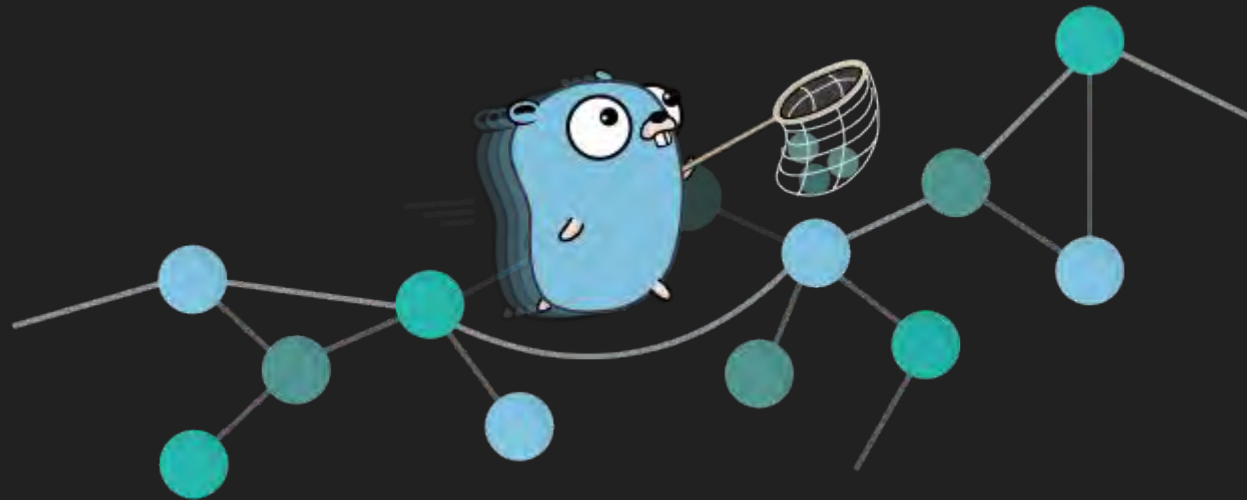


Ent: Making Data Easy in Go



What do I do?

About Me

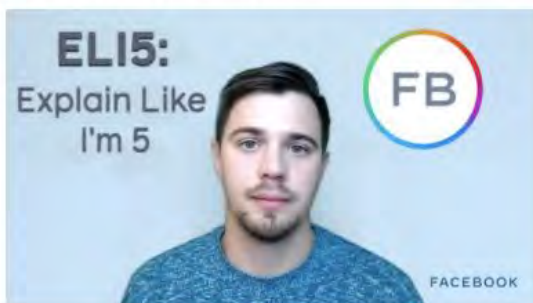
@DmitryVinnik

About Me

Open Source Developer Advocate

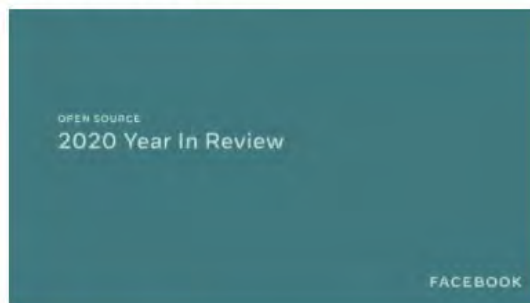
Empowering diverse community through open source technology

**Introducing ELI5:
Explain Like I'm 5 Series**



[Watch series on YouTube](#)

**Open Source:
2020 Year in Review**



[Read the update](#)

**The Diff:
A Podcast From Facebook Open Source**



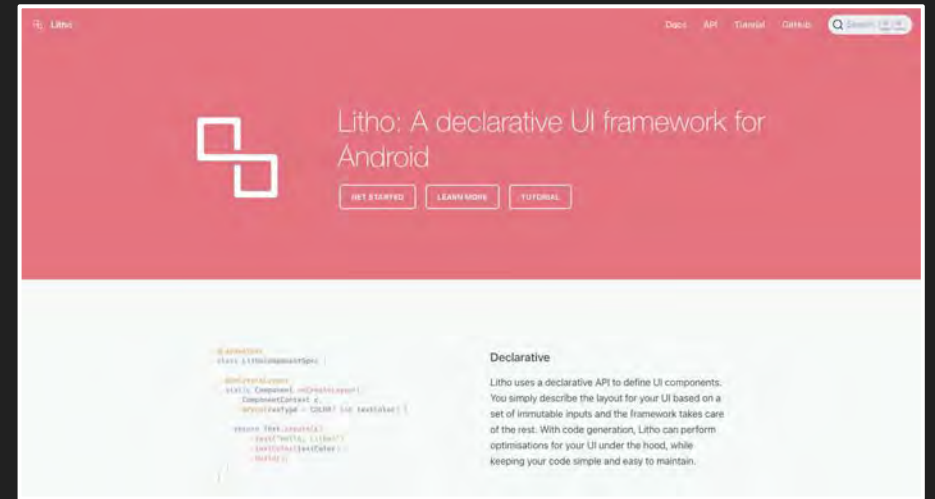
[Listen to the podcast](#)

About Me

Open Source Developer Advocate

Mobile Focus: Android, iOS, Hybrid







React Native 0.44

Docs Components API Community Blog

React Native

Learn once, write anywhere.

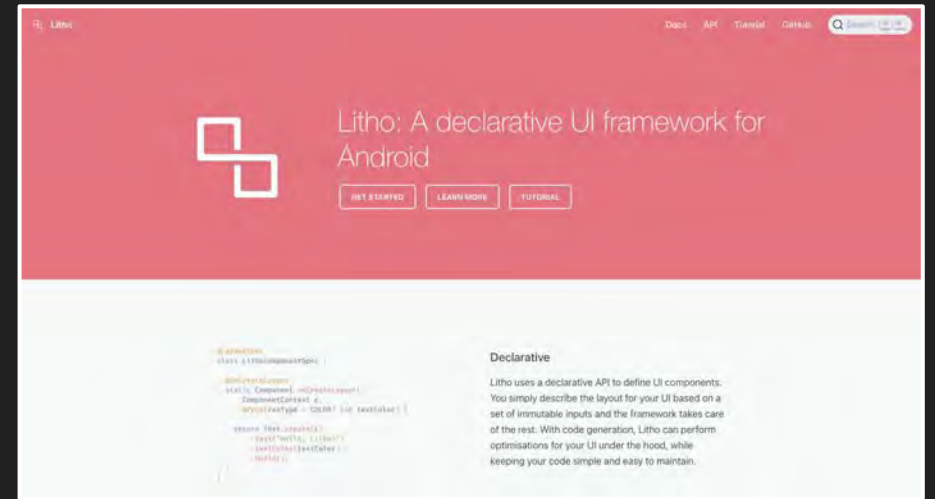
Get started Learn basics



Create native apps for Android and iOS using React

React Native combines the best parts of native development with React, a best-in-class JavaScript library for building user interfaces.

Use a little—or a lot. You can use React Native today in your existing Android and iOS projects or you can create a whole new app from scratch.



Litho

Docs API Tutorial GitHub



Litho: A declarative UI framework for Android

GET STARTED LEARN MORE TUTORIAL

```
class MyComponent {
  @DrawableRes
  int[] colors;

  @DrawableRes
  int[] colors2;

  @DrawableRes
  int[] colors3;

  @DrawableRes
  int[] colors4;

  @DrawableRes
  int[] colors5;

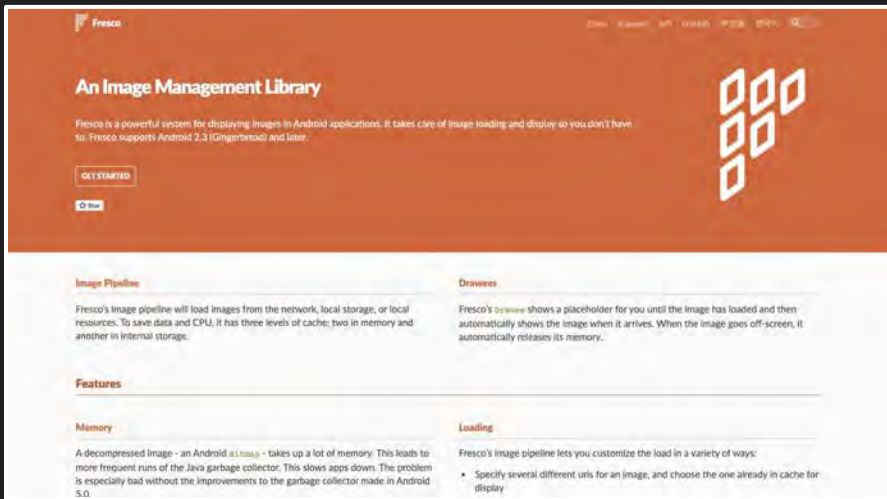
  @DrawableRes
  int[] colors6;

  @DrawableRes
  int[] colors7;

  @DrawableRes
  int[] colors8;
}
```

Declarative

Litho uses a declarative API to define UI components. You simply describe the layout for your UI based on a set of immutable inputs and the framework takes care of the rest. With code generation, Litho can perform optimisations for your UI under the hood, while keeping your code simple and easy to maintain.



Fresco

Docs API Tutorial GitHub

An Image Management Library

Fresco is a powerful system for displaying images in Android applications. It takes care of image loading and display so you don't have to. Fresco supports Android 2.3 (Gingerbread) and later.

GET STARTED

Use

Image Pipeline

Fresco's image pipeline will load images from the network, local storage, or local resources. To save data and CPU, it has three levels of cache: two in memory and another in internal storage.

Drawees

Fresco's `Drawee` shows a placeholder for you until the image has loaded and then automatically shows the image when it arrives. When the image goes off-screen, it automatically releases its memory.

Features

Memory

A decompressed image - an Android `Bitmap` - takes up a lot of memory. This leads to more frequent runs of the Java garbage collector. This slows apps down. The problem is especially bad without the improvements to the garbage collector made in Android 5.0.

Loading

Fresco's image pipeline lets you customize the load in a variety of ways:

- Specify several different URIs for an image, and choose the one already in cache for display

React Native 0.44

Docs Components API Community Blog

React Native

Learn once, write anywhere.

Get started Learn basics



Create native apps for Android and iOS using React

React Native combines the best parts of native development with React, a best-in-class JavaScript library for building user interfaces.

Use a little—or a lot. You can use React Native today in your existing Android and iOS projects or you can create a whole new app from scratch.



Litho

Docs API Tutorial GitHub



Litho: A declarative UI framework for Android

GET STARTED LEARN MORE TUTORIAL

```

@android.support.design.widget.TextInputLayout
class LithoTextInputLayout {
    @android.support.design.widget.TextInputLayout
    LithoTextInputLayout(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
    }
}

```

Declarative

Litho uses a declarative API to define UI components. You simply describe the layout for your UI based on a set of immutable inputs and the framework takes care of the rest. With code generation, Litho can perform optimizations for your UI under the hood, while keeping your code simple and easy to maintain.

Fresco

Docs API Tutorial GitHub

An Image Management Library

Fresco is a powerful system for displaying images in Android applications. It takes care of image loading and display so you don't have to. Fresco supports Android 2.3 (Gingerbread) and later.

GET STARTED

View

Image Pipeline

Fresco's image pipeline will load images from the network, local storage, or local resources. To save data and CPU, it has three levels of cache: two in memory and another in internal storage.

Drawers

Fresco's `Drawable` shows a placeholder for you until the image has loaded and then automatically shows the image when it arrives. When the image goes off-screen, it automatically releases its memory.

Features

Memory

A decompressed image - an Android `Bitmap` - takes up a lot of memory. This leads to more frequent runs of the Java garbage collector. This slows apps down. The problem is especially bad without the improvements to the garbage collector made in Android 5.0.

Loading

Fresco's image pipeline lets you customize the load in a variety of ways:

- Specify several different URIs for an image, and choose the one already in cache for display.

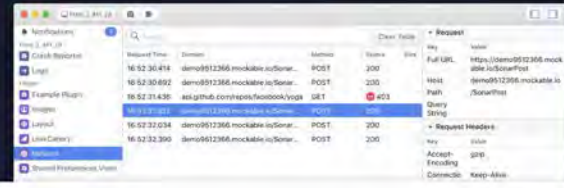
Flipper

Features Setup Creating Plugins Under the Hood

Extensible mobile app debugger

Flipper is a platform for debugging iOS, Android and React Native apps. Visualize, inspect, and control your apps from a simple desktop interface. Use Flipper as is or extend it using the plugin API!

Download Mac Linux Windows Learn more



Timestamp	Domain	Method	Status	Size	Request
16:52:30.474	demo12366.mockable.ioServer...	POST	200		Full URL: https://demo12366.mockable.io/ServerPost
16:52:30.492	demo12366.mockable.ioServer...	POST	200		Full URL: https://demo12366.mockable.io/ServerPost
16:52:31.436	img.mdn.cdn.mozilla.net	GET	200	403	Query String: ...
16:52:31.802	demo12366.mockable.ioServer...	POST	200		Request Headers: ...
16:52:32.034	demo12366.mockable.ioServer...	POST	200		Request Headers: ...
16:52:32.390	demo12366.mockable.ioServer...	POST	200		Request Headers: ...

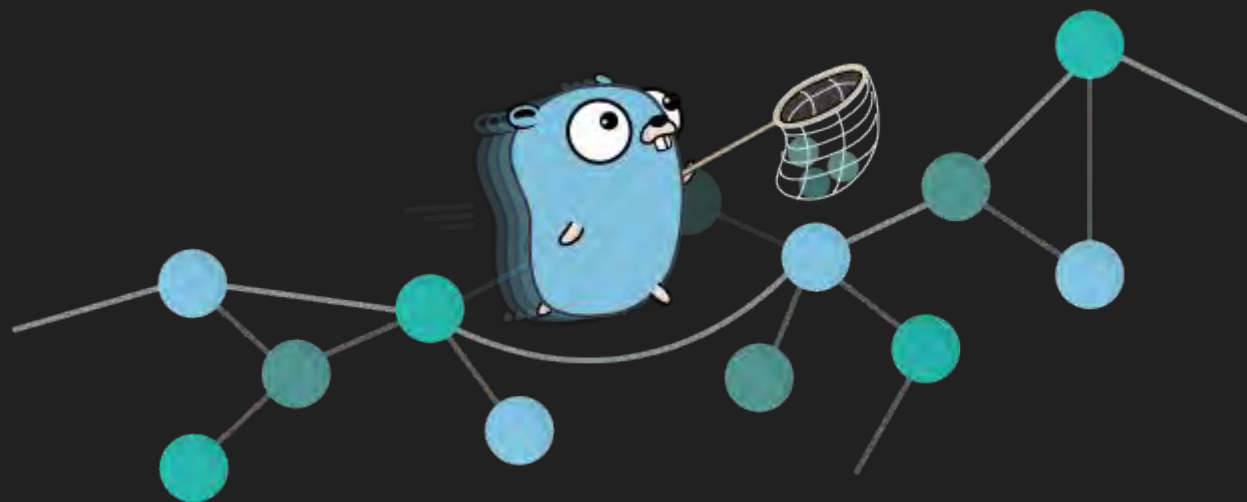
About Me

Open Source Developer Advocate

Mobile Focus: Android, iOS, Hybrid

Passionate about Open Source

Ent



What is Ent?

Ent is ORM for Go

Ent is **ORM** for Go

ORM

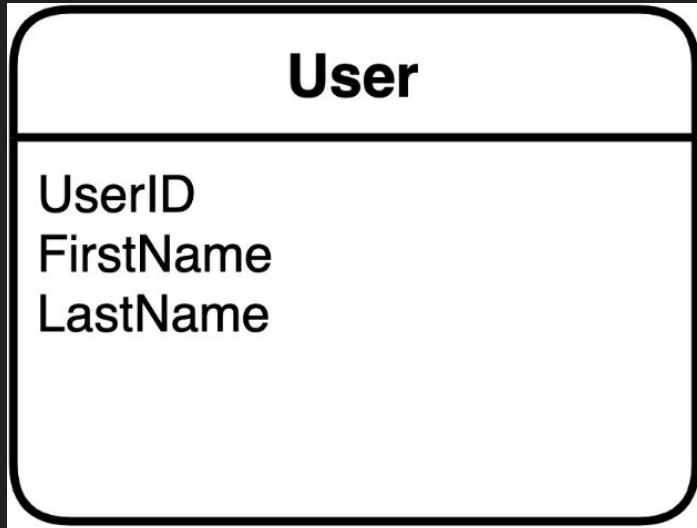
Object Relational Mapping

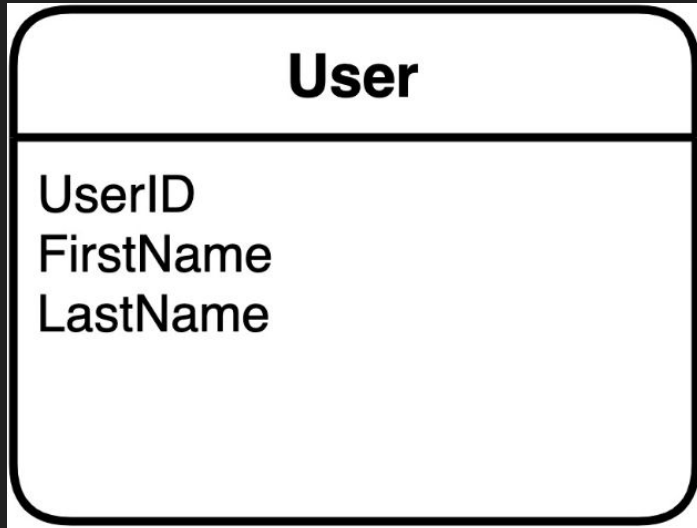
“Technique for converting data between incompatible type systems using object-oriented programming languages”

Best online source ever

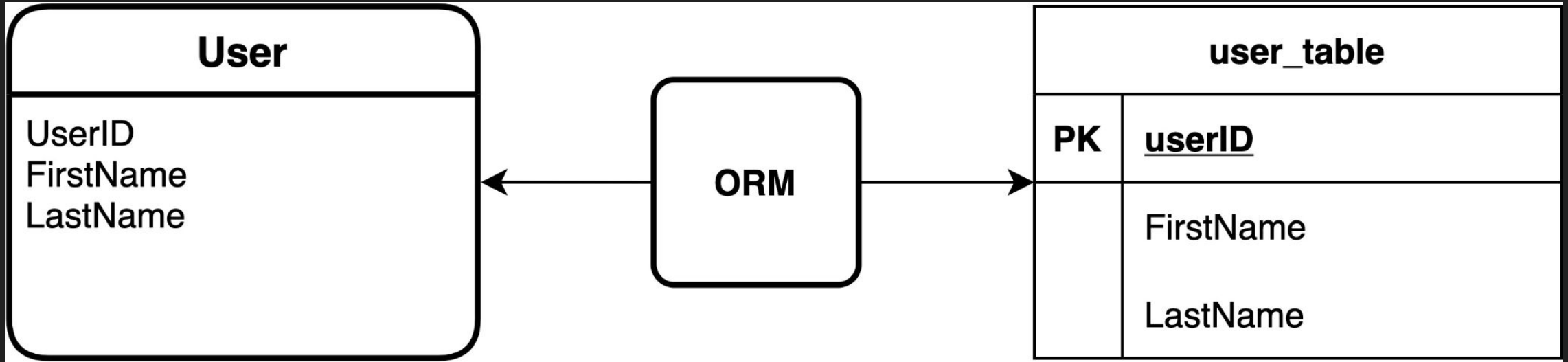
“Technique for converting data between incompatible type systems using object-oriented programming languages”

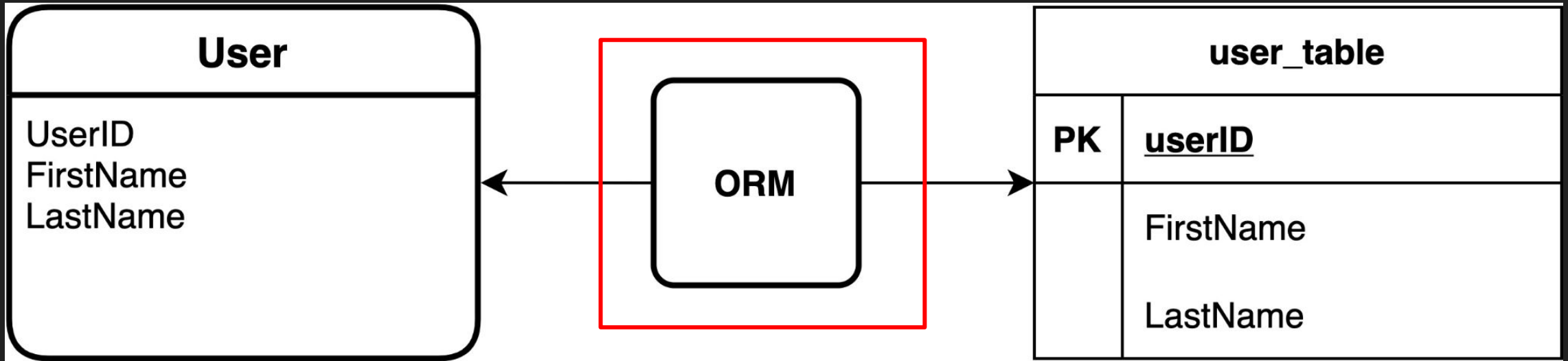
Wikipedia

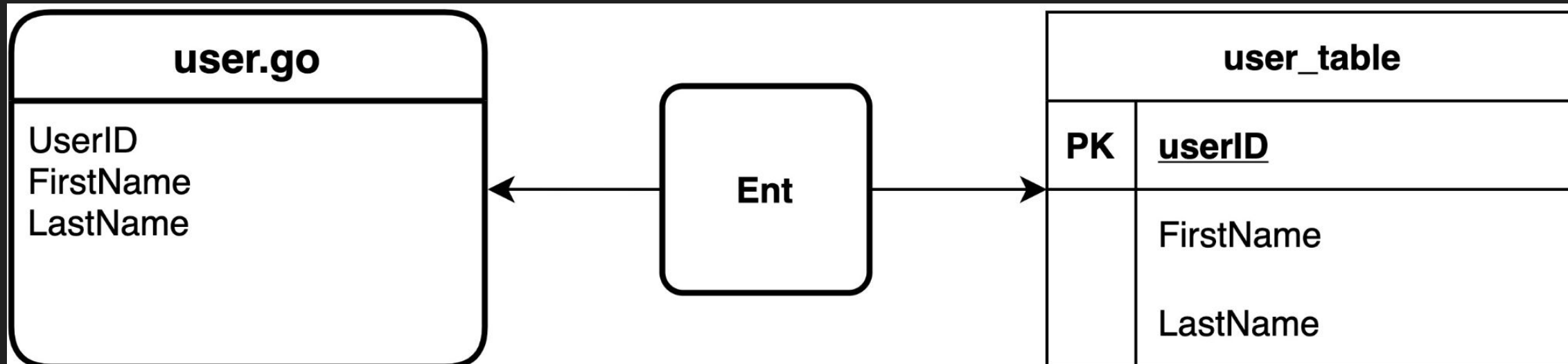




user_table	
PK	<u>userID</u>
	FirstName
	LastName







Why is ORM important?

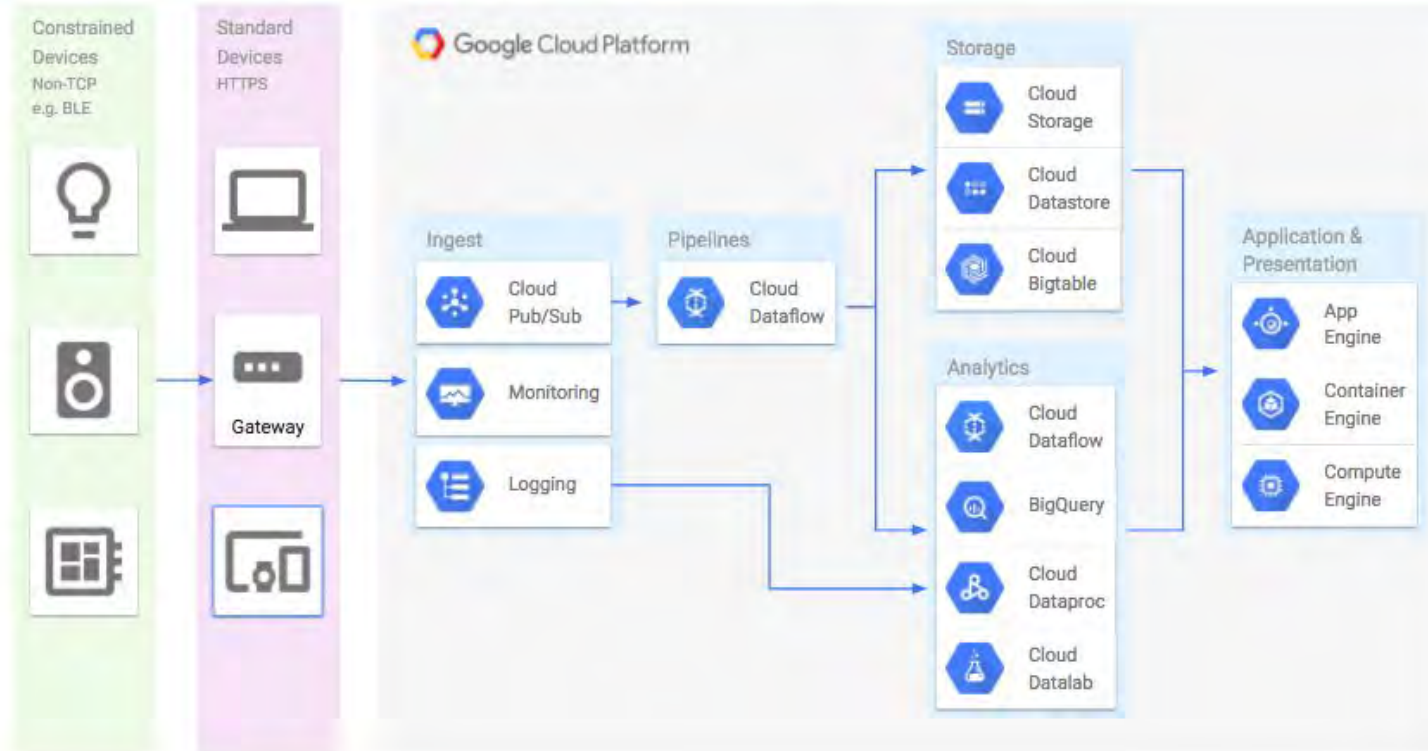
It's all about *data*

Big or Small

Most apps are data
driven

General > Real Time Stream Processing IoT

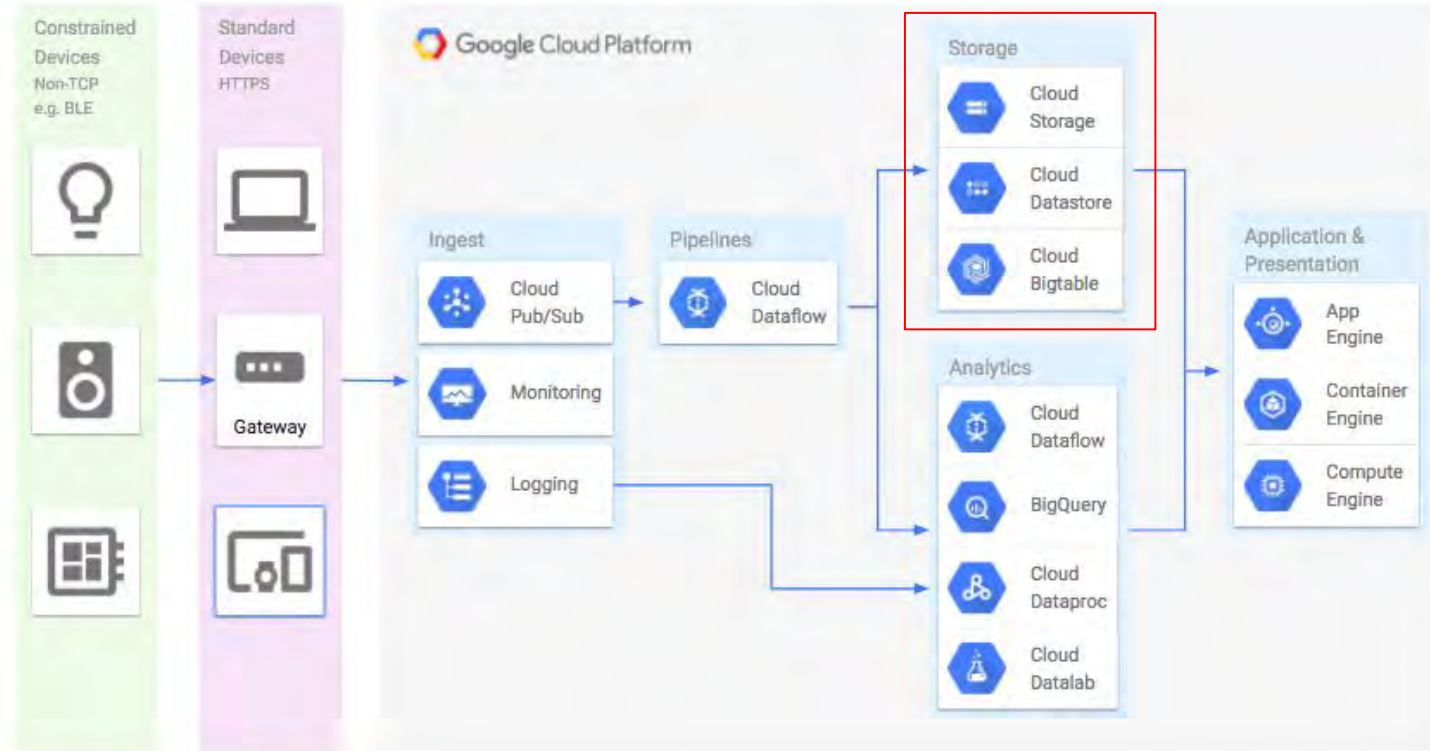
Architecture: General > Real Time Stream Processing - Internet of Things



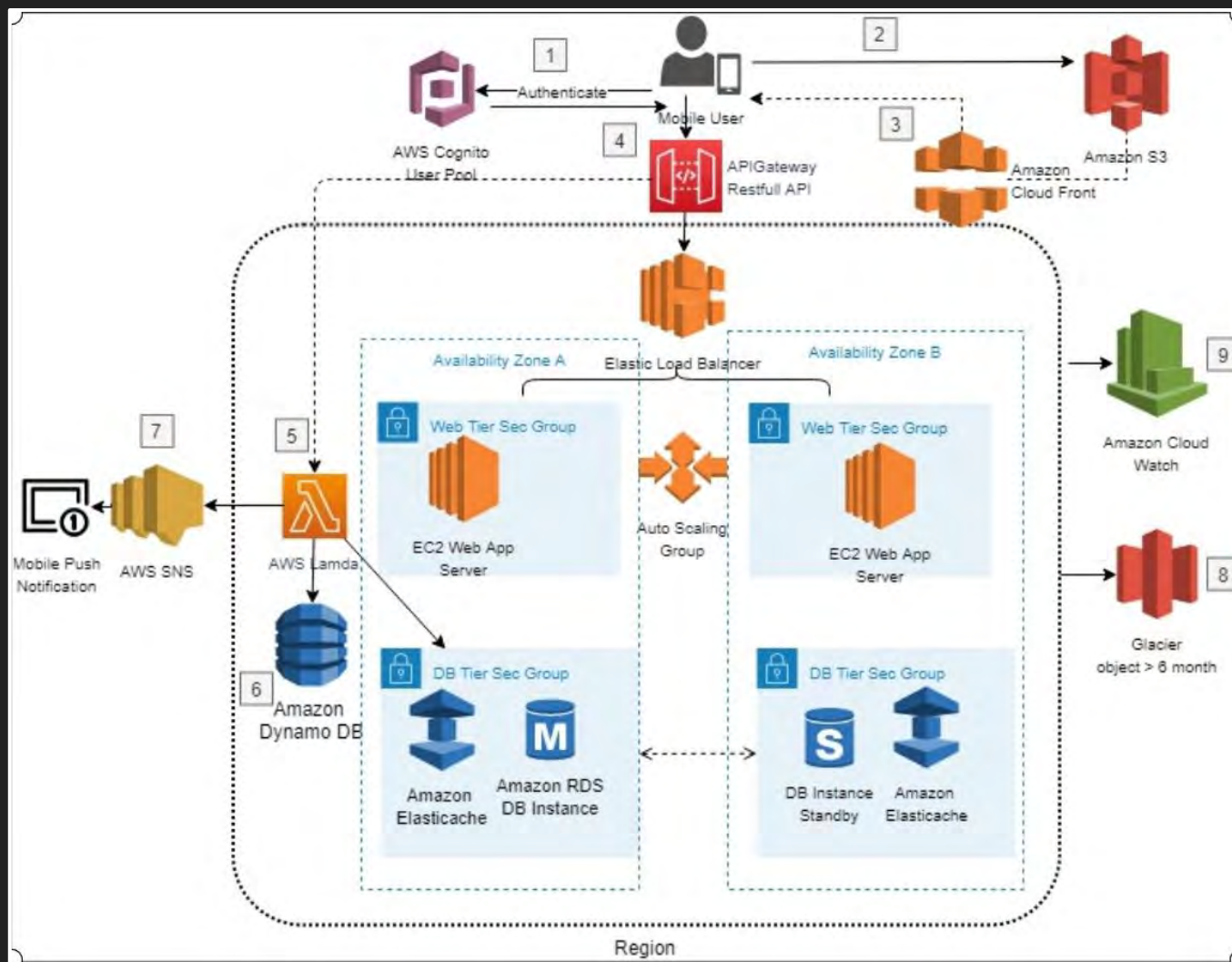
Google, Cloud Integration

General > Real Time Stream Processing IoT

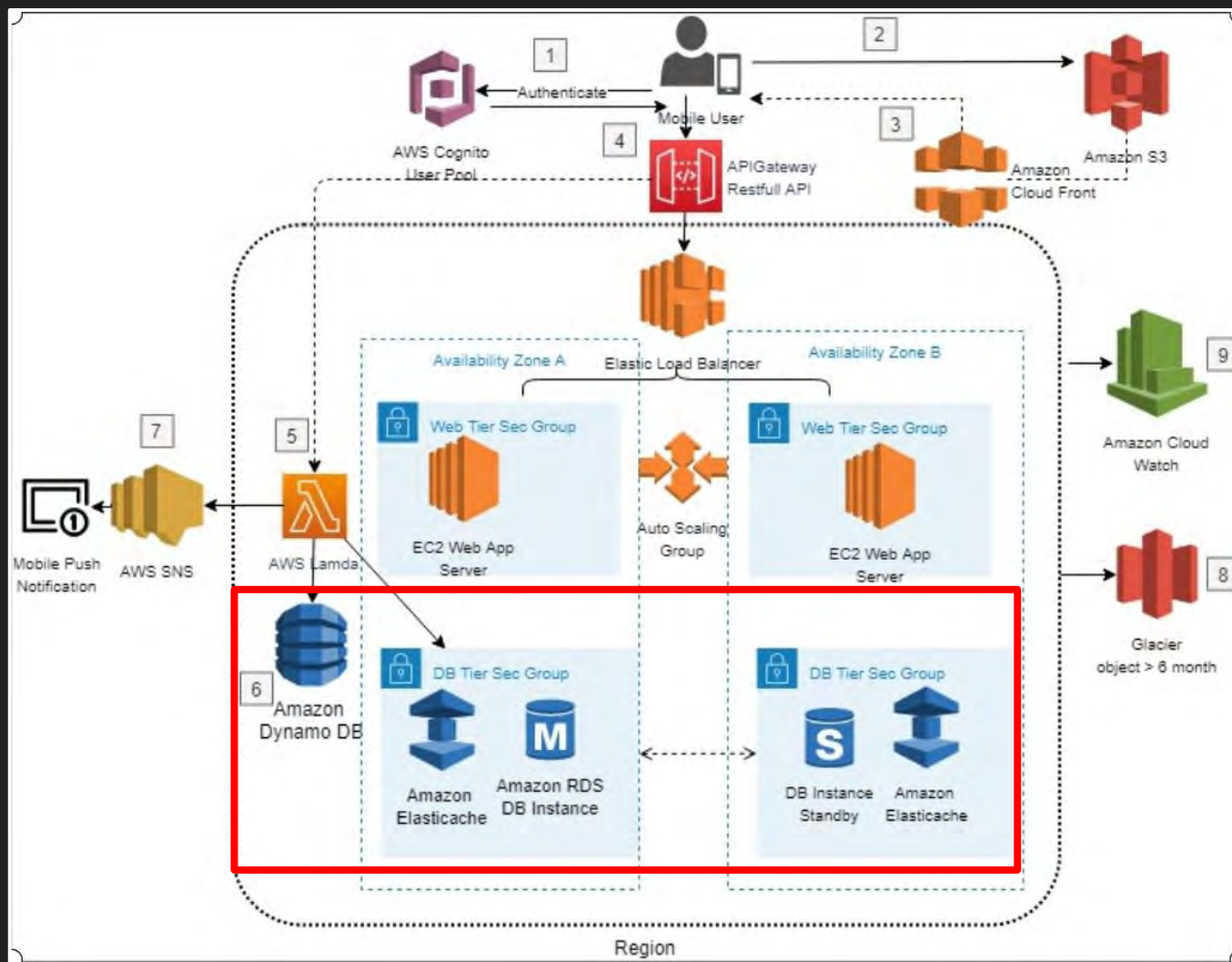
Architecture: General > Real Time Stream Processing - Internet of Things



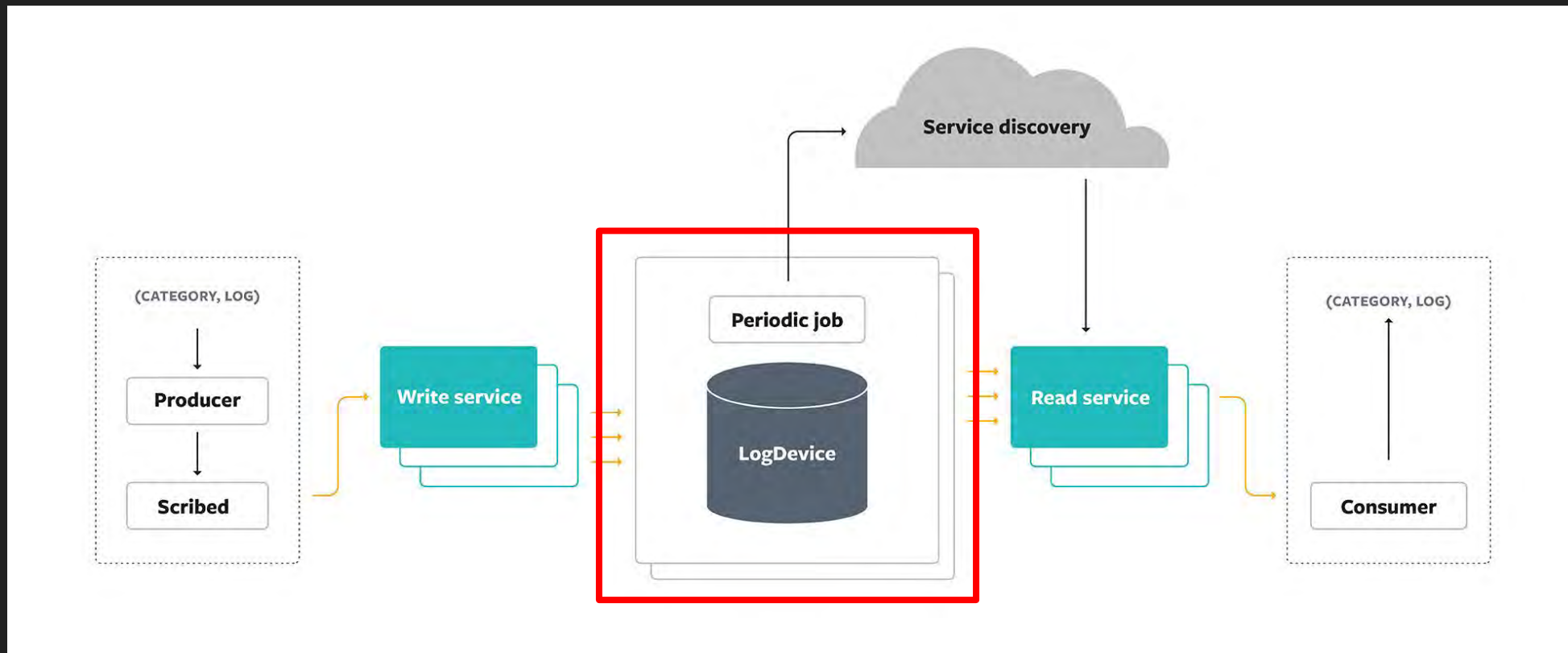
Google, Cloud Integration



AWS, Startups Use-Case

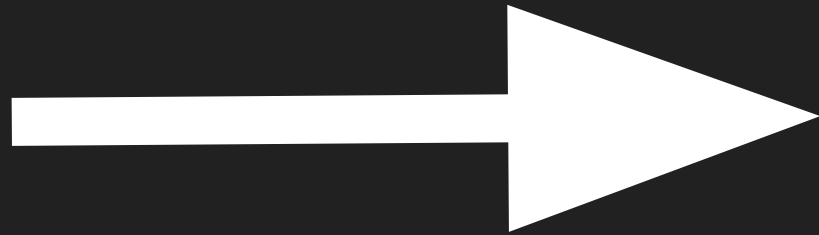


AWS, Startups Use-Case



Facebook Scribe, Log Processor

Data is important



ORM is important too!

Enough theory

Let's code!

Intro to Ent



```
go get entgo.io/ent/cmd/ent
```




```
go mod init <project_name>
```

The image shows a screenshot of the Visual Studio Code editor interface. On the left, the Explorer sidebar is visible, showing a project named 'ENTGO-TALK' with a subdirectory 'ent' and a file 'go.mod'. The main editor window displays the content of 'go.mod' with the following code:

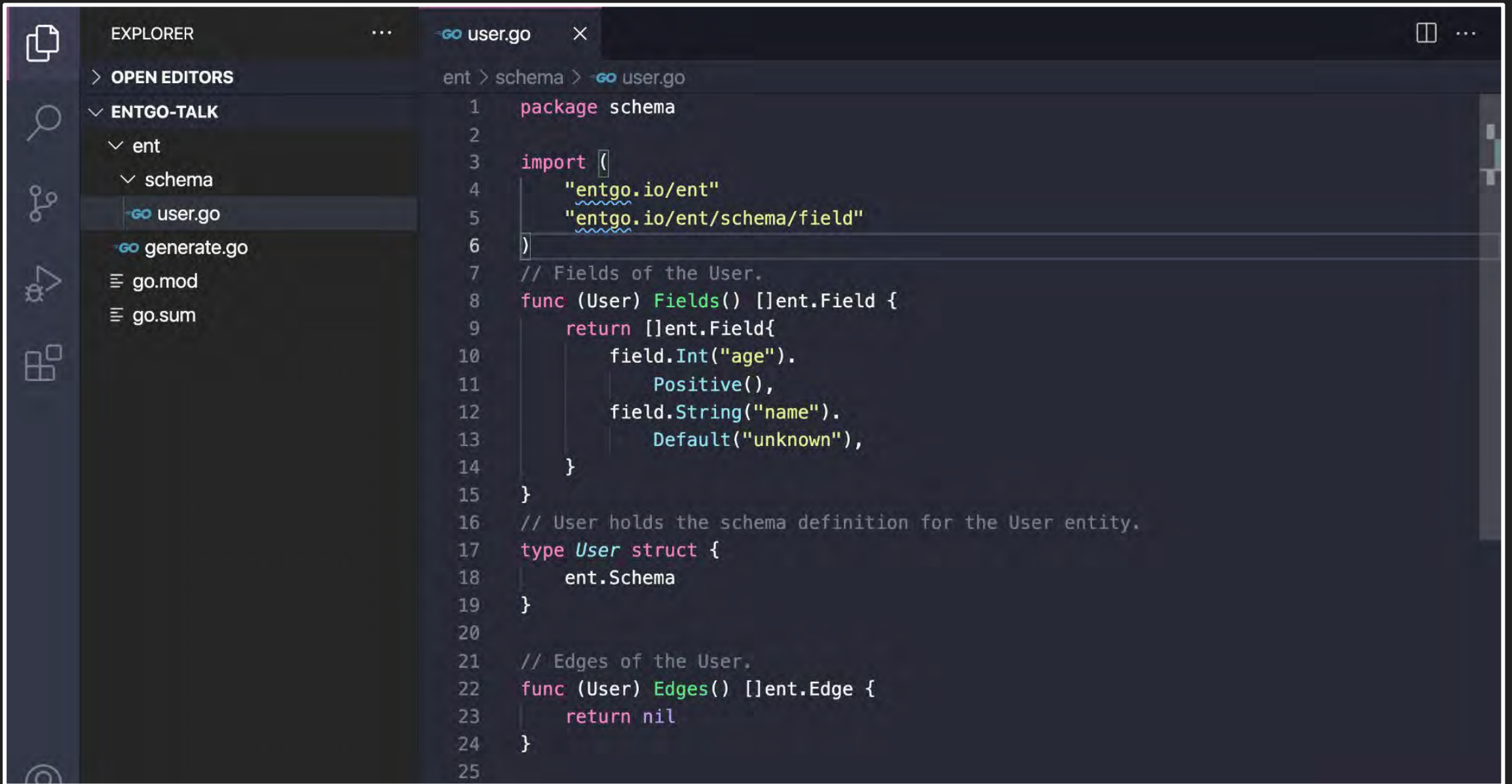
```
1  module entgo-talk
2
3  go 1.15
4
```



```
go run entgo.io/ent/cmd/ent init User
```

```
EXPLORER
  > OPEN EDITORS
  < ENTGO-TALK
    < ent
      < schema
        -GO user.go
        -GO generate.go
      go.mod
      go.sum

-GO user.go
ent > schema > -GO user.go
1 package schema
2
3 import "entgo.io/ent"
4
5 // User holds the schema definition for the User entity.
6 type User struct {
7     ent.Schema
8 }
9
10 // Fields of the User.
11 func (User) Fields() []ent.Field {
12     return nil
13 }
14
15 // Edges of the User.
16 func (User) Edges() []ent.Edge {
17     return nil
18 }
```



The image shows a screenshot of an IDE with a dark theme. On the left, the Explorer sidebar shows a project structure with folders 'ent' and 'schema', and files 'user.go', 'generate.go', 'go.mod', and 'go.sum'. The main editor window displays the content of 'user.go' with the following Go code:

```
1 package schema
2
3 import (
4     "entgo.io/ent"
5     "entgo.io/ent/schema/field"
6 )
7 // Fields of the User.
8 func (User) Fields() []ent.Field {
9     return []ent.Field{
10         field.Int("age").
11             Positive(),
12         field.String("name").
13             Default("unknown"),
14     }
15 }
16 // User holds the schema definition for the User entity.
17 type User struct {
18     ent.Schema
19 }
20
21 // Edges of the User.
22 func (User) Edges() []ent.Edge {
23     return nil
24 }
25
```

```
1 package schema
2
3 import (
4     "entgo.io/ent"
5     "entgo.io/ent/schema/field"
6 )
7 // Fields of the User.
8 func (User) Fields() []ent.Field {
9     return []ent.Field{
10         field.Int("age").
11             Positive(),
12         field.String("name").
13             Default("unknown"),
14     }
15 }
16 // User holds the schema definition for the User entity.
17 type User struct {
18     ent.Schema
19 }
20
21 // Edges of the User.
22 func (User) Edges() []ent.Edge {
23     return nil
24 }
25
```

A terminal window with a dark background and a white border. At the top left, there are three colored circles: red, yellow, and green. Below them, the text "go generate ./ent" is displayed in a light green monospace font.

```
go generate ./ent
```


ENTGO-TALK

ent

> enttest

> hook

> migrate

> predicate

> runtime

schema

GO user.go

> user

GO client.go

GO config.go

GO context.go

GO ent.go

GO generate.go

GO mutation.go

GO runtime.go

GO tx.go

GO user_create.go

GO user_delete.go

GO user_query.go

GO user_update.go

GO user.go

```
1 package schema
2
3 import (
4     "entgo.io/ent"
5     "entgo.io/ent/schema/field"
6 )
7 // Fields of the User.
8 func (User) Fields() []ent.Field {
9     return []ent.Field{
10         field.Int("age").
11             Positive(),
12         field.String("name").
13             Default("unknown"),
14     }
15 }
16 // User holds the schema definition for the User entity.
17 type User struct {
18     ent.Schema
19 }
20
21 // Edges of the User.
22 func (User) Edges() []ent.Edge {
23     return nil
24 }
25
```


Creating Entities

-go Client.go

```
1 package main
2
3 import (
4     "context"
5     "log"
6
7     "entgo-talk/ent"
8
9     _ "github.com/mattn/go-sqlite3"
10 )
11
12 func main() {
13     client, err := ent.Open("sqlite3", "file:ent?mode=memory&cache=shared&_fk=1")
14     if err != nil {
15         log.Fatalf("failed opening connection to sqlite: %v", err)
16     }
17     defer client.Close()
18     // Run the auto migration tool.
19     if err := client.Schema.Create(context.Background()); err != nil {
20         log.Fatalf("failed creating schema resources: %v", err)
21     }
22 }
23
24
```

```

8
9     _ "github.com/matttn/go-sqlite3"
10 )
11
12 func main() {
13     client, err := ent.Open("sqlite3", "file:ent?mode=memory&cache=shared&_fk=1")
14     if err != nil {
15         log.Fatalf("failed opening connection to sqlite: %v", err)
16     }
17     defer client.Close()
18     // Run the auto migration tool.
19     if err := client.Schema.Create(context.Background()); err != nil {
20         log.Fatalf("failed creating schema resources: %v", err)
21     }
22 }
23
24 func CreateUser(ctx context.Context, client *ent.Client) (*ent.User, error) {
25     u, err := client.User.
26         Create().
27         SetAge(30).
28         SetName("a8m").
29         Save(ctx)
30     if err != nil {
31         return nil, fmt.Errorf("failed creating user: %w", err)
32     }
33     log.Println("user was created: ", u)
34     return u, nil
35 }

```

```
24 func CreateUser(ctx context.Context, client *ent.Client) (*ent.User, error) {
25     u, err := client.User.
26         Create().
27         SetAge(30).
28         SetName("a8m").
29         Save(ctx)
30     if err != nil {
31         return nil, fmt.Errorf("failed creating user: %w", err)
32     }
33     log.Println("user was created: ", u)
34     return u, nil
35 }
```

Querying Entities

GO query.go

```
1 package main
2
3 import (
4     "log"
5
6     "<project>/ent"
7     "<project>/ent/user"
8 )
9
10 func QueryUser(ctx context.Context, client *ent.Client) (*ent.User, error) {
11     u, err := client.User.
12         Query().
13         Where(user.NameEQ("a8m")).
14         // `Only` fails if no user found,
15         // or more than 1 user returned.
16         Only(ctx)
17     if err != nil {
18         return nil, fmt.Errorf("failed querying user: %w", err)
19     }
20     log.Println("user returned: ", u)
21     return u, nil
22 }
```

And a lot more!

Tutorials

Setting Up | ent

entgo.io/docs/tutorial-setup/

Help us improve by taking [our user survey](#).

ent. Docs Tutorials GoDoc Blog

English

Search

First Steps

- Setting Up
- Query and Mutation

GraphQL Basics

- Introduction
- Relay Node Interface
- Relay Cursor Connections
- Field Collection
- Transactional Mutations
- Mutation Inputs

Setting Up

This guide is intended for first-time users who want instructions on how to set up an Ent project from scratch. Before we get started, make sure you have the following prerequisites installed on your machine.

Prerequisites

- Go
- Docker (optional)

After installing these dependencies, create a directory for the project and initialize a Go module:

```
mkdir todo
cd $_
go mod init todo
```

Installation

Run the following Go commands to install Ent, and tell it to initialize the project structure along with a `Todo` schema.

```
go get entgo.io/ent/cmd/ent
```

Prerequisites

Installation

Code Generation

Create a Test Case

Query and Mutation | ent

entgo.io/docs/tutorial-todo-crud

Help us improve by taking [our user survey](#).

ent. Docs Tutorials GoDoc Blog

English

Search

First Steps

- Setting Up
- Query and Mutation**

GraphQL Basics

- Introduction
- Relay Node Interface
- Relay Cursor Connections
- Field Collection
- Transactional Mutations
- Mutation Inputs

Query and Mutation

After setting up our project, we're ready to create our Todo list and query it.

- Create a Todo
- Add Fields To The Schema
- Add Edges To The Schema
- Connect 2 Todos
- Query Todos

Create a Todo

Let's create a Todo in our testable example. We do it by adding the following code to `example_test.go`:

```
func Example_Todo() {
    // ...
    task1, err := client.Todo.Create().Save(ctx)
    if err != nil {
        log.Fatalf("failed creating a todo: %v", err)
    }
    fmt.Println(task1)
    // Output:
    // Todo(id=1)
}
```

Running `go test` should pass successfully.

Add Fields To The Schema

As you can see, our Todos are too boring as they contain only the `ID` field. Let's improve this example by adding multiple fields to the schema in `todo/ent/schema/todo.go`:

<https://entgo.io/docs/tutorial-todo-crud>

Query and Mutation | ent

entgo.io/docs/tutorial-todo-crud

Help us improve by taking [our user survey](#).

ent. Docs Tutorials GoDoc Blog

English

Search

First Steps

- Setting Up
- Query and Mutation**

GraphQL Basics

- Introduction
- Relay Node Interface
- Relay Cursor Connections
- Field Collection
- Transactional Mutations
- Mutation Inputs

Query and Mutation

After setting up our project, we're ready to create our Todo list and query it.

Create a Todo

Let's create a Todo in our testable example. We do it by adding the following code to `example_test.go`:

```
func Example_Todo() {
    // ...
    task1, err := client.Todo.Create().Save(ctx)
    if err != nil {
        log.Fatalf("failed creating a todo: %v", err)
    }
    fmt.Println(task1)
    // Output:
    // Todo(id=1)
}
```

Running `go test` should pass successfully.

Add Fields To The Schema

As you can see, our Todos are too boring as they contain only the `ID` field. Let's improve this example by adding multiple fields to the schema in `todo/ent/schema/todo.go`:

Create a Todo

- Add Fields To The Schema
- Add Edges To The Schema
- Connect 2 Todos
- Query Todos

<https://entgo.io/docs/tutorial-todo-crud>

Introduction | ent

entgo.io/docs/tutorial-todo-gql

Help us improve by taking [our user survey](#).

ent. Docs Tutorials GoDoc Blog

First Steps

- Setting Up
- Query and Mutation

GraphQL Basics

- Introduction**
- Relay Node Interface
- Relay Cursor Connections
- Field Collection
- Transactional Mutations
- Mutation Inputs

Introduction

In this section, we will learn how to connect Ent to GraphQL. If you're not familiar with GraphQL, it's recommended to go over its [introduction guide](#) before going over this tutorial.

Clone the code (optional)

The code for this tutorial is available under github.com/a8m/ent-graphql-example, and tagged (using Git) in each step. If you want to skip the basic setup and start with the initial version of the GraphQL server, you can clone the repository and checkout `v0.1.0` as follows:

```
git clone git@github.com:a8m/ent-graphql-example.git
git checkout v0.1.0
cd ent-graphql-example
go run ./cmd/todo/
```

Basic Skeleton

[gqlgen](#) is a framework for easily generating GraphQL servers in Go. In this tutorial, we will review Ent's official integration with it.

This tutorial begins where the previous one ended (with a working Todo-list schema). We start by creating a simple GraphQL schema for our todo list, then install the [99designs/gqlgen](#) package and configure it. Let's create a file named `todo.graphql` and paste the following:

<https://entgo.io/docs/tutorial-todo-gql>

Relay Node Interface | ent

entgo.io/docs/tutorial-todo-gql-node

Help us improve by taking [our user survey](#).

ent. Docs Tutorials GoDoc Blog

English

Search

First Steps

- Setting Up
- Query and Mutation

GraphQL Basics

- Introduction
- Relay Node Interface**
- Relay Cursor Connections
- Field Collection
- Transactional Mutations
- Mutation Inputs

Relay Node Interface

Implementation

Query Nodes

In this section, we continue the [GraphQL example](#) by explaining how to implement the [Relay Node Interface](#). If you're not familiar with the Node interface, read the following paragraphs that were taken from [relay.dev](#):

To provide options for GraphQL clients to elegantly handle for caching and data refetching GraphQL servers need to expose object identifiers in a standardized way. In the query, the schema should provide a standard mechanism for asking for an object by ID. In the response, the schema provides a standard way of providing these IDs.

We refer to objects with identifiers as "nodes". An example of both of those is the following query:

```
{
  node(id: "4") {
    id
    ... on User {
      name
    }
  }
}
```

Clone the code (optional)

The code for this tutorial is available under [github.com/a8m/ent-graphql-example](#), and tagged

<https://entgo.io/docs/tutorial-todo-gql-node>

More Docs!

ent. Docs Tutorials GoDoc Blog

Help us improve by taking [our user survey](#).

Quick Introduction

Schema

Introduction

Fields

Edges

Indexes

Mixin

Annotations

Code Generation

Introduction

CRUD API

Graph Traversal

Eager Loading

Hooks

Privacy

Transactions

Predicates

Aggregation

Paging And Ordering

Introduction

Quick Summary

It's Just Another ORM.

Schema describes the definition of one entity type in the graph, like `User` or `Group`, and can contain the following configurations:

- Entity fields (or properties), like: name or age of a `User`.
- Entity edges (or relations), like: `User`'s groups, or `User`'s friends.
- Database specific options, like: indexes or unique indexes.

Here's an example of a schema:

```
package schema

import (
    "entgo.io/ent"
    "entgo.io/ent/schema/field"
    "entgo.io/ent/schema/edge"
    "entgo.io/ent/schema/index"
)

type User struct {
    ent.Schema
}

func (User) Fields() []ent.Field {
    return []ent.Field{
        field.Int("age"),
    }
}
```

The screenshot shows a web browser window displaying the documentation for 'ent' database migration. The page title is 'Database Migration'. The main content area includes a section for 'Auto Migration #' with a code snippet and a section for 'Drop Resources' with a code snippet. A sidebar on the left contains a navigation menu with categories like 'Hooks', 'Privacy', 'Transactions', 'Predicates', 'Aggregation', 'Paging And Ordering', 'Migration', 'Supported Dialects', 'Misc', 'External Templates', 'GraphQL Integration', 'sql.DB Integration', 'Testing', 'FAQ', 'Generating Schemas', 'Feature Flags', 'Translations', 'Contributors', and 'Writing Docs'. The 'Migration' category is expanded, showing 'Database Migration' as the selected item. The browser's address bar shows 'entgo.io/docs/migrate' and the page is viewed in Incognito mode.

ent. Docs Tutorials GoDoc Blog

Help us improve by taking [our user survey](#).

Auto Migration
Drop Resources
Universal IDs
Offline Mode
Foreign Keys
Migration Hooks

Database Migration

The migration support for `ent` provides the option for keeping the database schema aligned with the schema objects defined in `ent/migrate/schema.go` under the root of your project.

Auto Migration

Run the auto-migration logic in the initialization of the application:

```
if err := client.Schema.Create(ctx); err != nil {
    log.Fatalf("failed creating schema resources: %v", err)
}
```

`Create` creates all database resources needed for your `ent` project. By default, `Create` works in an "append-only" mode; which means, it only creates new tables and indexes, appends columns to tables or extends column types. For example, changing `int` to `bigint`.

What about dropping columns or indexes?

Drop Resources

`WithDropIndex` and `WithDropColumn` are 2 options for dropping table columns and indexes.

```
package main
```


Graph Traversal | ent ent - pkg.go.dev entgo.io/docs/traversals

ent. Docs Tutorials GoDoc Blog English Search

- Getting Started
 - Quick Introduction
- Schema
 - Introduction
 - Fields
 - Edges
 - Indexes
 - Mixin
 - Annotations
- Code Generation
 - Introduction
 - CRUD API
 - Graph Traversal**
 - Eager Loading
 - Hooks
 - Privacy
 - Transactions
 - Predicates
 - Aggregation
 - Paging And Ordering

Graph Traversal

For the purpose of the example, we'll generate the following graph:

```

graph TD
    G6((6 Group)) -- users --> U1((1 User))
    G6 -- users --> U7((7 User))
    U1 -- owner --> P3((3 Pet))
    U1 -- owner --> P4((4 Pet))
    U2((2 User)) -- owner --> P5((5 Pet))
    U7 -- friends --> U1
    P3 -- friends --> P4
    P4 -- friends --> P5
    P5 -- friends --> P3
  
```

The first step is to generate the 3 schemas: `Pet`, `User`, `Group`.

```
go run entgo.io/ent/cmd/ent init Pet User Group
```

Add the necessary fields and edges for the schemas:

```
ent/schema/pet.go
```

```
// Pet holds the schema definition for the Pet entity.
type Pet struct {
    ent.Schema
}
```

Testing | ent ent - pkg.go.dev + entgo.io/docs/testing ☆ Incognito

Help us improve by taking [our user survey](#).

ent. Docs Tutorials GoDoc Blog 🌐 English 🌙 🔍 Search

- Hooks
- Privacy
- Transactions
- Predicates
- Aggregation
- Paging And Ordering
- Migration ▼
 - Database Migration
 - Supported Dialects
- Misc ▼
 - External Templates
 - GraphQL Integration
 - sql.DB Integration
 - Testing**
 - FAQ
 - Generating Schemas
 - Feature Flags
 - Translations
 - Contributors
 - Writing Docs

Testing

If you're using `ent.Client` in your unit-tests, you can use the generated `enttest` package for creating a client and auto-running the schema migration as follows:

```
package main

import {
    "testing"

    "<project>/ent/enttest"

    _ "github.com/mattn/go-sqlite3"
}

func TestXXX(t *testing.T) {
    client := enttest.Open(t, "sqlite3", "file:ent?mode=memory&cache=shared&_fk=1"
    defer client.Close()
    // ...
}
```

In order to pass functional options to `Open`, use `enttest.Option`:

```
func TestXXX(t *testing.T) {
    opts := []enttest.Option{
        enttest.WithOptions(ent.Log(t.Log)),
        enttest.WithMigrateOptions(migrate.WithGlobalUniqueID(true)),
    }
    client := enttest.Open(t, "sqlite3", "file:ent?mode=memory&cache=shared&_fk=1"
```

GoDoc for Devs

Relay Node Interface | ent x ent · pkg.go.dev x +

pkg.go.dev/entgo.io/ent ☆ Incognito

Black Lives Matter Support the Equal Justice Initiative

GO Search for a package Why Go Getting Started Discover Packages About

Discover Packages > entgo.io/ent

ent package module

Version: v0.8.0 LATEST | Published: Apr 14, 2021 | License: Apache-2.0 | Imports: 6 | Imported by: 115

Jump to ...

README

ent - An Entity Framework F...
Quick Installation
Docs and Support
Join the ent Community
About the Project
License

Documentation
Source Files
Directories

README

ent - An Entity Framework For Go

Follow @entgo_io

English | 中文

Simple, yet powerful entity framework for Go, that makes it easy to build and maintain applications with large data-models.

- Schema As Code - model any database schema as Go objects.

Expand ▾

<> Documentation

Overview

Package ent is the interface between end-user schemas and entc (ent codegen).

Details Learn more


- Valid go.mod file
- Redistributable license
- Tagged version
- Stable version

Repository
github.com/ent/ent

But it's all about
community

Testing | ent x ent · pkg.go.dev x https://twitter.com/entgo_io x The Ent Newsletter | Revue x Join us on Slack | ent x

getrevue.co/profile/ent



The Ent Newsletter

An entity framework for Go

65 SUBSCRIBERS 2 ISSUES

Subscribe to our newsletter

By subscribing, you agree with Revue's [Terms of Service](#) and [Privacy Policy](#).

Your email address... [Subscribe now](#)

Previous issues


Language	Transla
kazutoshi takaki (kazutaki)	Japanese
hengxi (hengxi)	Chinese Simplified
ten Ogaki (tenogaki)	Japanese
noriyuki (noriyuki)	Japanese, Chinese Si...
rensong	Chinese Simplified
naoto (naoto_01)	Japanese
otani	Chinese Simplified
ipuz200	Japanese
gen. Takahashi (gen.takahashi)	Japanese
sh	Japanese
shiro Suzuki (shirosuzuki)	Japanese

The Ent Newsletter #2 - v0.8.0 and updates

Apr 21, 2021 #2

The Ent Newsletter #1 - v0.7.0 updates

Mar 15, 2021 #1



Blog | ent ent - pkg.go.dev +

entgo.io/blog ☆ Incognito

Help us improve by taking [our user survey](#).


ent. Docs Tutorials GoDoc Blog 🌐 ✉ 🐦 🗨 🇬🇧 English 🌙 🔍 Search

Recent posts

- Announcing the "Schema Import Initiative" and protoc-gen-ent
- Generate a fully-working Go gRPC server in two minutes with Ent
- Announcing Edge-field Support in v0.7.0
- Introducing ent

Announcing the "Schema Import Initiative" and protoc-gen-ent

May 4, 2021 · 3 min read

 **Rotem Tamir**

Migrating to a new ORM is not an easy process, and the transition cost can be prohibitive to many organizations. As much as we developers are enamoured by "Shiny New Things", the truth is that we rarely get a chance to work on a truly "green-field" project. Most of our careers, we operate in contexts where many technical and business constraints (a.k.a legacy systems) dictate and limit our options for moving forward. Developers of new technologies that want to succeed must offer interoperability capability and integration paths to help organizations seamlessly transition to a new way of solving an existing problem.

To help lower the cost of transitioning to Ent (or simply experimenting with it), we have started the **"Schema Import Initiative"** to help support many use cases for generating Ent schemas from external resources. The centrepiece of this effort is the `schemast` package ([source code](#), [docs](#)) which enables developers to easily write programs that generate and manipulate Ent schemas. Using this package, developers can program in a high-level API, relieving them from worrying about code parsing and AST manipulations.




Testing | ent x ent · pkg.go.dev x ent (@entgo_io) / Twitter x The Ent Newsletter | Revue x Join us on Slack | ent x

twitter.com/entgo_io


Search Twitter

New to Twitter?
Sign up now to get your own personalized timeline!
[Sign up](#)


You might like

-  **Paul Ryan**
@ryanpham92 [Follow](#)
-  **shinden (新田)**
@t_shinden [Follow](#)
-  **Jaime Blasco**
@jaimeblasco [Follow](#)

[Show more](#)

 **ent**
17 Tweets


ent. An entity framework for Go
Simple, yet powerful ORM for modeling and querying data.


 [Follow](#)

ent
@entgo_io
An entity framework for Go
[entgo.io](#) Joined February 2021
0 Following 113 Followers

Tweets Tweets & replies Media Likes

Pinned Tweet

 **ent** @entgo_io · Mar 10
Subscribe to our newsletter for occasional updates, release notes and content:

 The Ent Newsletter - Revue
An entity framework for Go
[getrevue.co](#)

Join us on Slack | ent x ent · pkg.go.dev x ent (@entgo_io) / Twitter x The Ent Newsletter | Revue x Join us on Slack | ent x +

entgo.io/docs/slack ☆ Incognito

Help us improve by taking [our user survey](#).

ent. Docs Tutorials GoDoc Blog

Hooks
Privacy
Transactions
Predicates
Aggregation
Paging And Ordering
Migration
Database Migration
Supported Dialects
Misc
External Templates
GraphQL Integration
sql.DB Integration
Testing
FAQ
Generating Schemas
Feature Flags
Translations

Join us on Slack

Ent maintainers, contributors and users hang out on the #ent channel in the Gophers Slack workspace.

To join the discussion:

1. Sign up to the Gophers Slack workspace via the [invite link](#).
2. Join the #ent channel manually or via [this link](#). (it will work only if you are logged in to slack from your browser)

Previous
« [Writing Docs](#)

Call to Action

Experiment
with Tools

Call to Action

Experiment
with Tools

Join Ent
Community

Call to Action

**Experiment
with Tools**

**Join Ent
Community**

Call to Action

**Contribute
Back**

About Speaker

Twitter: [@DmitryVinnik](https://twitter.com/DmitryVinnik)

Blog: dvinnik.dev

LinkedIn: [in/dmitry-vinnik/](https://www.linkedin.com/in/dmitry-vinnik/)

Email: dmitry@dvinnik.dev