Not your Fathers Java

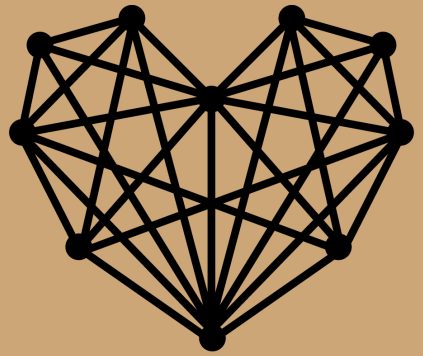
GraphQL Servers with Kotlin and GraphQL-Java

goto; berlin

Michael Hunger

Head of Developer Relations
neo4j.com/developer
"Spreading GraphLove since 2008"

Follow me: @mesirii

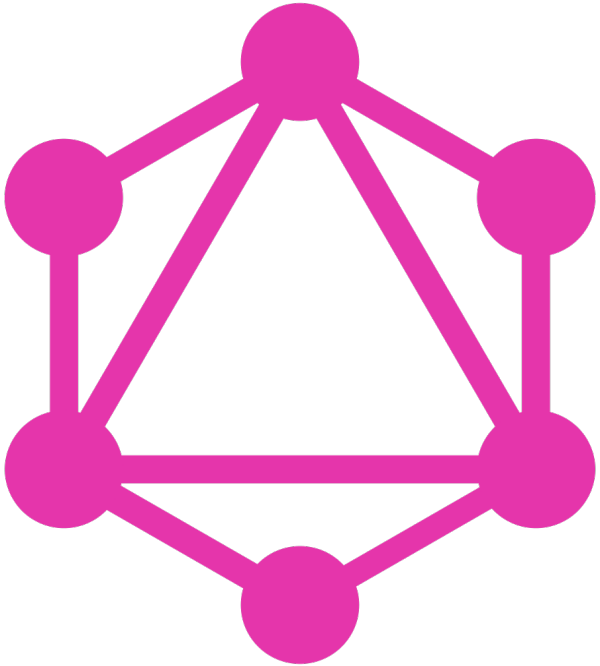


Graphlove

PSA 1

PSA 2

What is GraphQL?



What is GraphQL?

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"
  }
}
```

A schema based query language for your APIs

GraphQL Schema

GraphQL Query

Ask for what you want

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

Get predictable results

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

Top Level, Parameterized Queries & Drill Down

Demo – Movie Database

movies.grandstack.io

GraphQL Mutations

```
type Mutations {
  newProject(name: String, tagline: String): Project
  addContributors(name: String, contributor: String): Project
}
schema {
  mutation: Mutations
}
```

Update Commands that can return data

GraphQL Subscriptions

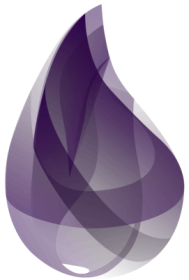
```
type Subscriptions {
  projectUpdate (projectName: String!): Project
}
schema {subscription: Subscriptions }
subscription($name: String!) {
   projectUpdate (projectName: $name) {
      tagline, contributors { name }
}}
```

Live Updates on Changes

GraphQL

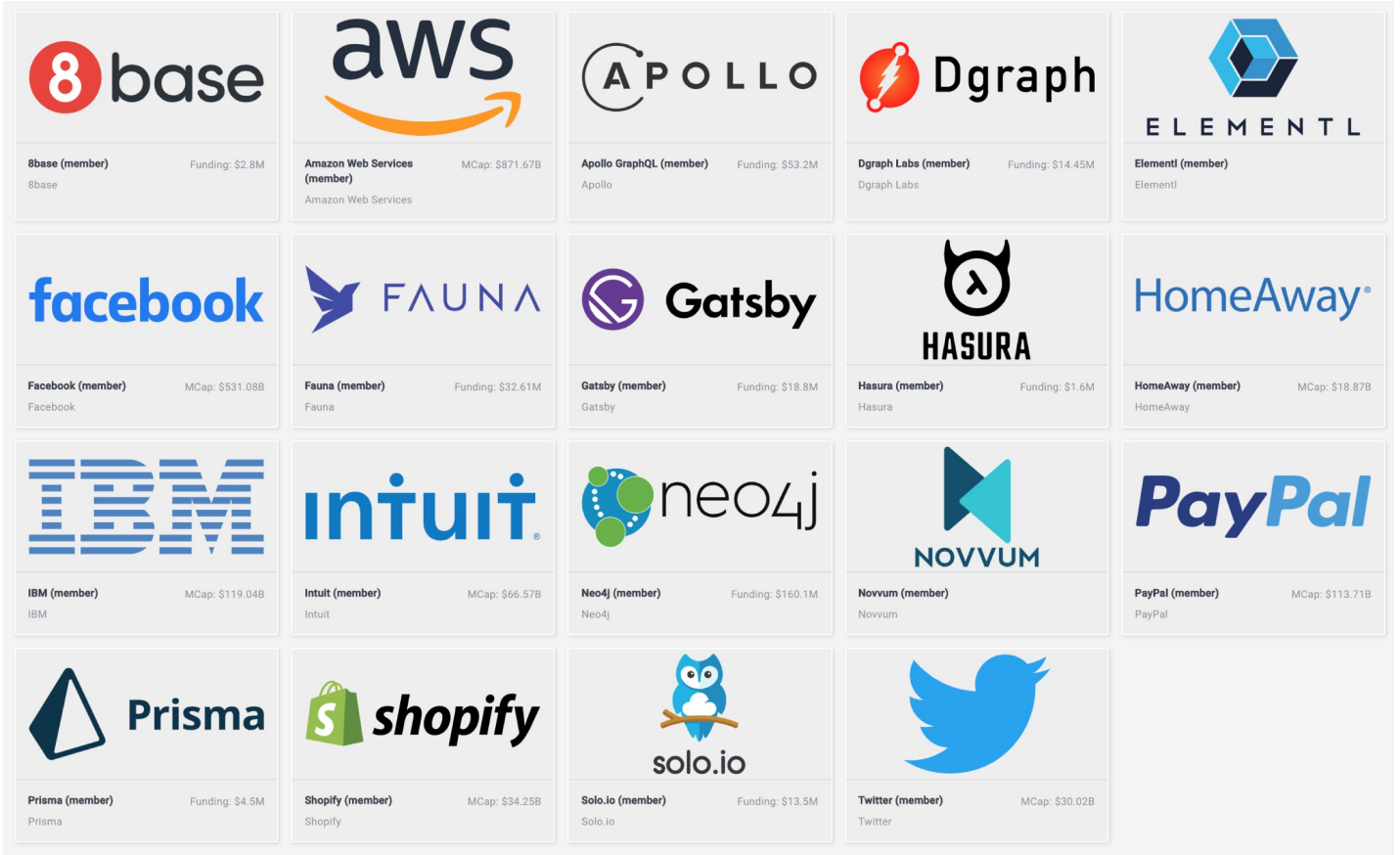
- by Facebook for mobile apps, open-sourced in 2015
- GraphQL working group
- Reference Implementation in JavaScript but many language impls.
- now GraphQL Foundation
- Schema is the ***joint language*** for front-end and back-end
- Self Introspection & very helpful tooling
- Shape your request by data needs of front-end
- DDD and GraphQL
- GraphQL vs. REST

Languages

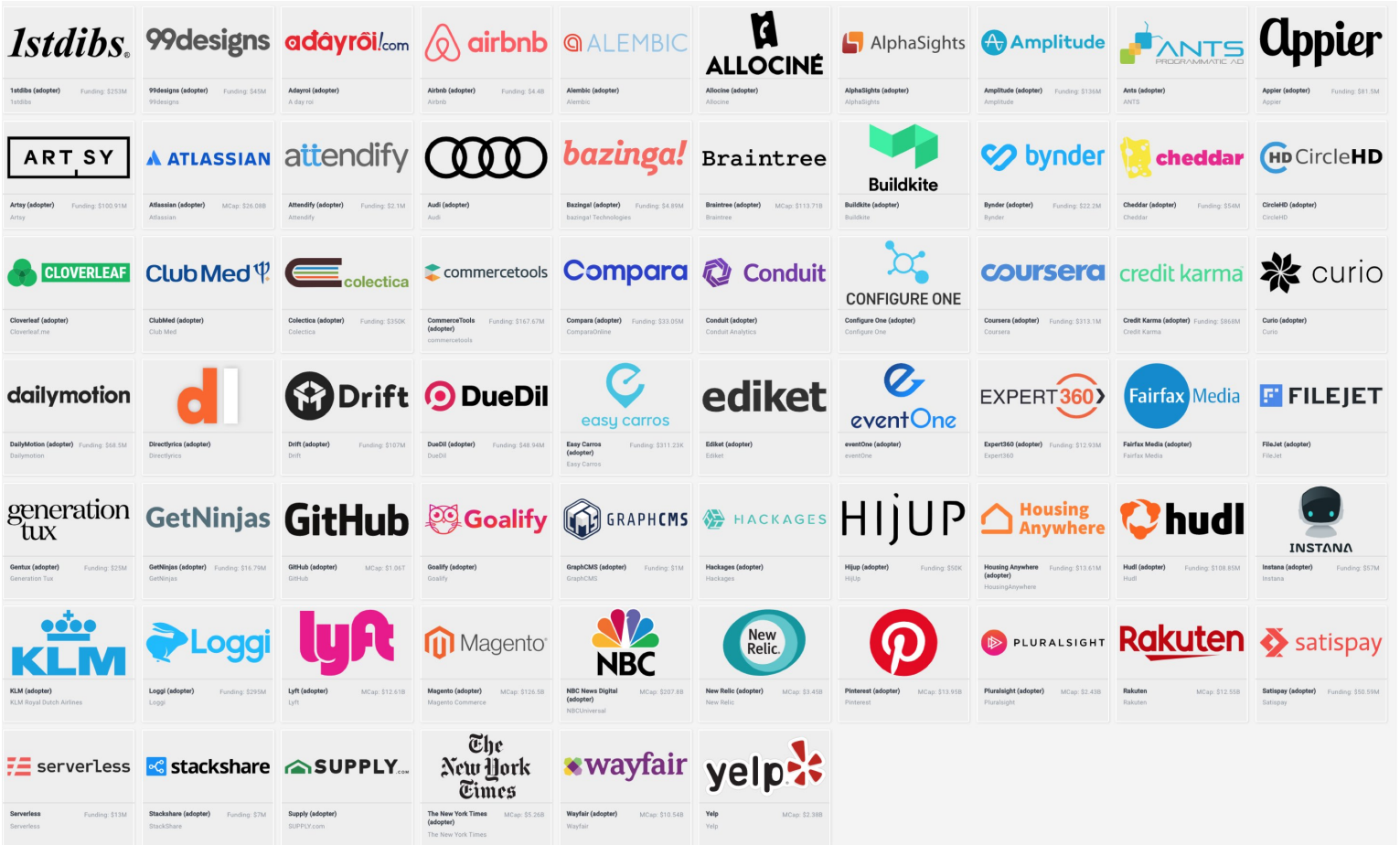


graphql.org/code/

Foundation


8base
8base (member)
Funding: $2.8M
8base
aws
Amazon Web Services (member)
MCap: $871.67B
Amazon Web Services
APOLLO
Apollo GraphQL (member)
Funding: $53.2M
Apollo
Dgraph
Dgraph Labs (member)
Funding: $14.45M
Dgraph Labs
ELEMENTL
Elementl (member)
Elementl
facebook
Facebook (member)
MCap: $531.08B
Facebook
FAUNA
Fauna (member)
Funding: $32.61M
Fauna
Gatsby
Gatsby (member)
Funding: $18.8M
Gatsby
HASURA
Hasura (member)
Funding: $1.6M
Hasura
HomeAway
HomeAway (member)
MCap: $18.87B
HomeAway
IBM
IBM (member)
MCap: $119.04B
IBM
intuit
Intuit (member)
MCap: $66.57B
Intuit
neo4j
Neo4j (member)
Funding: $160.1M
Neo4j
NOVVUM
Novvum (member)
Novvum
PayPal
PayPal (member)
MCap: $113.71B
PayPal
Prisma
Prisma (member)
Funding: $4.5M
Prisma
shopify
Shopify (member)
MCap: $34.25B
Shopify
solo.io
Solo.io (member)
Funding: $13.5M
Solo.io
Twitter (member)
MCap: $30.02B
Twitter

Adopters


1stdibs
99designs
adayroi.com
airbnb
ALEMBIC
ALLOCINÉ
AlphaSights
Amplitude
ANTS PROGRAMMATIC AD
Appier
ARTSY
ATLASSIAN
attendify
bazinga!
Braintree
Buildkite
bynder
cheddar
CircleHD
CLOVERLEAF
Club Med
colectica
commercetools
Compara
Conduit
CONFIGURE ONE
coursera
credit karma
curio
dailymotion
Drift
DueDil
easy carros
ediket
eventOne
EXPERT360
Fairfax Media
FILEJET
generation tux
GetNinjas
GitHub
Goalify
GRAPHCMS
HACKAGES
HIJUP
Housing Anywhere
hudl
INSTANA
KLM
Loggi
lyft
Magento
NBC
New Relic
PLURALSIGHT
Rakuten
satispay
serverless
stackshare
SUPPLY
The New York Times
wayfair
yelp

GraphQL vs. REST

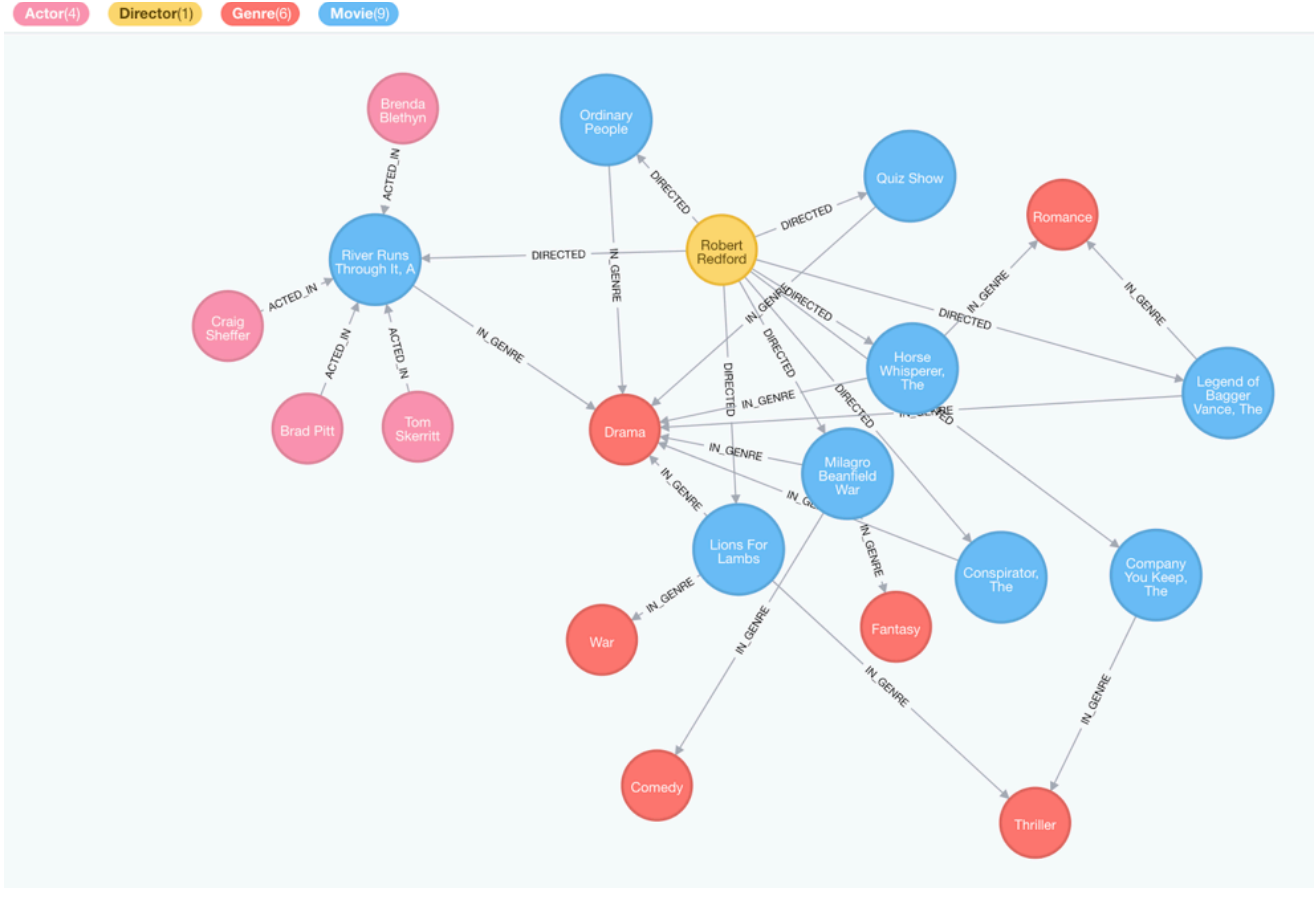
REST

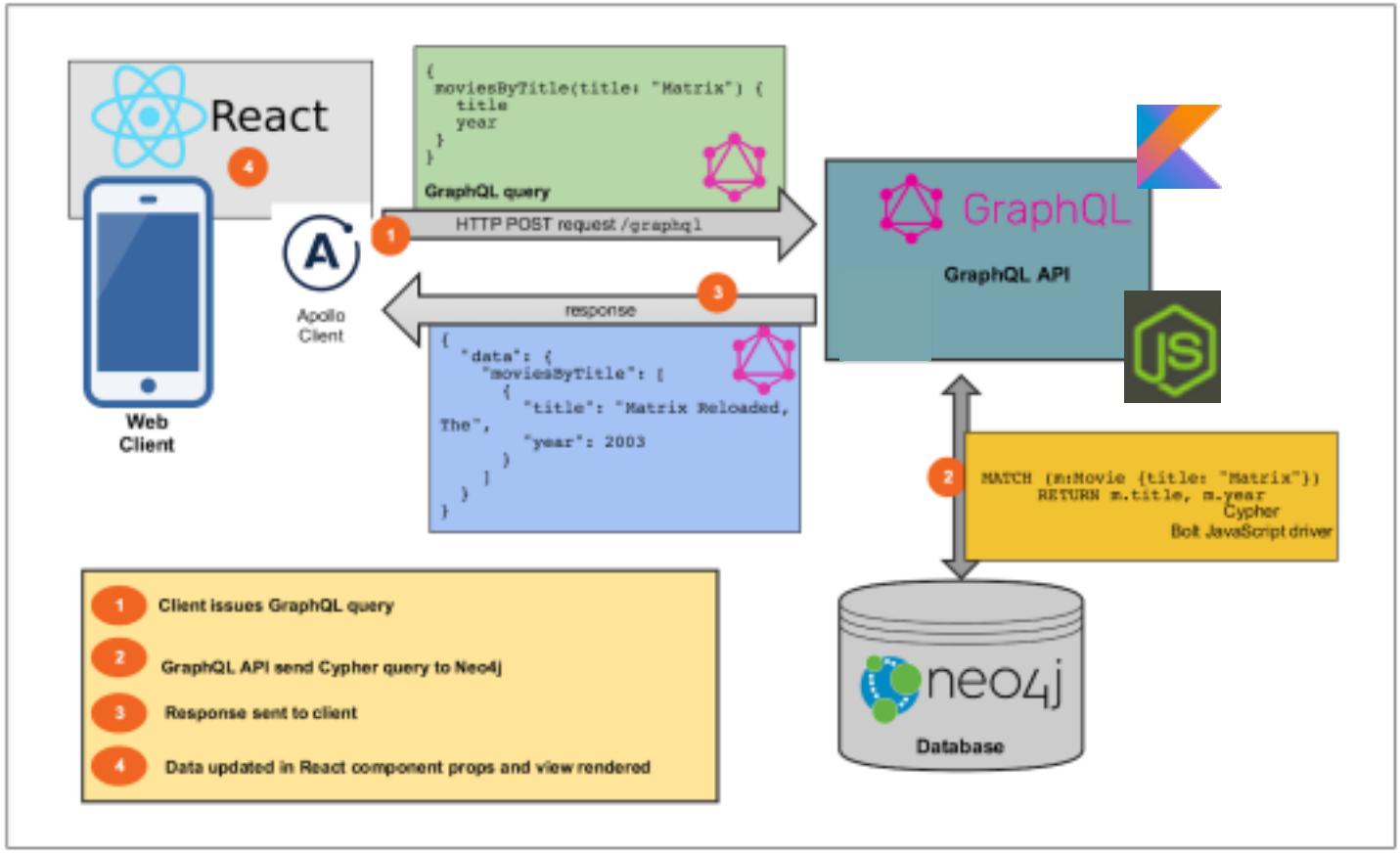
- One HTTP endpoint per Resource
- HTTP middleware/headers
- Send HTTP Request
- External API (Swagger, RAML)
- Client Navigation via HATEOAS
- Possibly n+1 requests

GraphQL

- One endpoint
- Schema driven development
- Send typed query, mutation
- Subscribe to subscription updates
- GraphQL specific, schema aware middleware
- No prescription on transport layer (often HTTP)

Your Application Data is a Graph



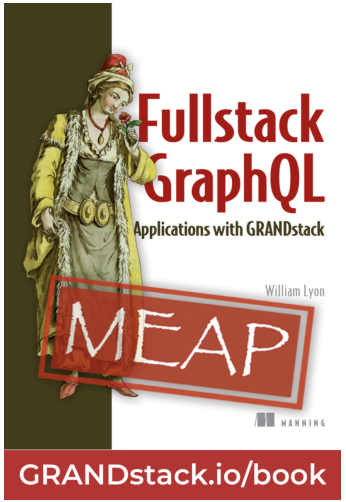

React
4
Web Client
Apollo Client
{
moviesByTitle(title: "Matrix") {
title
year
}
}
GraphQL query
1
HTTP POST request /graphql
3
response
{
"data": {
"moviesByTitle": [
{
"title": "Matrix Reloaded, The",
"year": 2003
}
]
}
}
GraphQL
GraphQL API
2
MATCH (m:Movie {title: "Matrix"})
RETURN m.title, m.year
Cypher
Bolt JavaScript driver
neo4j
Database
1 Client issues GraphQL query
2 GraphQL API send Cypher query to Neo4j
3 Response sent to client
4 Data updated in React component props and view rendered

GrandStack.io

William Lyon
@lyonwj

🚨 I'm writing a book! 🚨

Fullstack GraphQL published by @ManningBooks



GRANDstack.io/book

It brings together everything I've learned while working on #GraphQL integrations w/ @neo4j over the last few years

The early release is out today, use "friendofwill40" for 40% off

Building a GraphQL Server

How hard can it be?

A GraphQL-Server on the JVM in 33 Lines

Andy Marek's GraphQL-Java 13

easy to use
spec compatible
async APIs

github.com/graphql-java/awesome-graphql-java

GraphQL-Java Hello-World

Minimal Example - Schema

```
type Query {
  hello(what:String):String
}
```

Minimal Example - Query

```
query($p:String) {
    hello(what: $p)
}
```

parameters:

```
{"p" : "Berlin"}
```

response:

```
{"hello" : "Hello Berlin!"}
```

GraphQL-Java


```
public class HelloWorld {

   public static void main(String[] args) {
     String schema = "type Query { " +
                     " hello(what:String = \"World\"): String " +
                     " }";

     SchemaParser schemaParser = new SchemaParser();
     TypeDefinitionRegistry registry = schemaParser.parse(schema);

     RuntimeWiring runtimeWiring = newRuntimeWiring()
              .type("Query", builder -> builder.dataFetcher("hello",
                                 env -> "Hello "+env.getArgument("what")+"!"))
              .build();

     SchemaGenerator schemaGenerator = new SchemaGenerator();
     GraphQLSchema graphQLSchema = schemaGenerator.makeExecutableSchema(registry, runtimeWiring);

     GraphQL graphql = GraphQL.newGraphQL(graphQLSchema).build();
     ExecutionResult executionResult = graphql.execute("{hello(what:\"Java\")}");

     System.out.println(executionResult.<Map<String,Object>>getData());
     // Prints: {hello=Hello Java!}
}}
```

GraphQL-Java - Schema


```
public class HelloWorld {

   public static void main(String[] args) {
     String schema = "type Query { " +
                     " hello(what:String = \"World\"): String " +
                     " }";

     SchemaParser schemaParser = new SchemaParser();
     TypeDefinitionRegistry registry = schemaParser.parse(schema);
```

GraphQL-Java – Data Fetching


```
RuntimeWiring runtimeWiring = newRuntimeWiring()
        .type("Query",
                builder -> builder.dataFetcher("hello",
                        env -> "Hello "+env.getArgument("what")+"!"))
                .build();

    SchemaGenerator schemaGenerator = new SchemaGenerator();
    GraphQLSchema graphQLSchema =
      schemaGenerator.makeExecutableSchema(registry, runtimeWiring);
```

http://graphql-java.readthedocs.io/en/latest/getting_started.html

GraphQL-Java – Execution


```
GraphQL graphql = GraphQL.newGraphQL(graphQLSchema).build();

ExecutionResult result= graphql.execute("{hello(what:\"Java\")}");

System.out.println(result.<Map<String,Object>>getData());
```

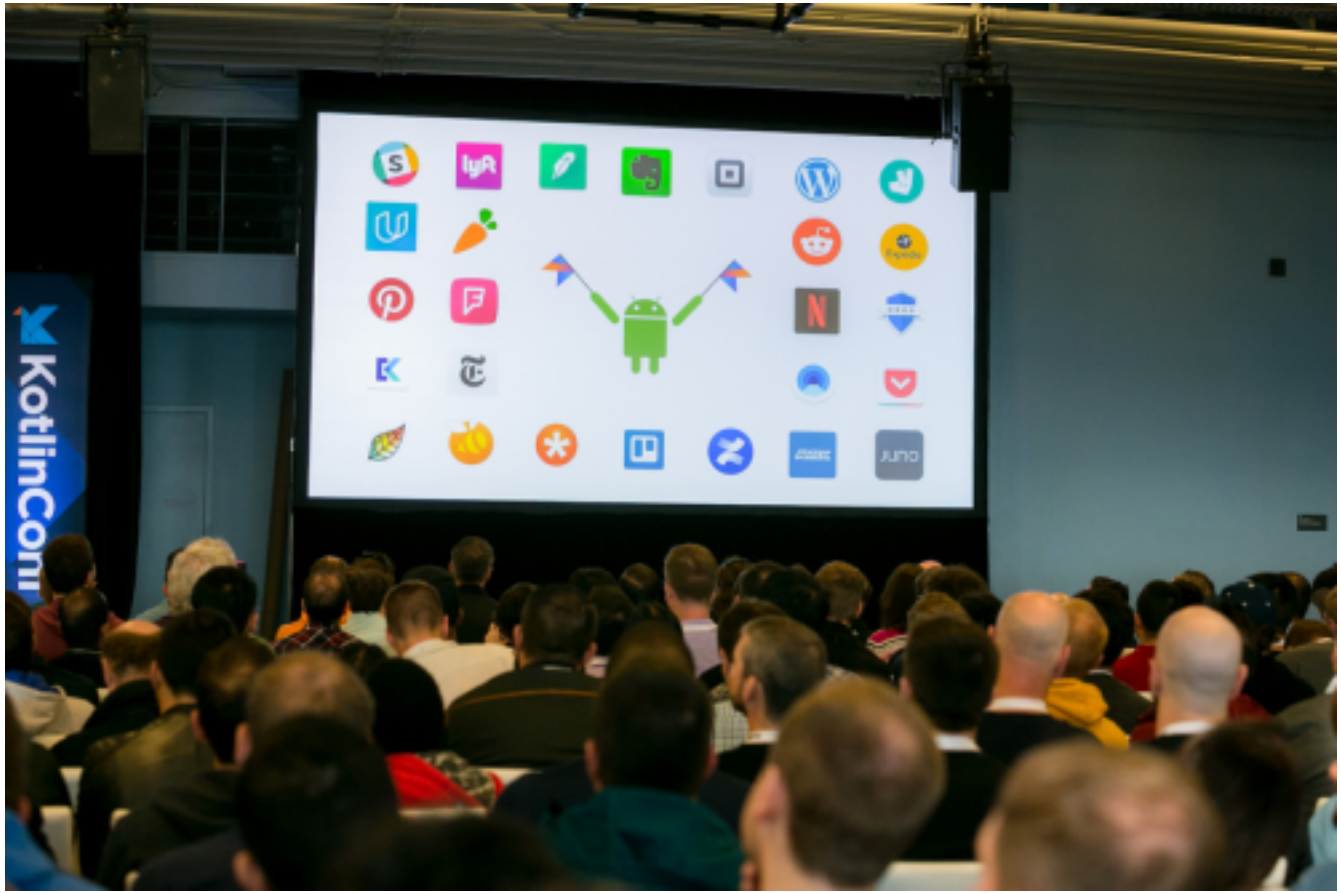
GraphQL-Java

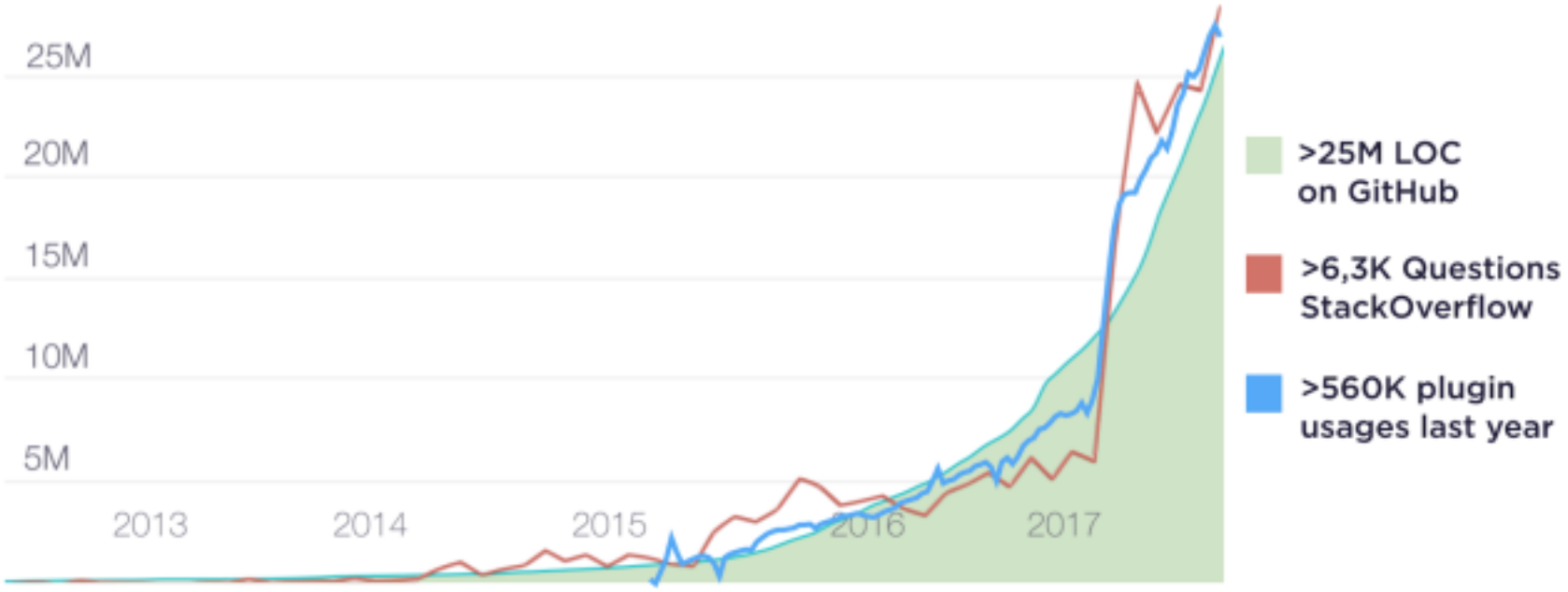

```
public class HelloWorld {

   public static void main(String[] args) {
     String schema = "type Query { " +
                     " hello(what:String = \"World\"): String " +
                     " }";

     SchemaParser schemaParser = new SchemaParser();
     TypeDefinitionRegistry registry = schemaParser.parse(schema);

     RuntimeWiring runtimeWiring = newRuntimeWiring()
              .type("Query", builder -> builder.dataFetcher("hello",
                                 env -> "Hello "+env.getArgument("what")+"!"))
              .build();

     SchemaGenerator schemaGenerator = new SchemaGenerator();
     GraphQLSchema graphQLSchema = schemaGenerator.makeExecutableSchema(registry, runtimeWiring);

     GraphQL build = GraphQL.newGraphQL(graphQLSchema).build();
     ExecutionResult executionResult = build.execute("{hello(what:\"Java\")}");

     System.out.println(executionResult.<Map<String,Object>>getData());
     // Prints: {hello=Hello Java!}
}}
```

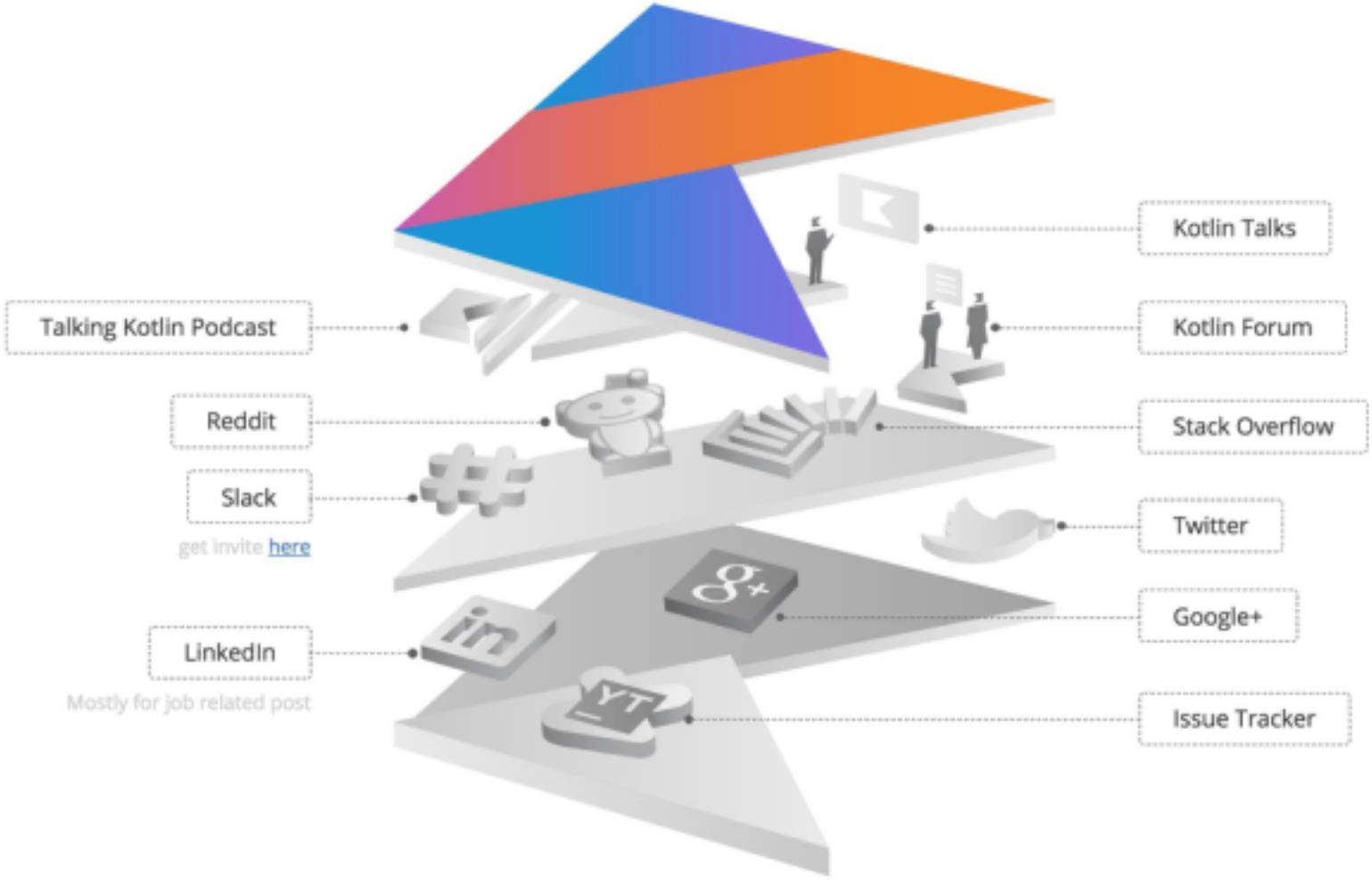
Kotlin

```
fun main(args: Array<String>) {
    println("Hello, World!")
}
```

```
val Int.odd : Boolean get() = this % 2 == 1

inline fun <R,reified T> Map<*,*>.aggregate(acc: R, operation: (R,T)->R) =
       values.filterIsInstance<T>().fold(acc, operation)

data class Word(val text:String, val len:Int = text.length)

fun main(args: Array<String>) {
   val message = """Hello ${args.firstOrNull() ?: "GotoCon"}
                  |in ${args.getOrNull(1) ?:"Berlin"} <3""".trimMargin()
   println(message)
   // Hello GotoCon in Berlin <3

   val words = message.split("\\s+".toRegex())
           .withIndex()
           .associate { (i,w) -> i to if (i.odd) w else Word(w)}
           .toSortedMap()

   println(words)
   // {0=Word(text=Hello, len=5), 1=GotoCon, 2=Word(text=in, len=2), 3=Berlin
   println(words.aggregate(1) {a,w:Word -> a*w.len})
   // 420
}
```


KotlinConf
Evernote
WordPress
Reddit
Pinterest
Pocket
Trello
Juno


25M
20M
15M
10M
5M
2013
2014
2015
2016
2017
>25M LOC
on GitHub
>6,3K Questions
StackOverflow
>560K plugin
usages last year


Kotlin Talks
Talking Kotlin Podcast
Kotlin Forum
Reddit
Stack Overflow
Slack
get invite here
Twitter
Google+
LinkedIn
Mostly for job related post
Issue Tracker

GraphQL-Kotlin

```
fun main(args: Array<String>) {
   val schema = """type Query {
                       hello(what:String = "World"): String
                   }"""

   val schemaParser = SchemaParser()
   val typeDefinitionRegistry = schemaParser.parse(schema)

   val runtimeWiring = newRuntimeWiring()
           .type("Query")
               { it.dataFetcher("hello") { env -> "Hello ${env.getArgument<Any>("what")}!" } }
           .build()

   val schemaGenerator = SchemaGenerator()
   val graphQLSchema = schemaGenerator.makeExecutableSchema(typeDefinitionRegistry, runtimeWiring)

   val build = GraphQL.newGraphQL(graphQLSchema).build()
   val executionResult = build.execute("""{ hello (what:"Kotlin") } """)

   println(executionResult.getData<Any>())
   // Prints: {hello=Hello Kotlin!}
}
```

GraphQL-Java


```
public class HelloWorld {

   public static void main(String[] args) {
     String schema = "type Query { " +
                     " hello(what:String = \"World\"): String " +
                     " }";

     SchemaParser schemaParser = new SchemaParser();
     TypeDefinitionRegistry registry = schemaParser.parse(schema);

     RuntimeWiring runtimeWiring = newRuntimeWiring()
              .type("Query", builder -> builder.dataFetcher("hello",
                                  env -> "Hello "+env.getArgument("what")+"!"))
              .build();

     SchemaGenerator schemaGenerator = new SchemaGenerator();
     GraphQLSchema graphQLSchema = schemaGenerator.makeExecutableSchema(registry, runtimeWiring);

     GraphQL graphql = GraphQL.newGraphQL(graphQLSchema).build();
     ExecutionResult executionResult = graphql.execute("{hello(what:\"Java\")}");

     System.out.println(executionResult.<Map<String,Object>>getData());
     // Prints: {hello=Hello Java!}
}}
```

GraphQL-Kotlin

```
fun main(args: Array<String>) {
    val schema = """type Query {
                        hello(what:String = "World"): String
                    }"""

    val runtimeWiring = newRuntimeWiring()
            .type("Query")
                { it.dataFetcher("hello") { env -> "Hello ${env.getArgument<Any>("what")}!" } }
            .build()

    val graphQLSchema = SchemaGenerator()
                            .makeExecutableSchema(SchemaParser().parse(schema), runtimeWiring)

    val graphql = GraphQL.newGraphQL(graphQLSchema).build()
    val executionResult = graphql.execute("""{ hello (what:"Kotlin") } """)

    println(executionResult.getData<Any>())
    // Prints: {hello=Hello Kotlin!}
}
```

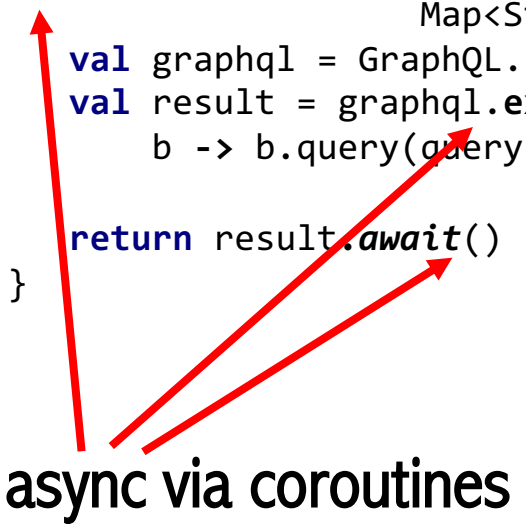
Generify

GraphQL- Handler - Schema

```
class GraphQLHandler(private val schema:GraphQLSchema) {

   constructor(schema:String, resolvers: Map<String, List<Pair<String, DataFetcher<*>>>>) :
      this(schema(schema,resolvers))

   companion object {

   fun schema(schema: String, resolvers: Map<String,List<Pair<String, DataFetcher<*>>>>):GraphQLSchema {

       val typeDefinitionRegistry = SchemaParser().parse(schema)

       val runtimeWiring = newRuntimeWiring()
           .apply { resolvers.forEach { (type, fields) -> this.type(type)
           { builder -> fields.forEach { (field, resolver) ->
                           builder.dataFetcher(field, resolver) };builder } } }
           .build()

       return SchemaGenerator().makeExecutableSchema(typeDefinitionRegistry, runtimeWiring)
   }
}
```

GraphQL-Kotlin - Execute

```
class GraphQLHandler(private val schema:GraphQLSchema) {

    suspend fun execute(query: String, params:
                        Map<String, Any>, op:String?, ctx : Any?): ExecutionResult {
        val graphql = GraphQL.newGraphQL(schema).build()
        val result = graphql.executeAsync {
            b -> b.query(query).variables(params).operationName(op).context(ctx) }

        return result.await()
    }
}
```



async via coroutines

GraphQL-Kotlin - Run

```
fun main(args: Array<String>) {
   runBlocking {
       val schema = """type Query {
                           answer: Int
                           hello(what:String="World"): String
                       }"""
       val resolvers = mapOf("Query" to
               listOf("hello" to DataFetcher { env -> "Hello ${env.getArgument("what")}!" },
                      "answer" to StaticDataFetcher(42)))

       with(GraphQLHandler(schema, resolvers)) {
           val result = execute(args.first(), args.drop(1).zipWithNext().toMap())
           println(result.getData() as Any)
       }
   }
}
```

Run main: "query($p:String) { hello(what:$p) }" p "GraphQL in Kotlin"
Output: {hello=Hello GraphQL in Kotlin}

Let's build a server

ktor.io

lightweight, concise Server & Client
async via coroutines
HTTP/2, WS, JWT, HTML DSL, ...

Ktor

Kotlin Web Server

```
val server = embeddedServer(Netty, port = 8080) {
   install(ContentNegotiation) { jackson { } }
   routing {
       get("/") {
           call.respondText("Hello World!\n", ContentType.Text.Plain)
       }
       get("/json") {
           call.respond(mapOf("OK" to true))
       }
       post("/post/{name}") {
           val body = call.receive<Text>()
           call.respond(Text(body.text+" from "+call.parameters["name"]))
       }
       get("/html") {
           call.respondHtml {
               head { title { +"Title"} }
               body { p { +"Body" }}
           }
       }
   }
}
server.start(wait = true)
```

```
http localhost:8080/
Hello World!

http localhost:8080/json
{"OK":true}

http POST localhost:8080/post/Joe text=Hi
{"text":"Hi from Joe"}

http localhost:8080/html
<html>
  <head>
    <title>Title</title>
  </head>
  <body>
    <p>Body</p>
  </body>
</html>
```

GraphQL Http Server

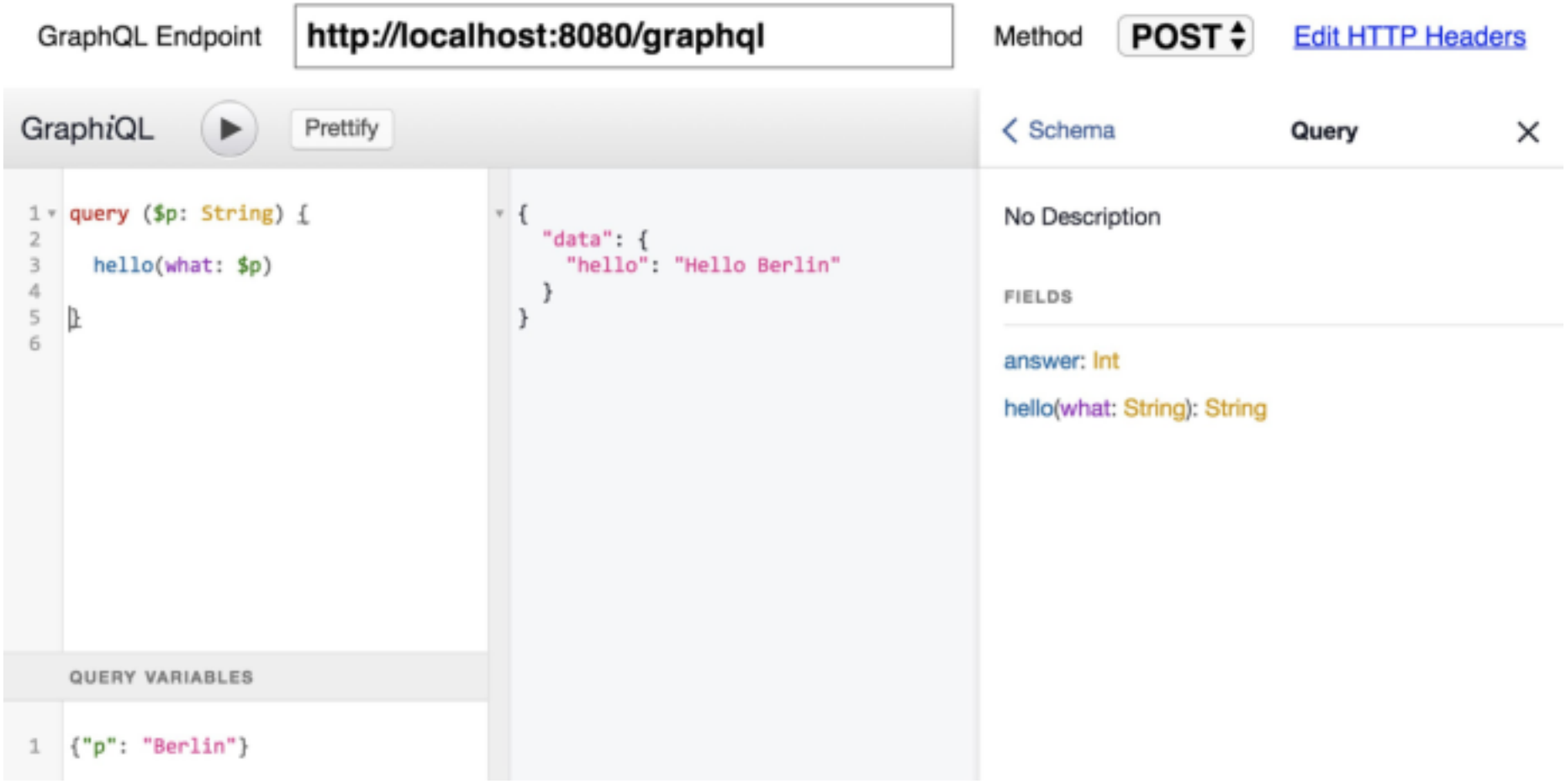
```
fun main(args: Array<String>) {
   val schema = """type Query {
                        answer: Int
                        hello(what:String="World"): String
                    }"""
   val fetchers = mapOf("Query" to
           listOf("hello" to DataFetcher { env -> "Hello "+env.getArgument("what") },
                  "answer" to StaticDataFetcher(42)))
   val handler = GraphQLHandler(schema, fetchers)

   data class Request(val query:String, val params:Map<String,Any>, val operationName : String?)
   val server = embeddedServer(Netty, port = 8080) {
       install(ContentNegotiation) { jackson { } }
       routing {
           post("/graphql") {
               val request = call.receive<Request>()
               val result = handler.execute(request.query, request.params)
               call.respond(mapOf("data" to result.getData<Any>()))
           }
       }
   }
   server.start(wait = true)
}
```

GraphQL Server

```
data class Request(val query:String, val variables:Any?, val operationName : String?) {
   // for GraphiQL
   val params get() =
           when (variables) {
               is String -> MAPPER.readValue(variables, Map::class.java)
               is Map<*, *> -> variables
               else -> emptyMap<String, Any>()
           } as Map<String, Any>
}
```

GraphQL Server (Response)

```
http POST localhost:8080/graphql \
query="{answer}"

{"data":{"answer":42}}
```


GraphQL Endpoint
http://localhost:8080/graphql
Method
POST
Edit HTTP Headers
GraphiQL
Prettify
query ($p: String) {
hello(what: $p)
}
{
"data": {
"hello": "Hello Berlin"
}
}
QUERY VARIABLES
{"p": "Berlin"}
Schema
Query
No Description
FIELDS
answer: Int
hello(what: String): String

What Else?

Kotlin JavaScript

Kotlin JavaScript

- Cross compile to JavaScript
- Include `kotlin.js` std library (`npm install kotlin`)
- Use existing JS libraries with TypeScript headers (`ts2kt`) or
- turn them into **`dynamic`** objects with **`asDynamic()`**
- Reuse common (business) code in multi-platform projects
- Node and Web support, incl. DOM

JavaScript

```
var express = require('express');
var graphqlHTTP = require('express-graphql');
var { buildSchema } = require('graphql');

var schema = buildSchema(`
 type Query {
   hello: String
 }
`);

var root = { hello: () => {   return 'Hello world!';  },};

var app = express();
app.use('/graphql', graphqlHTTP({schema: schema, rootValue: root, graphiql: true}));
app.listen(4000);

console.log('Running a GraphQL API server at localhost:4000/graphql');
```

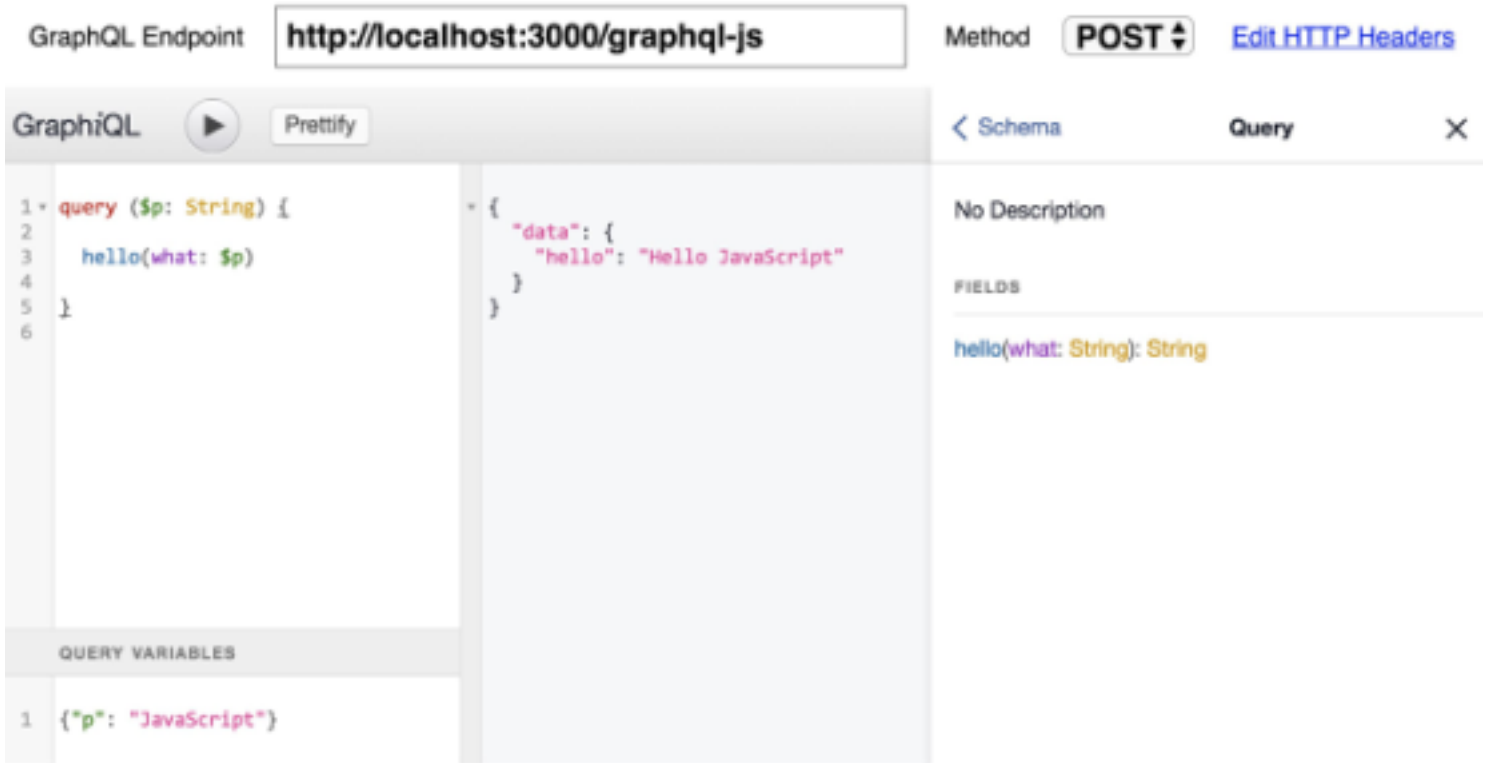
graphql.org/graphql-js/running-an-express-graphql-server/

Kotlin JavaScript

```
external fun require(module:String):dynamic

val express = require("express")
val graphqlHTTP = require("express-graphql");
val buildSchema = require("graphql").buildSchema;

val schema = buildSchema("""
type Query {
   hello(what:String="World"): String
}
""")

fun main(args: Array<String>) {
   val root = asObject("hello" to {ctx:dynamic -> "Hello ${ctx["what"]}"})
   val options = asObject("schema" to schema, "rootValue" to root, "graphiql" to true)

   val app = express()
   app.use("/graphql-js", graphqlHTTP(options))
   app.listen(3000, {
       println("Server started")
   })}
```

kotlinlang.org/docs/tutorials/javascript/

Kotlin JavaScript



Kotlin Native

Kotlin Native

- using LLVM, cross-compile to LLVM-IR
- iOS, Android, Windows, OSX, Unix, WebAssembly
- Use native libs
- Interop with Swift, C / C++, ...
- generate native libs and executables
- use typesafe, clean Kotlin code – shared across JVM, JS, Native

Kotlin Native

- use microHttpd (MHD) as webserver
- use github.com/graphql/libgraphqlparser
  for schema & query parsing
- resolvers as callback functions
- use any db connection or libcurl for API calls for resolvers

Other Notables

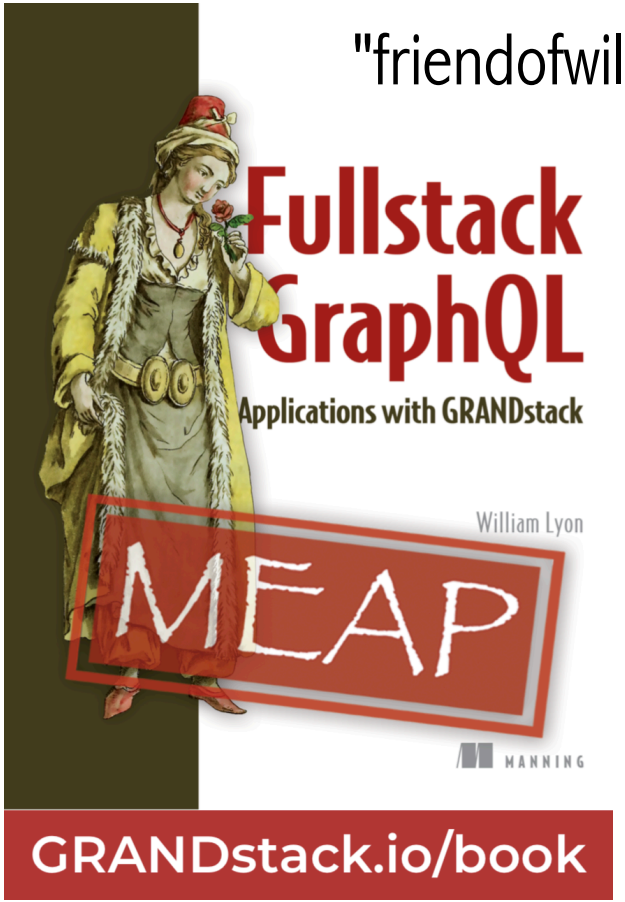
- KGraphQL - Pure Kotlin GraphQL implementation
- kraph: GraphQL request string builder in Kotlin
- kotlinq GraphQL client DSL
- graphql-kotlin libraries for graphql-servers (Expedia)
- Ktor + Koin + KGraphQL + Squash
- Android/Apollo + GraphKool + GraphQL Java Server
- Writing a GraphQL Server in Kotlin

Other Notables

- Micronaut.io Framework (Java, Groovy, Kotlin)
  - GraphQL Demo
- GraphQL Server via GORM
- GraphQL with Kotlin + Spring Boot

GrandStack.io

"friendofwill40"



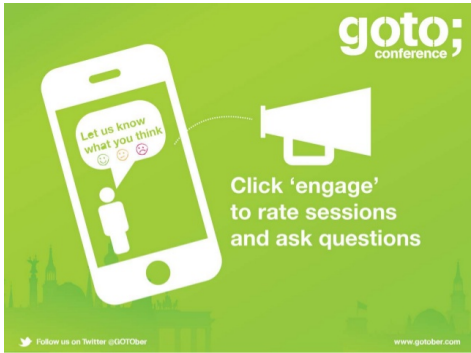
GRANDstack.io/book

Questions?

Please ask through the app!

★ Rate My Session ★

@mesirii



Links

- github.com/jexp/kotlin-graphql
- github.com/graphql-java
- ktor.io
- kotlin-lang.org
- kotlinbyexample.org
- grandstack.io