# Building ~~stateful~~ real apps on serverless

Tirumarai Selvan, Hasura

Horse Shoe bend - Page, Arizona

# I’m Tiru

**Hasura is :**

- GraphQL on Postgres
- Serverless business logic
- Postgres as event source

**I do:**

- Product development
- Product management
- Developer advocacy

https://hasura.io

@Tirumarai

A real app always has STATE!

Otherwise.
It is only.
Heating the place up!

# global warming

# Examples of few simple apps

1. Get an image as input, resize and store it somewhere. That's state.

2. Register a user, and create a list of Todo items

3. Given a city, retrieve the current weather.

Aha! No state?

# Given a city, retrieve the current weather

There is no state, only if you are NOT Yahoo Weather

What we have done is delegated state to a black-box

Real apps have state! All we can do is delegate, delegate and delegate.

# Patterns for delegating state

1) Send full context in event payload
2) Use BaaS aka managed services
3) Run long running “state planes”
4) Build workflows using state machine pattern
5) EASIEST: Don’t delegate state (w00t!)

# Pattern #1: Send context with event

Send as much context as possible with an event

Obviate the need for database read

**Event #1**

```
{ "id": “42” }
```

✗

**Event #2**

```
"user_info": {
   "user-id": "1"
},
"op": "UPDATE",
"data": {
   "old":{
      "id":"42",
      "name": "jane"
      },
    "new": {
      "id":"42",
      "name": "john doe"
      }
}
```

✓

# Example #1: Send context with event

Send email on user creation

```
"user_info": {
    "user-id": "1",
    "name": "Jane",
    "address": "Ch. S",
},
"login": "Facebook",
"created": "14-02-19"
"utm": {
    "source": "twit",
    "campaign": "ad"
}
```

@Tirumarai

# Pattern #2: Use BaaS/managed services

Use as much BaaS / managed services as you can afford.

Let someone else deal with state

E.g.

Identity: Auth0
Media tools: Cloudinary
Search: Algolia
Notifications: Twilio

# Example #2: Use BaaS/managed services

Index items for search

```
const algoliasearch = require('algoliasearch');

exports.function = async (req, res) => {
  const { event: { op, data } } = req.body;

  var client = algoliasearch(ALGOLIA_APP_ID, ALGOLIA_ADMIN_API_KEY);
  var index = client.initIndex('demo_serverless_etl_app');

  index.addObjects([data.new], function(err, content) {
      console.log(content);
    });
};
```

# Pattern #3: Run long running “state planes”

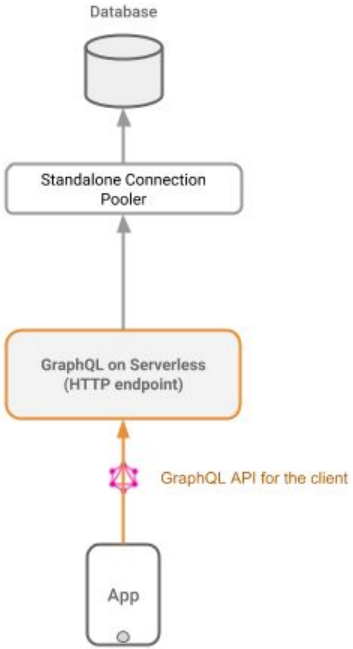
Sometimes, a long running proxy can manage state for your stateless infra aka “state plane”.

Very custom to problem.

E.g. Standalone database connection poolers can provide a “managed connection pool” for your functions.

# Example #3: Run long running “state planes”

Running a GraphQL service on serverless




https://medium.com/open-graphql/scaling-rdbms-for-graphql-backends-on-serverless-980ef24af171

# Pattern #4 : Build workflows using state-machine pattern

Workflows are one of the most common stateful tasks.

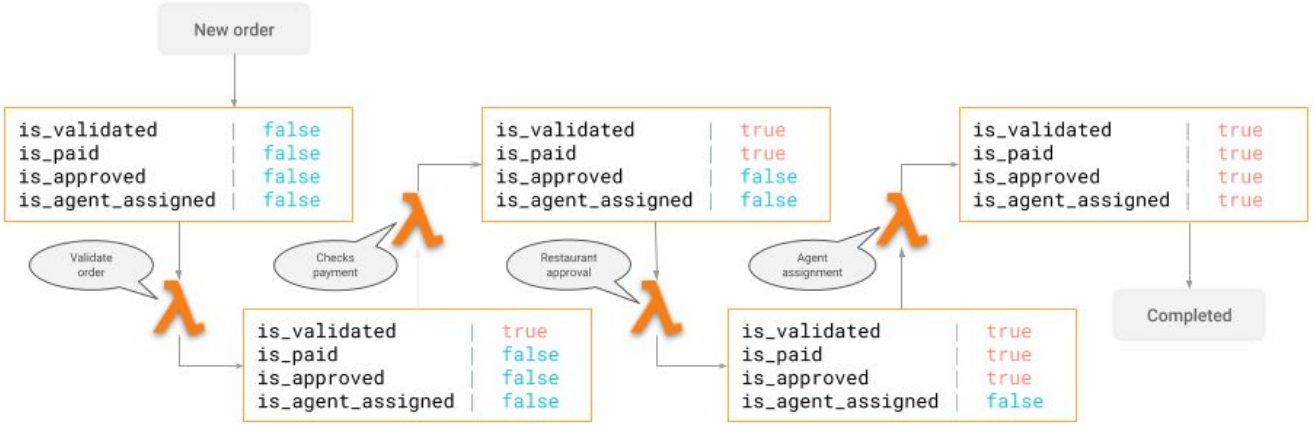
State machines model workflows perfectly.

| Column | Value |
|---|---|
| id | a705a43d-c55b-4a8a-ac16-80679bd02404 |
| created_at | 2018-09-23 13:24:32.159024+00 |
| user_id | eca786b6-73ae-44c7-b34f-1757af554464 |
| is_validated | false |
| is_paid | false |
| is_approved | false |
| is_agent_assigned | false |

@Tirumarai

# Pattern #4 : Build workflows using state-machine pattern

Hasura Event Triggers




https://blog.hasura.io/building-stateful-apps-using-serverless-postgres-and-hasura/

# Pattern #5 : Use fast databases (EASY)

Just use databases, normally!

The only downsides appear at scale: performance and resource utilization.

Use fast databases with lightweight connection management.

# Pattern #5 : Use fast databases (EASY)

1. Reuse connections using vendor best practices.
2. Create state in entirety at the beginning (may not work for transactional data)
3. Try to use an ORM free approach.
4. Use connection poolers like pgBouncer for scale.

# What is stateless then?

A new perspective for serverless

# Stateless == Idempotency?

What people usually mean by stateless is idempotency

Stateless => Idempotency, Idempotency => More than stateless

If we can ensure idempotency, our stateful function seems like a stateless function!

Managed services should be idempotent for use with serverless

# Caveats

# Good enough, but not everything

1) Can’t reuse state

2) Can’t share state (no global variables)

3) Can’t address each other (no  DNS names)

Demo time :)

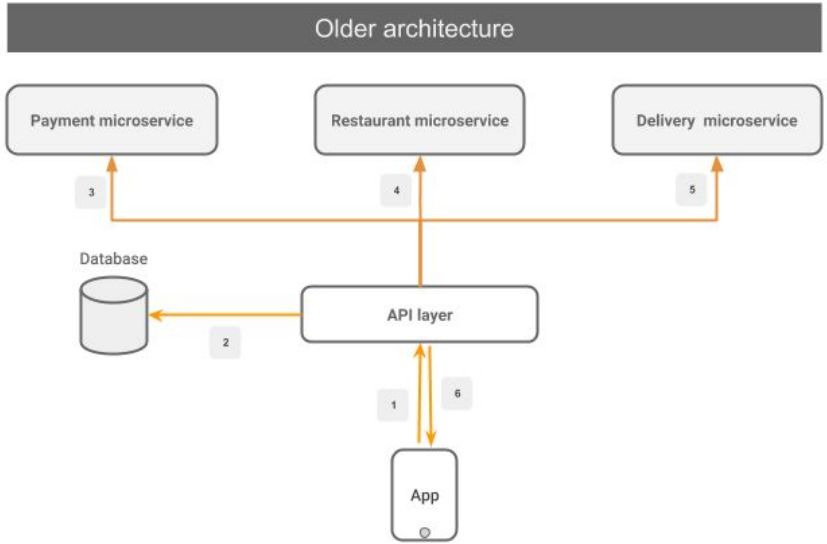
# The 3factor app

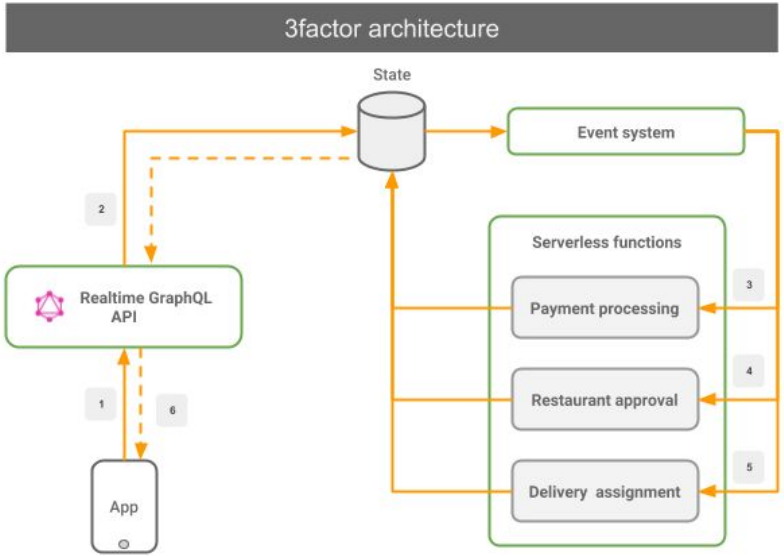
1. Realtime GraphQL for async consumption of data.

2. Reliable eventing system for triggering actions.

3. Serverless functions for scale and agility.

# The 3factor app




https://3factor.app

# What's the catch? Transactions!

Transactions are fundamentally sync operations.

Can we ensure transactionality across multiple functions?

Not, today. Workaround: make things idempotent (handle stale data gracefully, yeah right! )

# Let’s talk about Haskell :)

Composition and purity are the fundamental building blocks of functional programming.

IO actions can be composed and finally run in ONE shot.

Can we compose serverless functions, please?

# How to get started?

# Incremental adoption for brownfield projects

Move tiny async things inside your sync functions outside.

Reduce lines of code in critical path.

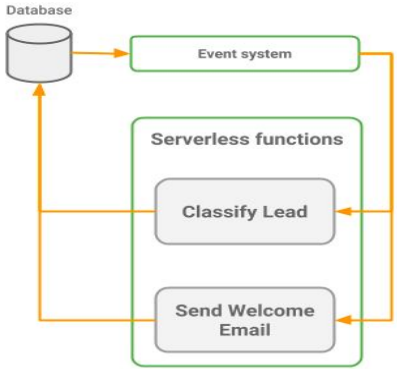
Decouple functionality

# An example: Before and After

```
def create_user(info) :

    SalesServices.classify_lead(info)

    u = User.create(info)

    EmailServices.send_welcome_email(u)
```

**Vs**

```
def create_user(info) :

    u = User.create(info)
```



# Hasura Event Triggers

Postgres as Event Source

Atomic events - never miss

Reliable delivery with retries

2 min demo!

# Thank you!

Tirumarai Selvan
@Tirumarai

hasura.io

github.com/hasura/graphql-engine


Questions?