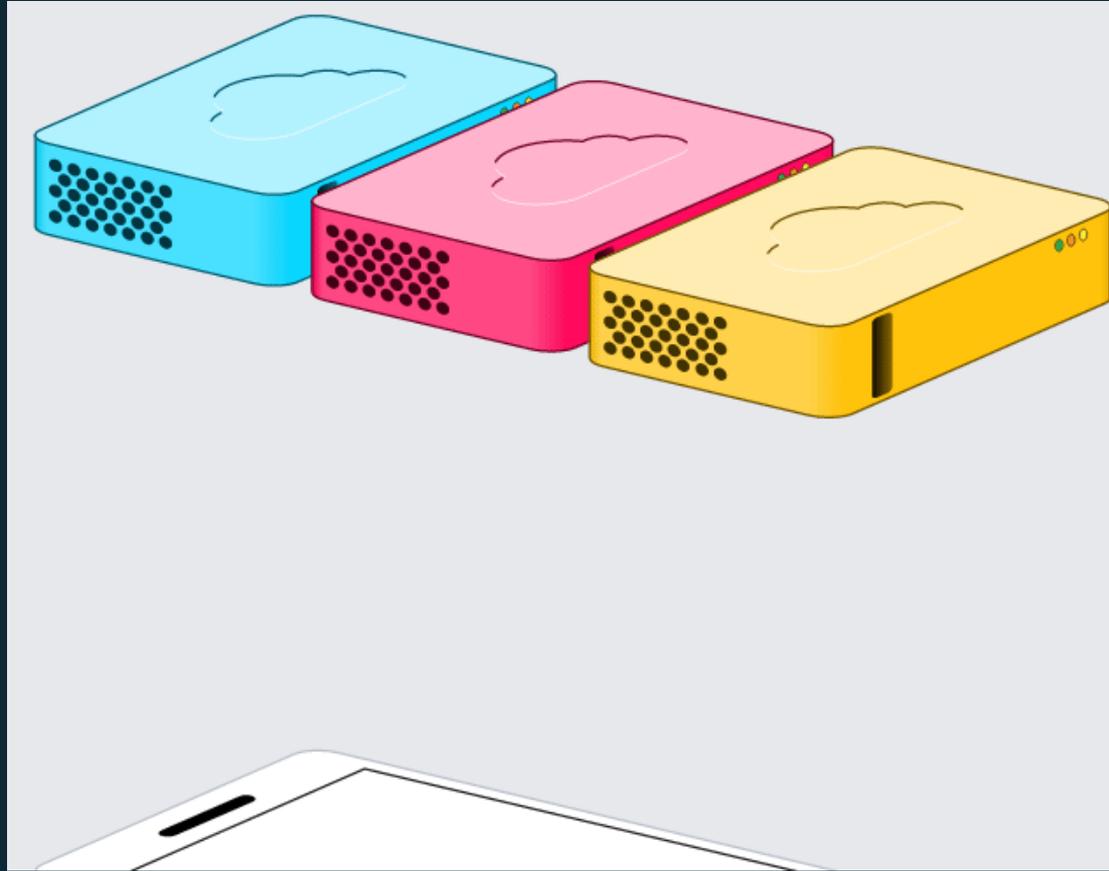
Request:

```
{
  hero {
    name
  }
}
```

Response:

```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

Get many resources in a single request



Describe what's possible with a type system

```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

```
type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

→ Did you notice? You can now build relations in DynamoDB without coding!

HOW COOL IS THAT?!?!1!

GraphQL API interactions

Query HTTP POST

```
query HeroNameAndFriends {  
  hero {  
    name  
    friends {  
      name  
    }  
  }  
}
```

Mutation HTTP POST

```
mutation {  
  createTask(title:"Cleanup"){  
    id  
  }  
}
```

Subscription WebSocket

```
subscription {  
  onCreateComment{  
    id  
  }  
}
```

Let's see this in action

Event Demo WebApp in React
with real-time comment updates

User A

[Back to events](#)

AWSomeday Zurich

September 13, 2018

9:00 AM

Digicomp Zurich

Welcome to the AWSomeday in Zurich!

Comments

[Add Comment](#)

User B

React App | AWS AppSync Cons | Dynam

localhost:3000/event/8f8ddfd4-d825-4ed3-9e69-df7d4ce

[Back to events](#)

AWSomeday Zurich

September 13, 2018

9:00 AM

Digicomp Zurich

Welcome to the AWSomeday in Zurich!

Comments

[Add Comment](#)

AWSomeday Zurich

📅 September 13, 2018

🕒 9:00 AM

📍 Digicomp Zurich

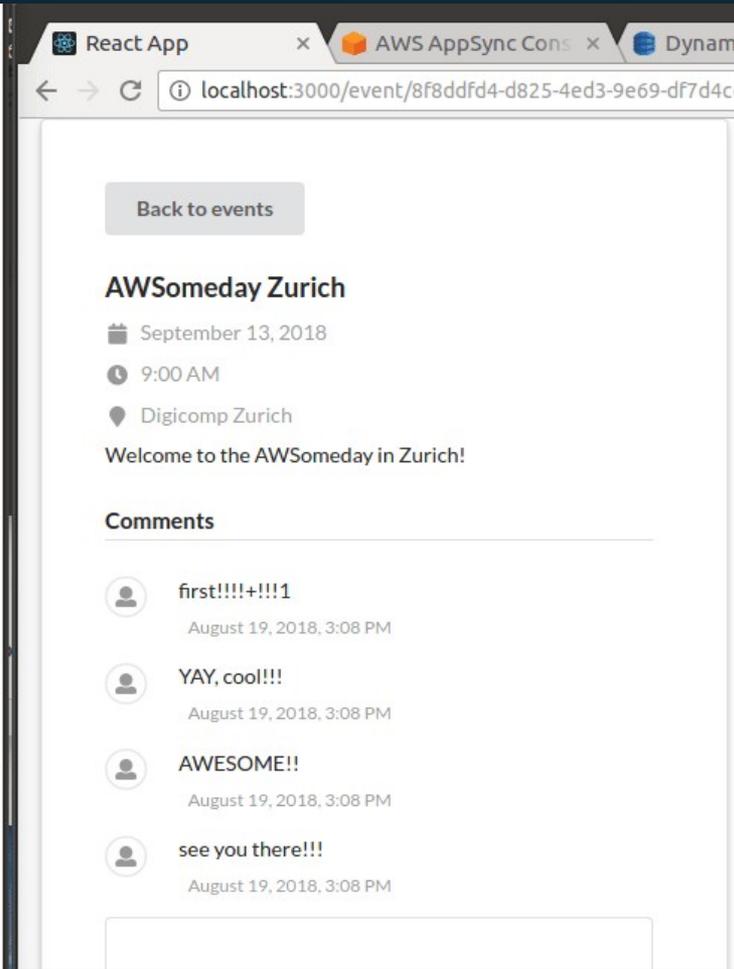
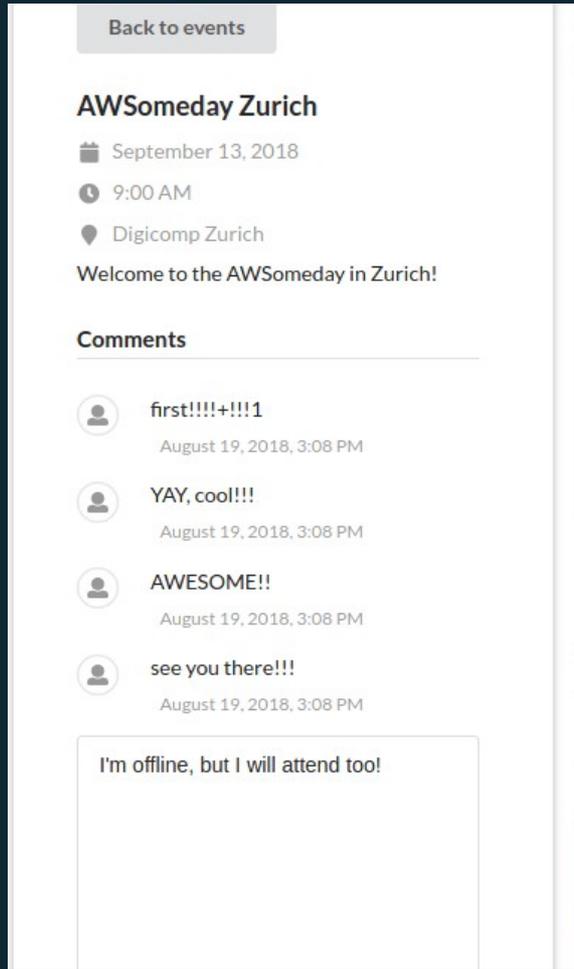
Welcome to the AWSomeday in Zurich!

Comments

Name	Method	Status	Type	Initiator
<input type="checkbox"/> graphql	OPTIONS	200	fetch	httpLink.js:83
<input type="checkbox"/> graphql	OPTIONS	200	fetch	httpLink.js:83
<input type="checkbox"/> graphql	POST	200	json	Other
<input type="checkbox"/> graphql	POST	200	json	Other
<input type="checkbox"/> mqtt?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=ASIAS...	GET	101	websocket	gaho-mqtt.js:1050

Offline interaction

UI is updated immediately, but request remains in the queue



Optimistic UI

- optimisticReponse in mutate function

Update the UI immediately, instead of waiting for server response.

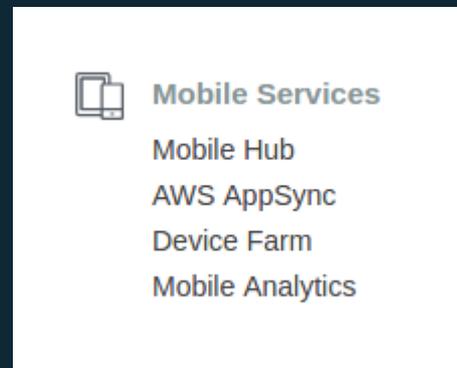
Not just for offline mode.

App appears to be faster for the user!

```
@Component({ ... })
class AppComponent {
  submit({ commentId, commentContent }) {
    this.apollo.mutate({
      variables: { commentId, commentContent },
      optimisticResponse: {
        __typename: 'Mutation',
        updateComment: {
          id: commentId,
          __typename: 'Comment',
          content: commentContent,
        },
      },
    }).subscribe();
  }
}
```

Create a new AWS AppSync API

- Open the AWS web console in a supported region and navigate to “Mobile Services > AppSync”



Create API

Author from scratch

Start with a blank template to build a GraphQL app with your custom schema.

Chat App

Learn how to use AWS AppSync subscriptions to enable real-time chat between users.

Blog App

Learn how to use AWS Lambda to read and write to Amazon RDS in a Blog app

Event App

Learn to build a basic AWS AppSync app for creating calendar events with user comments.

Tasks App

Learn how to use groups authorization with Cognito User Pools to build a Task app.

API configuration

API name

Type a name for your AWS AppSync API.

There are sample client apps available for:

- Angular (Web)
- React (Web)
- React Native
- iOS
- Android

Selecting a demo app will create required resources in the background (DynamoDB tables etc)



AWS AppSync



APIs

Event-Demo-App

Schema

Queries

Data Sources

Settings

AWS AppSync > Event-Demo-App

Event-Demo-App

API Details

API URL

Copy

API ID

Copy

Auth mode

API KEY

Copy

API keys by default expire 7 days after creation. Configure API keys in [Settings](#).

Getting Started

Welcome! AWS AppSync is a managed GraphQL service that allows web and mobile developers to create powerful data driven applications on top of AWS. Follow these steps to deploy a GraphQL API and integrate it into your application.

Design your schema

Settings

Configure your API and change authorization strategies. [Info](#)

Edit API name

Edit API name

Enter a name for your API.

Authorization type

- API key
- AWS Identity and Access Management (IAM)
- Amazon Cognito User Pool
- OpenID Connect

For webapps you will most probably use cognito

API keys

[Delete](#)[Edit](#)[New](#)

AWS AppSync



APIs

Event-Demo-App

Schema

Queries

Data Sources

Settings

[AWS AppSync](#) > [Event-Demo-App](#) > Data Sources

Data Sources

Connect existing AWS resources to your API. [Info](#)

Data Sources

Delete

Edit

New



1



	Name ▲	Type ▼	Resource ▼
<input type="radio"/>	AppSyncCommentTable	AMAZON_DYNAMODB	AppSyncCommentTable-NqcjATND
<input type="radio"/>	AppSyncEventTable	AMAZON_DYNAMODB	AppSyncEventTable-NqcjATND

New Data Source

Create new Data Source

Data source name

A name starts with letter and contains only numbers, letters and "_"

Data source type

Select the type of your data source.

	▼
Amazon DynamoDB table	
Amazon Elasticsearch domain	
AWS Lambda function	
HTTP endpoint	
None	

Cancel

Create

DynamoDB table as data source without Lambda

Data source type
Select the type of your data source.

Amazon DynamoDB table ▼

Region
Select the region that contains your data source.

EU-CENTRAL-1 ▼

Table name
Select a table from the dropdown.

wgc-employees ▼

[Don't see your table?](#)

Create or use an existing role
Allow AWS AppSync to securely interact with your data source.

New role

Existing role

Automatically generate GraphQL
Turning this on will extend your existing schema and automatically configure resolvers.



GraphQL schema automatically created from existing DynamoDB table data!

Or you can write your Schema first and click the “Create Resources” button.

Schema

Design your schema using GraphQL SDL, attach resolvers, and quickly deploy AWS resources. [Info](#)

```
Schema Export schema ▾
48 type EmployeeConnection {
49   items: [Employee]
50   nextToken: String
51 }
52
53 type Mutation {
54   createEmployee(input: CreateEmployeeInput!): Employee
55   updateEmployee(input: UpdateEmployeeInput!): Employee
56   deleteEmployee(input: DeleteEmployeeInput!): Employee
57 }
58
59 type Query {
60   getEmployee(username: String!): Employee
61   listEmployees(filter: TableEmployeeFilterInput, limit: Int, nextToken: String): [Employee]
62   queryEmployeesByShortnameIndex(shortname: String!, first: Int, nextToken: String): [Employee]
63 }
64
65 type Subscription {
66   onCreateEmployee(
67     username: String,
68     shortname: String,
69     department: String,
70     manager: String,
71     officephone: String
72   ): Employee
```

Another schema example

Access control



Mandatory fields



```
1 schema {
2   query: Query
3   mutation: Mutation
4 }
5
6 type Mutation {
7   # In this example, only users in the ManagerGroup can create tasks
8   createTask(
9     owner: String!,
10    title: String!,
11    taskStatus: String!,
12    description: String!
13  ): Task
14   @aws_auth(cognito_groups: ["ManagerGroup"])
15   # Both Employees and Managers can update a task's status
16   updateTaskStatus(id: ID!, taskStatus: String!): Task
17   @aws_auth(cognito_groups: ["EmployeeGroup", "ManagerGroup"])
18   updateTaskBody(id: ID!, title: String!, description: String!): Task
19   @aws_auth(cognito_groups: ["ManagerGroup"])
20 }
21
22 type Query {
23   # Users belonging to both EmployeesGroup and ManagerGroup can read a particular task
24   getTask(id: ID!): Task
25   @aws_auth(cognito_groups: ["EmployeeGroup", "ManagerGroup"])
26   # Only Managers can list all the Tasks
27   allTasks(nextToken: String): TaskConnection
28   @aws_auth(cognito_groups: ["ManagerGroup"])
29 }
30
31 type Task {
32   id: ID!
33   owner: String!
34   title: String!
35   description: String!
36   taskStatus: String
37 }
38
39 type TaskConnection {
40   items: [Task]
41   nextToken: String
42 }
```

Attach resolvers to create relations

Schema Export schema ▼

```
1 type AbsenceRequest {
2   id: String!
3   requestfor: String
4   reviewer: String
5   status: String
6   title: String
7 }
8
9 type AbsenceRequestConnection {
10  items: [AbsenceRequest]
11  nextToken: String
12 }
13
14 input CreateAbsenceRequestInput {
15  id: String!
16  requestfor: String
17  reviewer: String
18  status: String
19  title: String
20 }
21
22 input CreateEmployeeInput {
23  username: String!
```

Resolvers

Filter types...

AbsenceRequest

Field	Resolver
id: String!	<input type="button" value="Attach"/>
requestfor: String	<input type="button" value="Attach"/>
reviewer: String	<input type="button" value="Attach"/>
status: String	<input type="button" value="Attach"/>
title: String	<input type="button" value="Attach"/>

Resolver for AbsenceRequest.requestfor

Data source name
Select the data source to resolve.

employee_table

Configure the request mapping template.

Translate a GraphQL query into a format specific to your data source. [Info](#)

```
1 ## Below example shows how to look up an item with a Primary Key of "id" from GraphQ
2 ## The helper $util.dynamodb.toDynamoDBJson automatically converts to a DynamoDB for
3 ## There is a "context" object with arguments, identity, headers, and parent field i
4 ## It also has a shorthand notation available:
5 ## - $context or $ctx is the root object
6 ## - $ctx.arguments or $ctx.args contains arguments
7 ## - $ctx.identity has caller information, such as $ctx.identity.username
8 ## - $ctx.request.headers contains headers, such as $context.request.headers.xyz
9 ## - $ctx.source is a map of the parent field, for instance $ctx.source.xyz
10 ## Read more: https://docs.aws.amazon.com/appsync/latest/devguide/resolver-mapping-t
11
12 {
13   "version": "2017-02-28",
14   "operation": "GetItem",
15   "key": {
16     "id": $util.dynamodb.toDynamoDBJson($ctx.args.id),
17   }
18 }
```

Configure the response mapping template.

Translate the results back to GraphQL. [Info](#)

```
1 ## Pass back the result from DynamoDB. **
2 $util.toJson($ctx.result)
```

ACL using Resolver Mapping Templates

Apache Velocity Template Language (VTL)

Allow owner of document to read (based on field value in dynamoDB)

```
#if($context.result["Owner"] == $context.identity.username)
    $utils.toJson($context.result);
#else
    $utils.unauthorized()
#end
```

Query testing tool

AWS AppSync



APIs

AppSync Demo App

Schema

Queries

Data Sources

Settings

AWS AppSync > AppSync Demo App > Queries

Queries

Write, validate, and test GraphQL queries. [Info](#)



< Docs

```
1 query list {  
2   listEmployees (limit: 5) {  
3     items {  
4       shortname  
5       username  
6       department  
7       manager  
8     }  
9   }  
10 }  
11  
12  
13  
14  
15  
16  
17  
18
```

```
{  
  "data": {  
    "listEmployees": {  
      "items": [  
        {  
          "shortname": "SBA",  
          "username": "Samuel Baeumlin",  
          "department": "Development",  
          "manager": "Dirk Apel"  
        },  
        {  
          "shortname": "CGU",  
          "username": "Christian Guedemann",  
          "department": "IRS",  
          "manager": "Roman Weber"  
        },  
        {  
          "shortname": "BHE"
```

Code samples: Query by ID

```
import React, { Component } from "react";
import { graphql } from "react-apollo";
import QueryGetEvent from "../GraphQL/QueryGetEvent";

const ViewEventWithData = graphql(
  QueryGetEvent,
  {
    options: ({ match: { params: { id } } }) => ({
      variables: { id },
      fetchPolicy: 'cache-and-network',
    }),
    props: ({ data: { getEvent: event, loading } }) => ({
      event,
      loading,
    }),
  },
)(ViewEvent);

class ViewEvent extends Component {

  render() {
    const { event, loading } = this.props;
```

```
JS QueryGetEvent.js x
1 import gql from "graphql-tag";
2
3 export default gql(`
4 query($id: ID!) {
5   getEvent(id: $id) {
6     id
7     name
8     where
9     when
10    description
11    comments {
12      __typename
13      items {
14        commentId
15        content
16        createdAt
17      }
18    }
19  }
20 `);
```

Automatic conflict resolution in the cloud

Useful when using offline mode

- Client wins
- Server wins
- Silent reject
- Custom Lambda Fn

AppSync vs. API-GW+Lambda

- Almost no backend coding required
- Easy to implement access control, even on field level
- Build object relations, without coding
- Object related definition, type-safe programming
- Real time subscriptions to events
- Offline support incl. conflict resolution (useful for mobile webapps)
- Perfect for CRUD. API-GW+Lambda for complex tasks.

Pricing

Query and Data Modification Operations

- AppSync: \$4 per million Query and Data Modification Operations

(API-GW: \$3.50 per million API calls received, plus the cost of data transfer out, plus lambda execution time!)

Real-time Updates

- \$2 per million Real-time Updates
- \$0.08 per million minutes of connection to the AWS AppSync service

Integrate your GraphQL API

Once you have deployed your API, easily add it to your application with one of our powerful SDKs or using popular tools such as Apollo Client and Relay.

iOS

Android

Web (React)

Web (Angular)

React Native

1. First clone this repo:

```
git clone https://github.com/aws-samples/aws-mobile-appsync-chat-starter-angular
```

2. Download the AWS AppSync.js config file:

Download

3. Download the graphql schema:

Export schema ▼

Explore the demos!

Angular2 Demo: <https://github.com/aws-samples/aws-mobile-appsync-chat-starter-angular>

React Demo: <https://github.com/aws-samples/aws-mobile-appsync-events-starter-react>

BTW: You can also use NodeJS to connect to AppSync!

Documentation: <https://docs.aws.amazon.com/appsync/latest/devguide/welcome.html>

Thanks for attending this AWSome Day session

Join the AWS Swiss User Group on meetup.com
for more AWS technical sessions,
or follow our blog.



richard.schmid @ webgate.biz
blog.webgate.biz

«Mit unserem Innovation Lab den Wandel gestalten»



**Innovationen
Etablieren**

Innovation ist bei uns nicht nur ein Buzzword



«Mit uns holen Sie das Optimum aus Ihren Informationen heraus»

**Inhalte
Vitalisieren**

Inhalte vitalisieren, Kunden überzeugen



«Wir machen Sie konkurrenzfähig im Zeitalter der Digitalisierung»

**Prozesse
Digitalisieren**

Wettbewerbsvorteile sichern und ausbauen