PARALLELIZING PRODUCT DEVELOPMENT WITH GRAPHQL

@CHRISBISCARDI

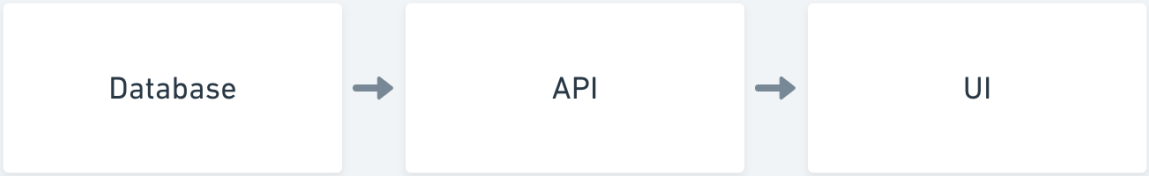

honeycomb.io

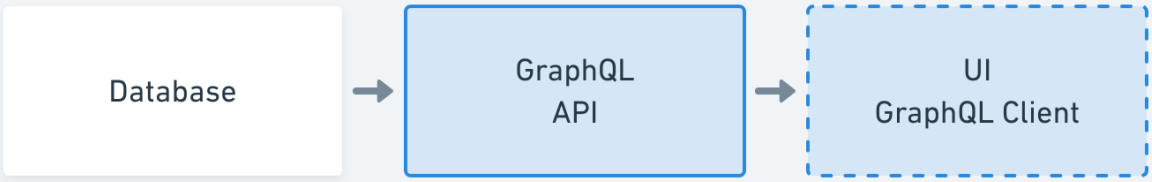

coffee

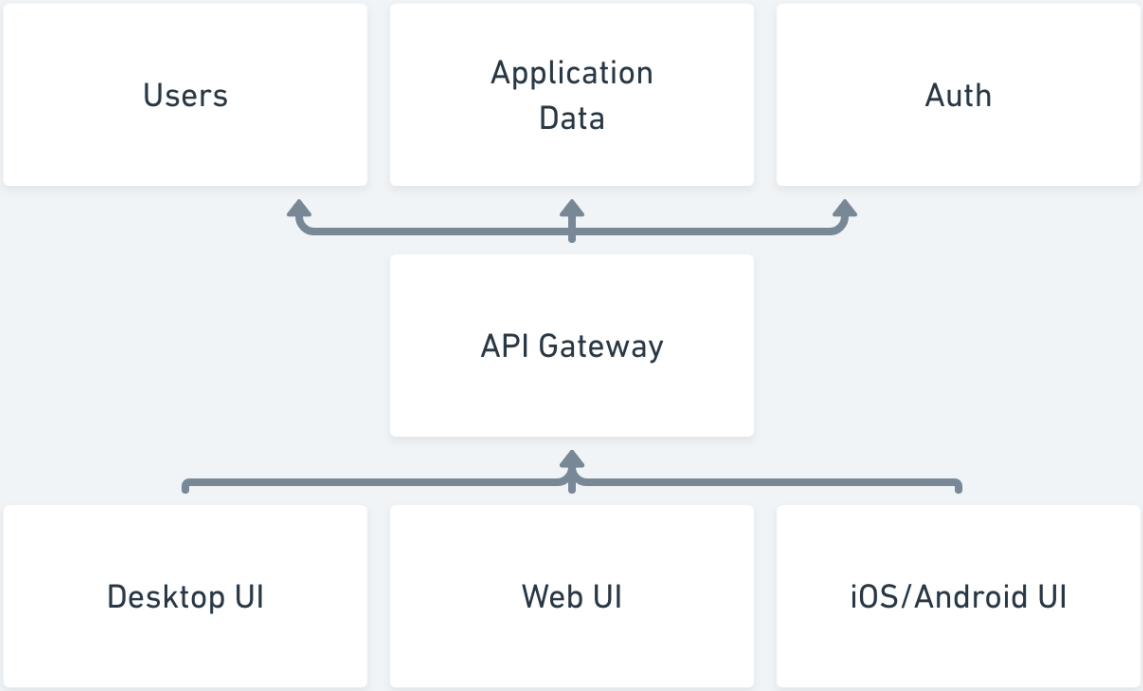

@chrisbiscardi

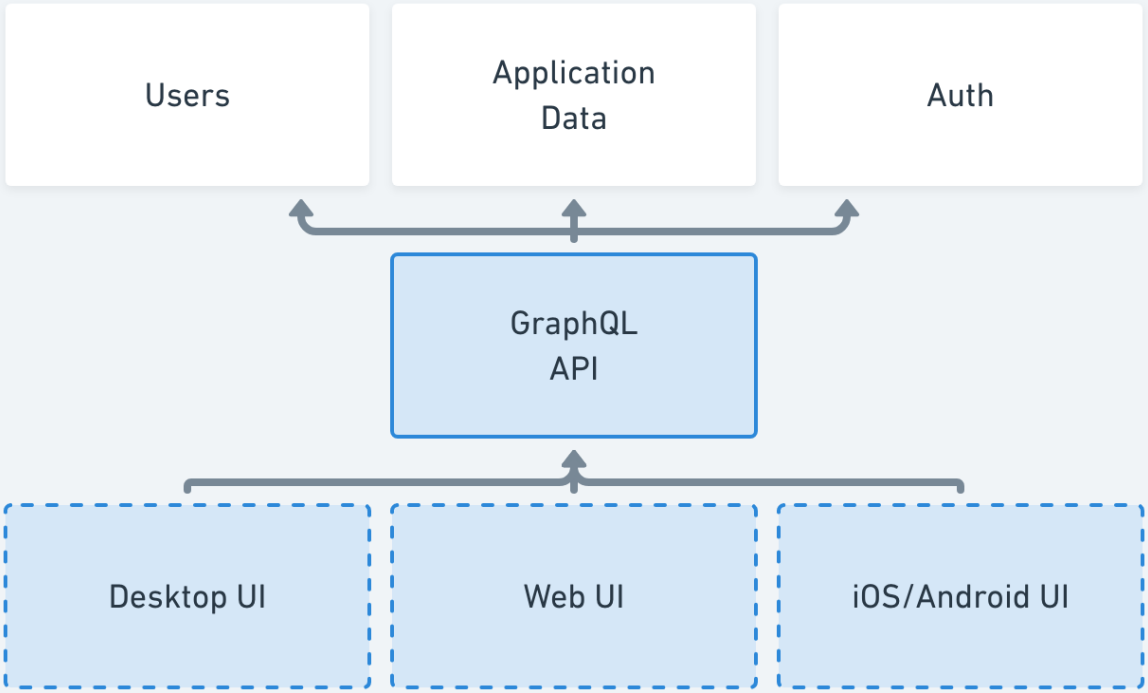

biscarch

APPLICATION ARCHITECTURE

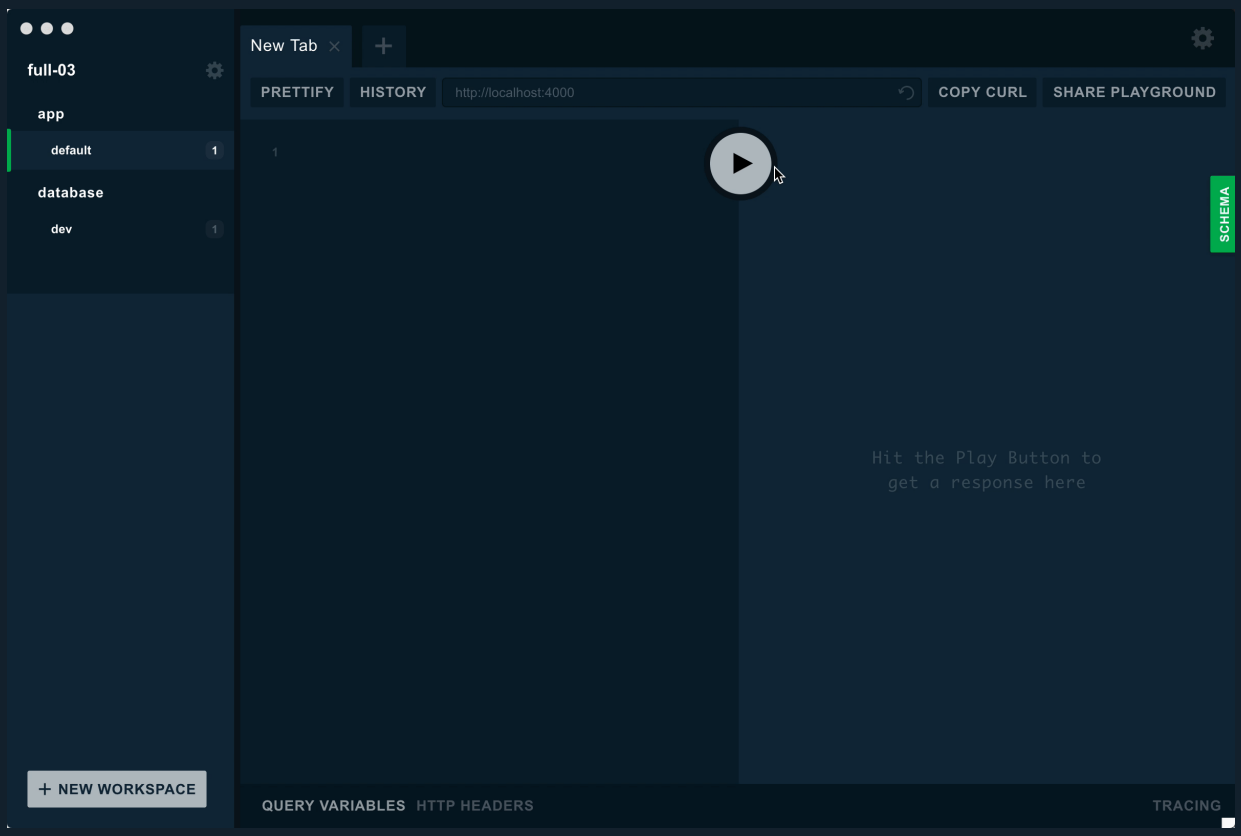

Database
API
UI


Database
GraphQL
API
UI
GraphQL Client


Users
Application Data
Auth
API Gateway
Desktop UI
Web UI
iOS/Android UI


Users
Application Data
Auth
GraphQL API
Desktop UI
Web UI
iOS/Android UI

SCHEMA DEFINITION LANGUAGE


full-03
app
default
1
database
dev
1
+ NEW WORKSPACE
New Tab
PRETTIFY
HISTORY
http://localhost:4000
COPY CURL
SHARE PLAYGROUND
1
SCHEMA
Hit the Play Button to
get a response here
QUERY VARIABLES
HTTP HEADERS
TRACING

Queries

```
{
  drafts {
    title
  }
}
```

Results

```
{
  "data" {
    "drafts": [{
      "title": "Solving World Hunger"
    }]
  }
}
```

blog schema

```
type Post {
  id: String!
  title: String!
  publishedAt: DateTime!
  likes: Int! @default(value: 0)
  blog: Blog @relation(name: "Posts")
}
type Blog {
  id: String!
  name: String!
  description: String,
  posts: [Post!]! @relation(name: "Posts")
}
```

Arrays

```
type Post {
  id: String!
  title: String!
  publishedAt: DateTime!
  likes: Int! @default(value: 0)
  blog: Blog @relation(name: "Posts")
}
type Blog {
  id: String!
  name: String!
  description: String,
  posts: [Post!]! @relation(name: "Posts")
}
```

DIRECTIVES
EVERYTHING ELSE

RESOLVERS

Trivial Resolvers

```
Human: {
  name(obj, args, context) {
    return obj.name
  }
}
```

Async Resolvers

```
human(obj, args, context) {
  return context.db.loadHumanByID(args.id).then(
    userData => new Human(userData)
  )
}
```

Directive Resolvers

```
directive @upper on FIELD_DEFINITION

type Query {
  hello: String @upper
}
```

Directive Resolvers

```
upper(next, src, args, context ) {
  return next().then((str) => {
    if (typeof(str) === 'string') {
      return str.toUpperCase();
    }
    return str;
  });
}
```

SCENARIO 1

A New Product


PRISMA

Resolving with SQL

```
return mysql.query(`SELECT
  "user"."id" AS "id",
  "posts"."id" AS "postId",
  "posts"."title" AS "postTitle",
  "posts"."body" AS "postText",
  "posts"."tags" AS "postTags",
  "posts"."created" AS "postCreated",
FROM accounts AS "user"
LEFT JOIN posts ON "user".id = "posts".authorId
WHERE "user".id = ${obj.id}
AND isPublished = 1
LIMIT ${args.skip}, 1000`)
```

SQL with Prisma

```
publicPosts(obj, args, ctx, info) {
  return ctx.prisma.query.posts(
    where: {
      author: { id: obj.id },
      isPublished: true,
    },
    skip: args.skip,
    info // enables schema stitching
  )
}
```

prisma init

```
➜ prisma init full-03
? How to set up a new Prisma service?

  Minimal setup: database-only
❯ GraphQL server/fullstack boilerplate (recommended)
```

prisma init

```
➜ prisma init full-03
? How to set up a new Prisma service?

Running $ graphql create ...
? Choose GraphQL boilerplate project:
❯ node-basic               Basic GraphQL server (incl.
database)
  node-advanced            GraphQL server (incl. database &
authentication)
  typescript-basic         Basic GraphQL server (incl.
database)
  typescript-advanced      GraphQL server (incl. database &
authentication)
  react-fullstack-basic    React app + GraphQL server (incl.
database )
```

prisma deploy

```
[graphql create] Running boilerplate install script...
Running $ prisma deploy...

? Please choose the cluster you want to deploy "full-03@dev"
to

  prisma-eu1      Public development cluster
  prisma-us1      Public development cluster
❯ local           Local cluster (requires Docker)
```

prisma deploy

```
Changes:

  Post (Type)
  + Created type `Post`
  + Created field `id` of type `GraphQLID!`
  + Created field `isPublished` of type `Boolean!`
  + Created field `title` of type `String!`
  + Created field `text` of type `String!`
  + Created field `updatedAt` of type `DateTime!`
  + Created field `createdAt` of type `DateTime!`

Applying changes 1.1s

Hooks:
Importing seed dataset from `seed.graphql` 498ms

Writing database schema to `src/generated/prisma.graphql`
0ms
```

tree . -I node_modules

```
➜ tree . -I node_modules
.
├── README.md
├── database
|   ├── datamodel.graphql
|   ├── prisma.yml
|   └── seed.graphql
├── package.json
├── src
|   ├── generated
|   |   └── prisma.graphql
|   ├── index.js
|   └── schema.graphql
└── yarn.lock

3 directories, 9 files
```

Schema

```
type Post {
  id: ID! @unique
  isPublished: Boolean! @default(value: false)
  title: String!
  text: String!
}
```

Seeds

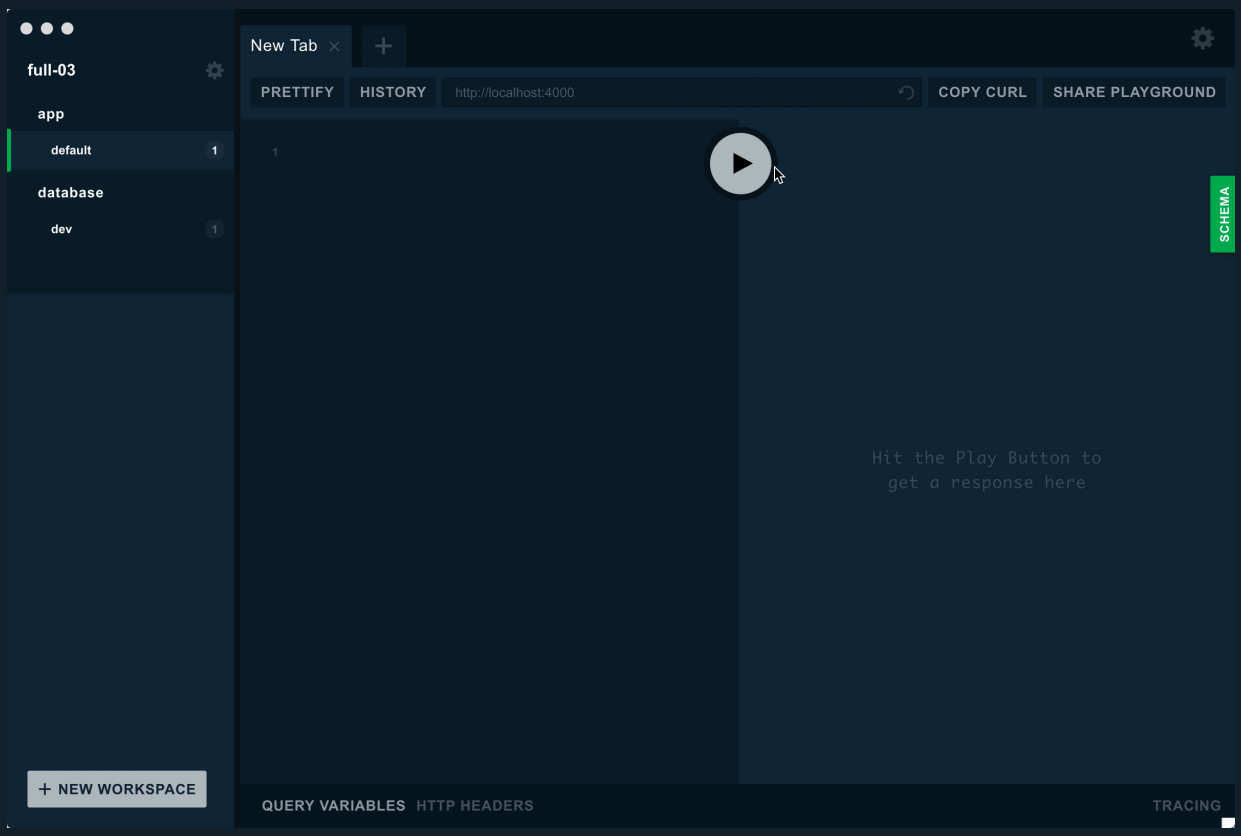
```
mutation {
  first: createPost(data: {
    title: "Hello World"
    text: "This is my first blog post ever!"
    isPublished: true
  }) {
    id
  }

  second: createPost(data: {
    title: "My Second Post"
    text: "My first post was good, but this one is better!"
    isPublished: true
  }) {
    id
  }

  third: createPost(data: {
    title: "Solving World Hunger"
    text: "This is a draft..."
    isPublished: false
  }) {
    id
  }
}
```

playground

```
➜ yarn playground
yarn run v1.3.2
$ graphql playground
Serving playground at http://localhost:3000/playground
```

playground

```
brew cask install graphql-playground
```


full-03
app
default
1
database
dev
1
+ NEW WORKSPACE
New Tab
PRETTIFY
HISTORY
http://localhost:4000
COPY CURL
SHARE PLAYGROUND
1
SCHEMA
Hit the Play Button to
get a response here
QUERY VARIABLES
HTTP HEADERS
TRACING

Updating

```
type Post {
  id: ID! @unique
  isPublished: Boolean! @default(value: false)
  title: String! @deprecated
  title2: String
  text: String!
}
```

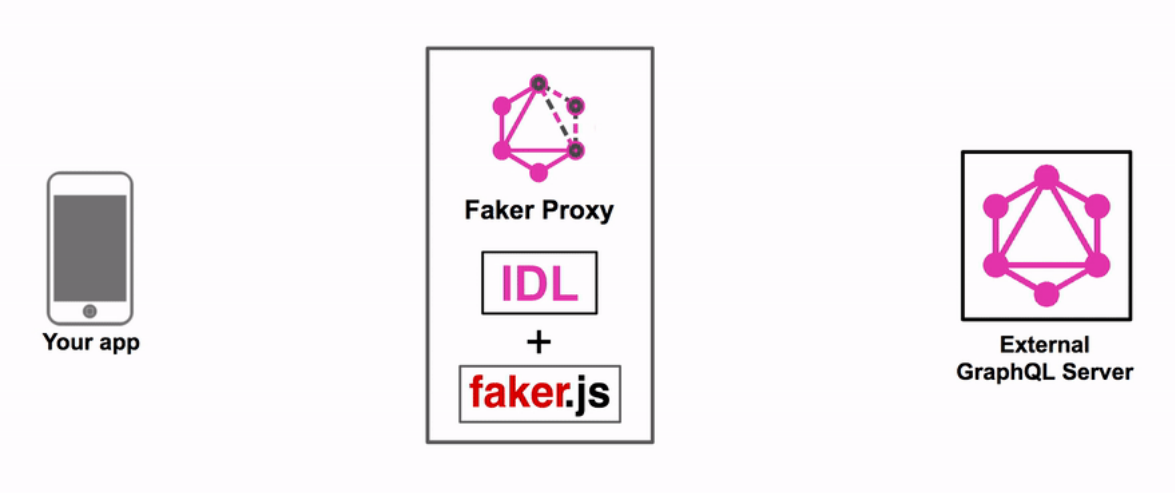
Updating

```
➜ yarn prisma deploy
Changes:

  Post (Type)
  + Created field `title2` of type `String`

Applying changes 1.1s

Your GraphQL database endpoint is live:

  HTTP:  http://localhost:4466/full-03/dev
  WS:    ws://localhost:4466/full-03/dev
```

SCENARIO 2

Frontend

graphql-faker



Faker Directives

```
type Person {
  name: String @fake(type: firstName)
  gender: String @examples(values: ["male", "female"])
}
```


APOLLO

Apollo Boost

- apollo-client
- apollo-cache-inmemory
- apollo-link-http
- apollo-link-error
- apollo-link-state
- graphql-tag

link-state

```
import { withClientState } from 'apollo-link-state';

// This is the same cache you pass into new ApolloClient
const cache = new InMemoryCache(...);

const stateLink = withClientState({
  cache,
  resolvers: {
    Mutation: {
      updateNetworkStatus: (_, { isConnected }, { cache }) =>
{
        const data = {
          networkStatus: {
            __typename: 'NetworkStatus',
            isConnected
          },
        };
        cache.writeData({ data });
        return null
      },
    },
  }
});
```

SCENARIO 3

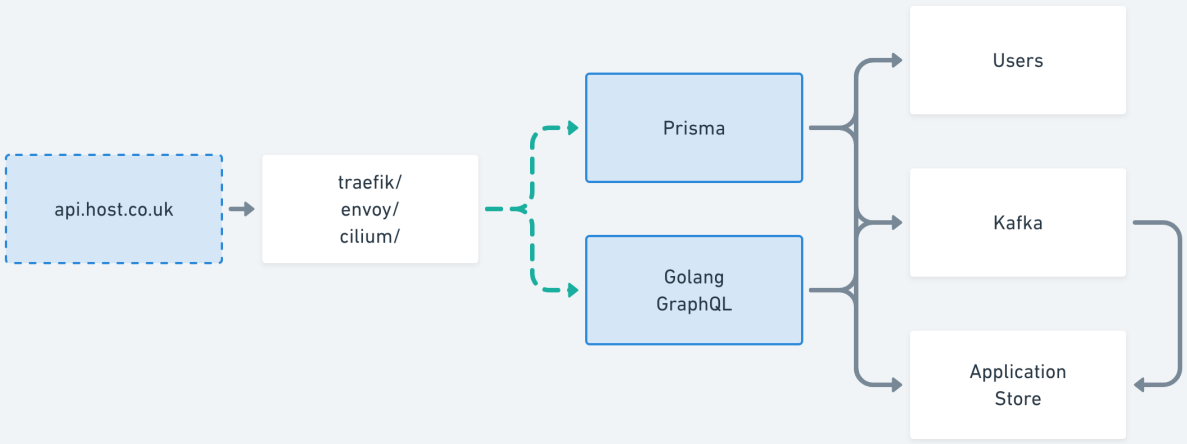
Go is Awesome!

src/schema.graphql

```
# import Post from "./generated/prisma.graphql"

type Query {
  feed: [Post!]!
  drafts: [Post!]!
  post(id: ID!): Post
}

type Mutation {
  createDraft(title: String!, text: String): Post
  deletePost(id: ID!): Post
  publish(id: ID!): Post
}
```

gqlgen -out generated.go -package main

```
type Query struct {
  PostsID           int
  PostID            int
  PostsConnectionID int
  NodeID            int
}
type PostWhereInput struct {
  AND                  []PostWhereInput
  OR                   []PostWhereInput
  ID                   *string
  Id_not               *string
  Id_in                []string
  Id_not_in            []string
  Id_lt                *string
}
```


api.host.co.uk
traefik/
envoy/
cilium/
Prisma
Golang
GraphQL
Users
Kafka
Application
Store

CREDITS

- [presentation]: Spectacle
- [diagrams]: whimsical.co

Q & A