SOA with tRPC

Service Oriented Architecture

Néstor López

Software Engineer

CoruñaWTF

12 Dec 2024 @ A Coruña

Néstor López

Software Engineer at Fever
Maintainer of trpc.io & create-t3-app

GitHub: nsttt

Twitter: @nstlopez

BlueSky: https://nst.blue/

¿Que es SOA ?

¿Que es SOA ?

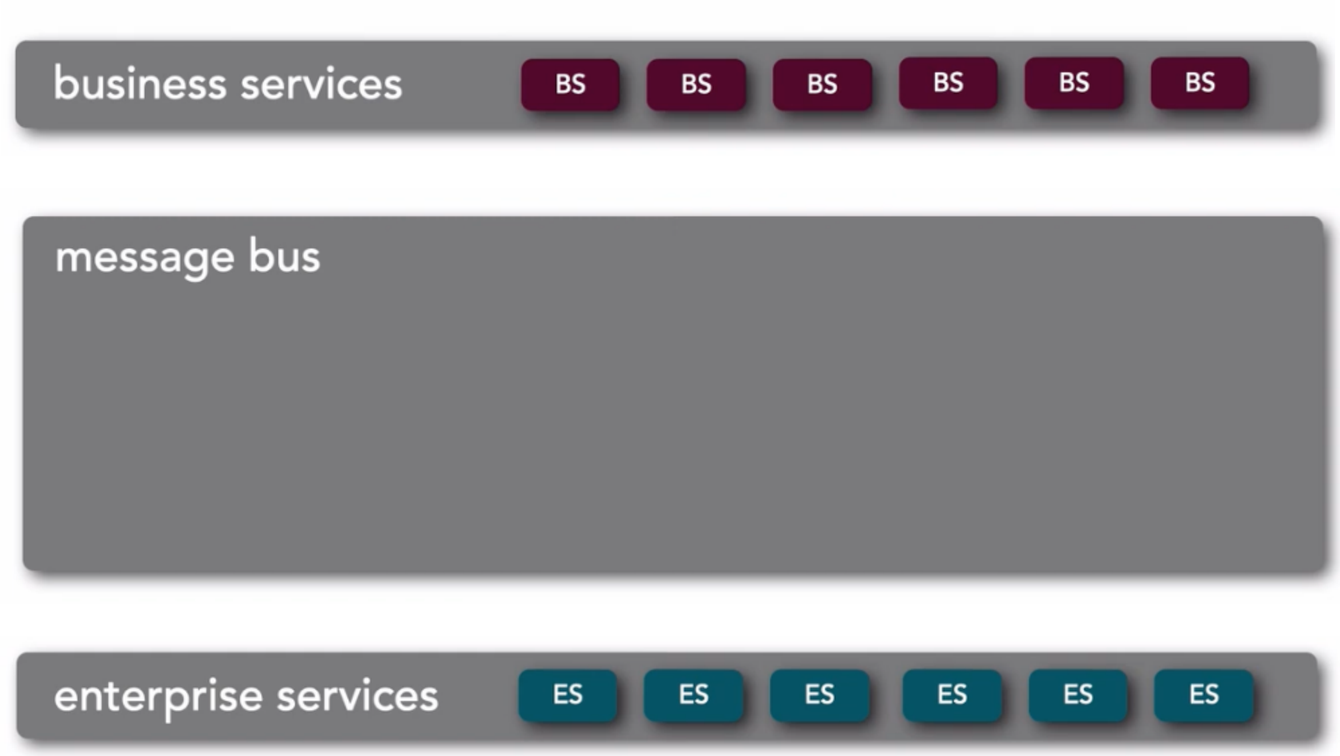
- Nacida alrededor de 2001.
- Funcionalidades centrales compartidas entre sistemas.
- Menos duplicidad entre sistemas.
- Basada pricipalmente en REST.
- REST !== Service Oriented.
- Tiene una taxonomía estricta de tipos de servicios.
- Todo empieza con un servicio de negocio.

¿Que es SOA ?

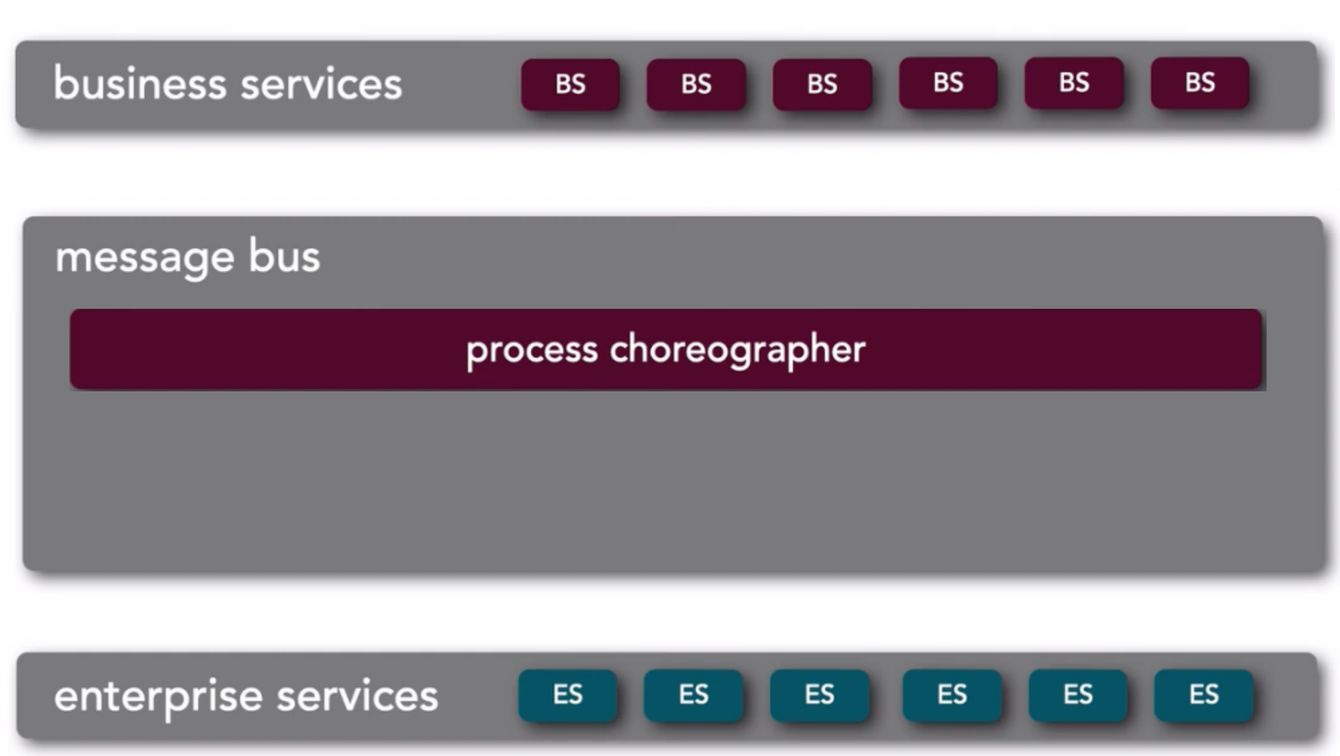
¿Que es SOA ?


business services
BS
BS
BS
BS
BS
BS
enterprise services
ES
ES
ES
ES
ES
ES

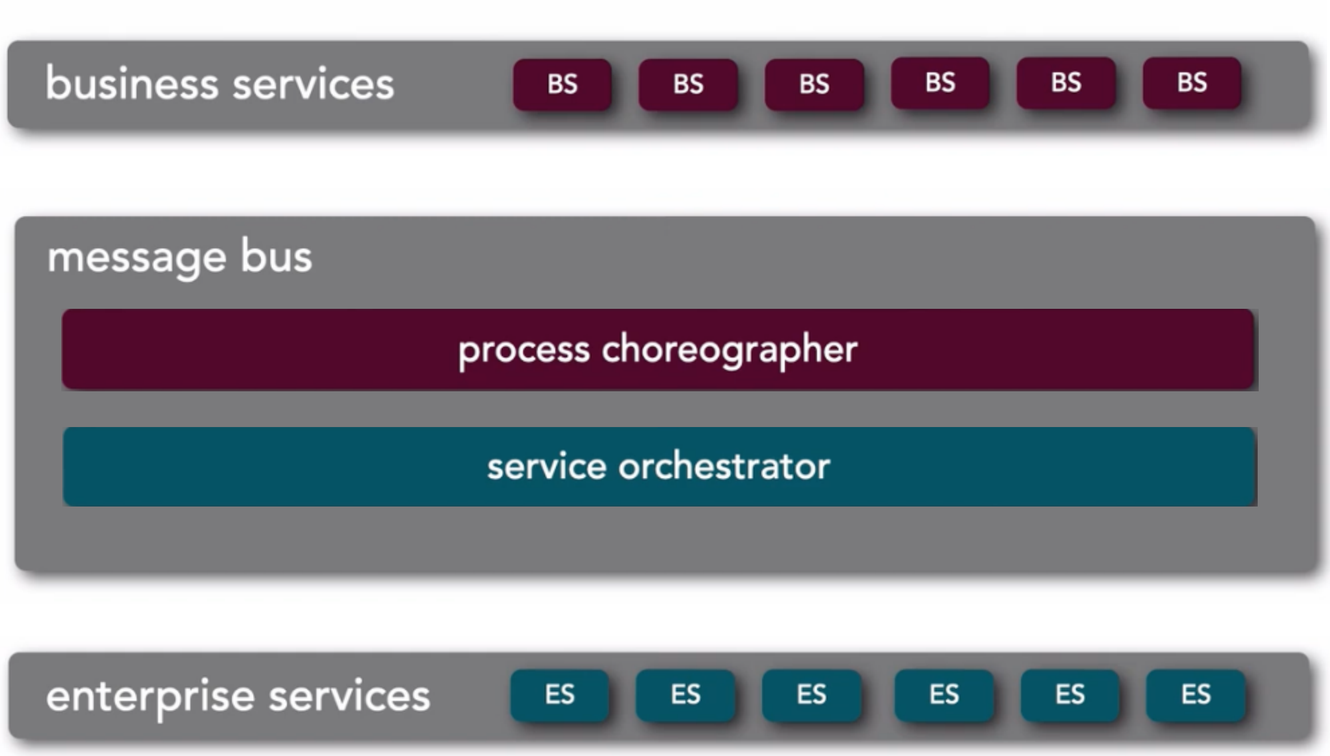
¿Que es SOA ?


business services
BS
BS
BS
BS
BS
BS
message bus
enterprise services
ES
ES
ES
ES
ES
ES

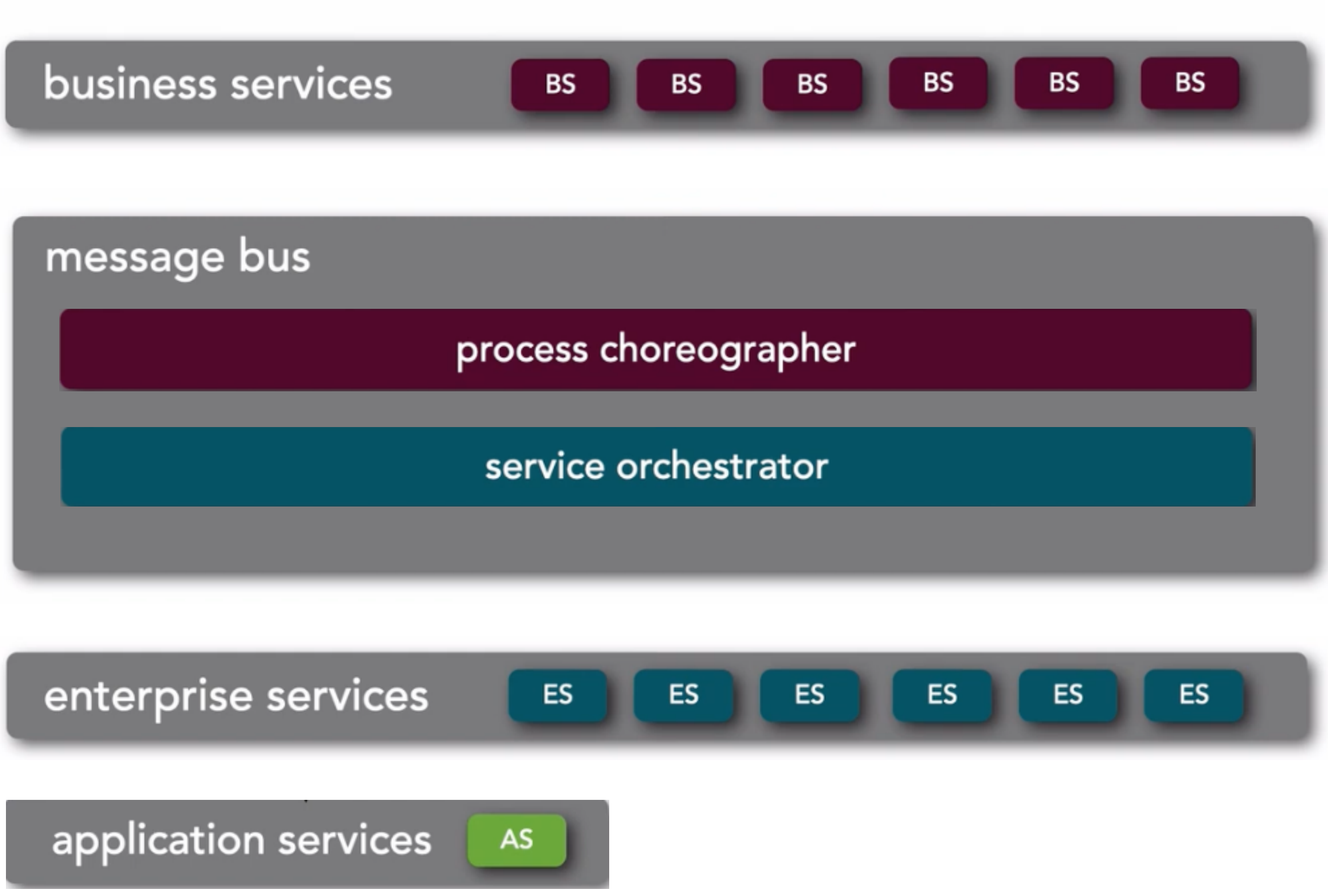
¿Que es SOA ?


business services
BS
BS
BS
BS
BS
BS
message bus
process choreographer
enterprise services
ES
ES
ES
ES
ES
ES

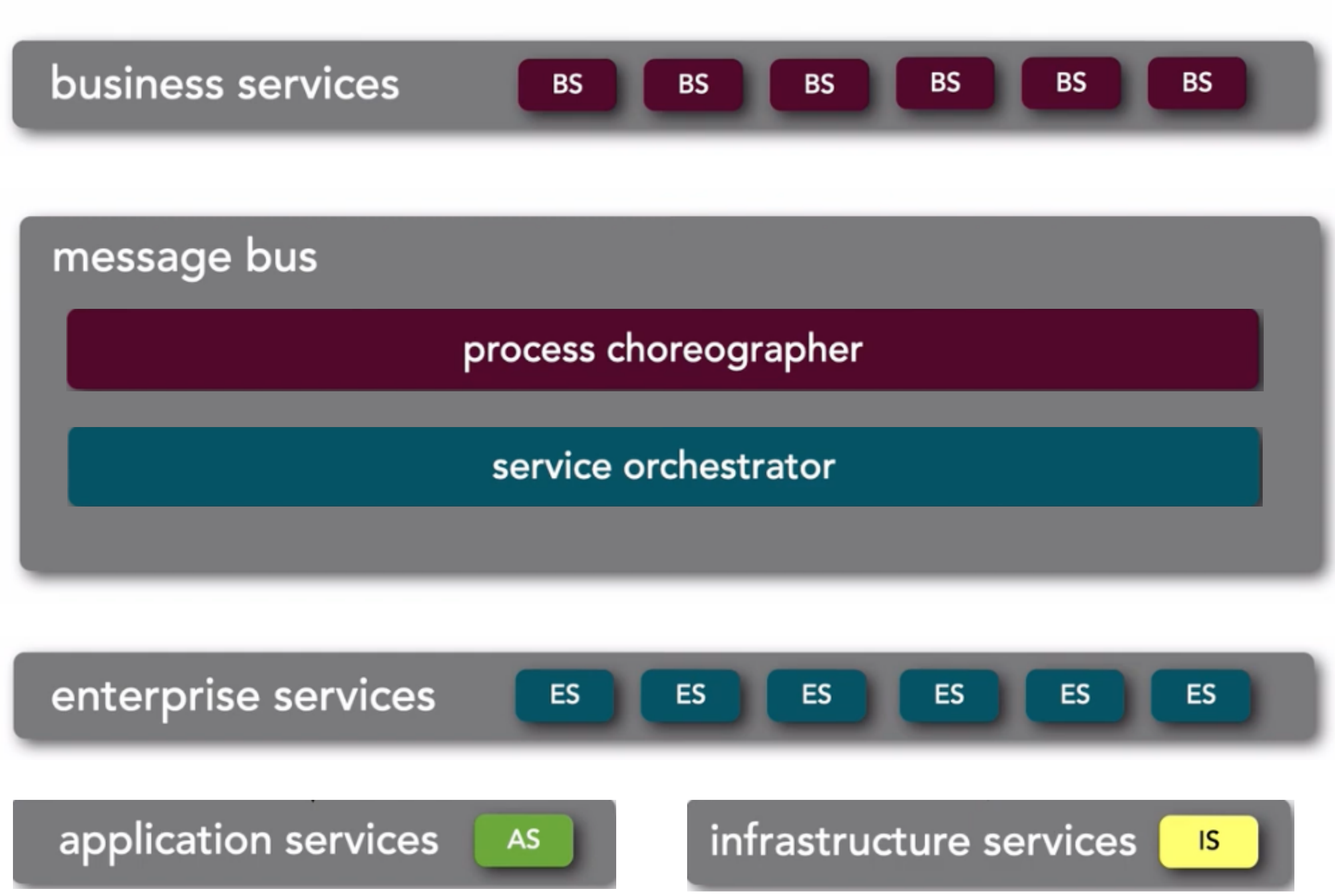
¿Que es SOA ?


business services
BS
BS
BS
BS
BS
BS
message bus
process choreographer
service orchestrator
enterprise services
ES
ES
ES
ES
ES
ES

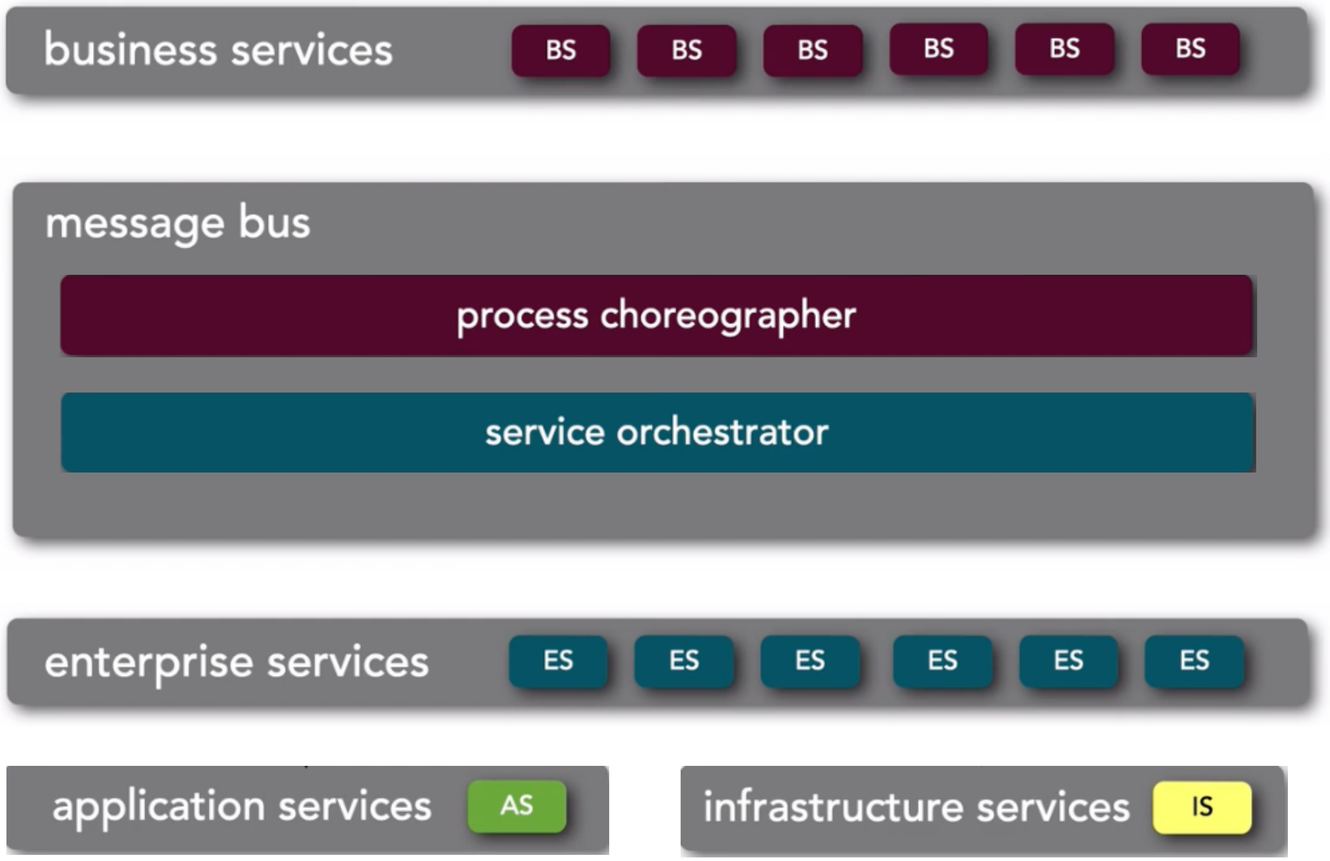
¿Que es SOA ?


business services
BS
BS
BS
BS
BS
BS
message bus
process choreographer
service orchestrator
enterprise services
ES
ES
ES
ES
ES
ES
application services
AS

¿Que es SOA ?


business services
BS
BS
BS
BS
BS
BS
message bus
process choreographer
service orchestrator
enterprise services
ES
ES
ES
ES
ES
ES
application services
AS
infrastructure services
IS

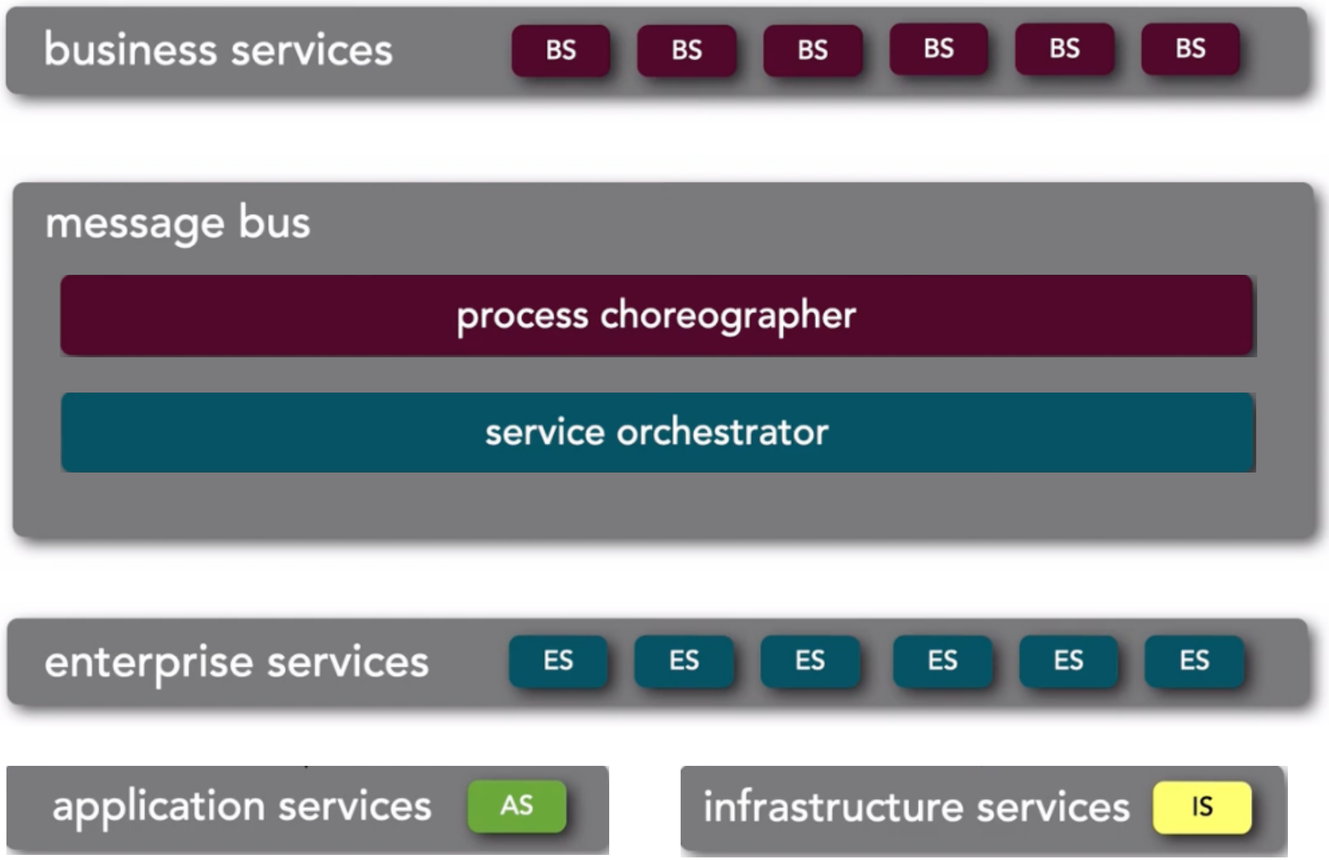
¿Que es SOA ?



ORACLE
SERVICE BUS

WSO2

¿Que es SOA ?



¿Cuando usarla?

Una realidad incomoda...

Coordinar APIs en diferentes equipos es una pesadilla

Anatomia de una API

API Schema

open-api-spec.yml

Swagger Editor File ▾ Edit ▾

```yaml
      type: array
      items:
        type: string
    Video:
      title: video
      type: array
      items:
        type: string
        format: binary
    Reviews:
      title: reviews
      type: array
      items:
        type: string
    Tags:
      title: tags
      type: array
      items:
        type: string
    Hypermedia:
      title: hypermedia
      type: object
      allOf:
        - $ref: '#/components/schemas/MediaMetadata'
        - $ref: '#/components/schemas/Images'
        - $ref: '#/components/schemas/Video'
        - $ref: '#/components/schemas/Tags'
        - $ref: '#/components/schemas/Reviews'
# Comment the next definition out and there will be no syntax errors displayed
    #Image:
      #title: image
      #type: object
      #properties:
        #size: int
```

schema.gql

```
type User {
  name: String
  email: GraphQLEmail
  websiteURL: GraphQLURL
  posts: [Post]
  comments: [Comment]
  groups: [Group]
}
type Post {
  id: ID!
  title: String
  author: User
}
```

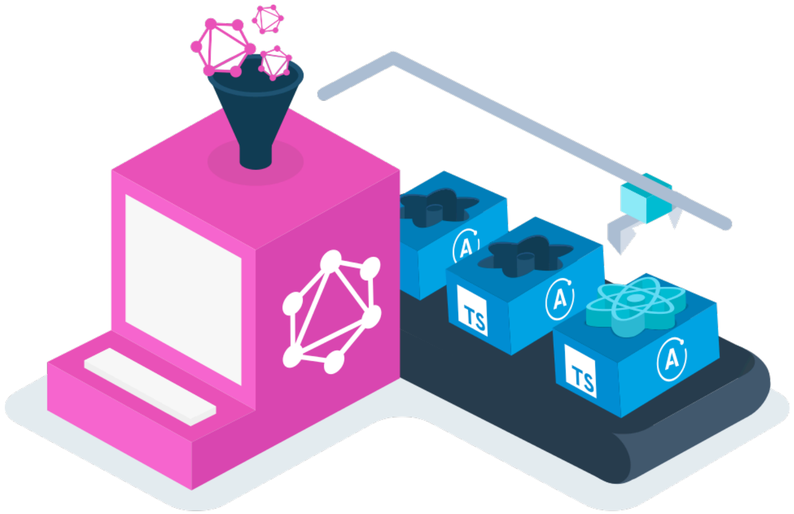
Anatomia de una API

API Schema

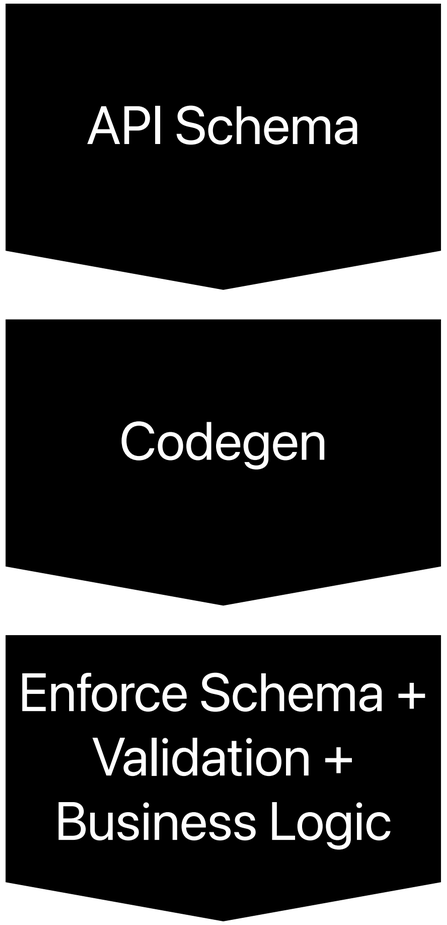
Codegen

Pydantic

Bringing schema and sanity to your data



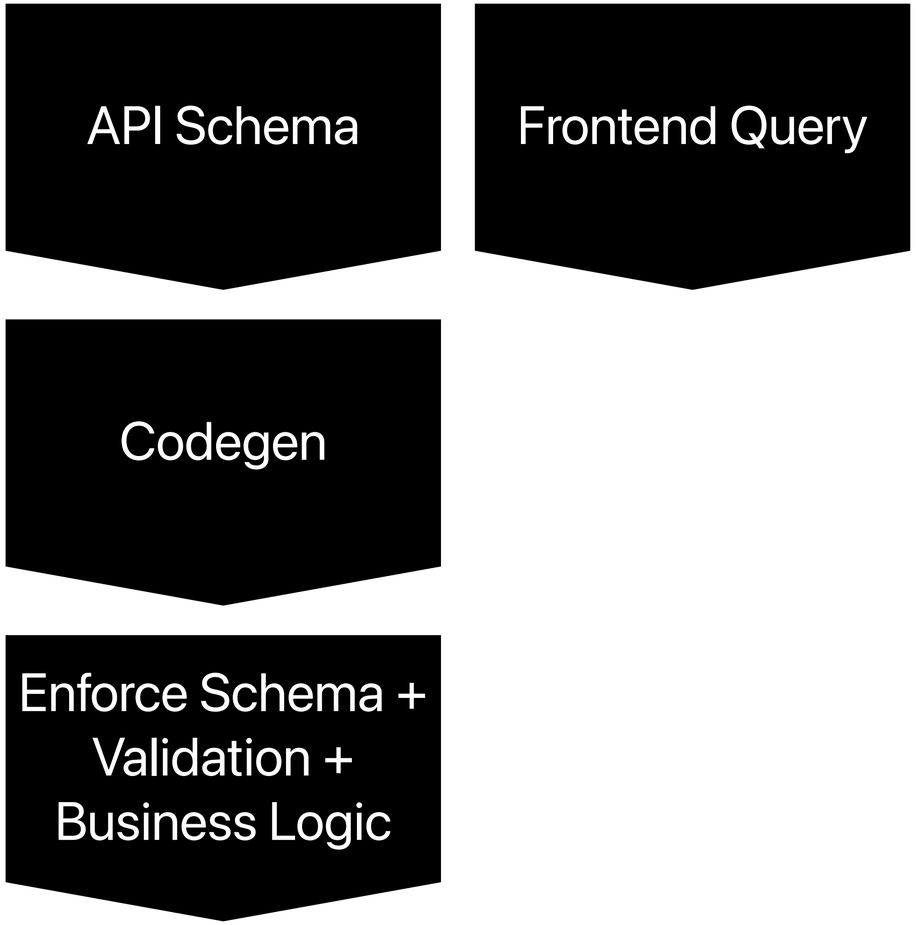
Anatomia de una API



Pydantic

Bringing schema and sanity to your data

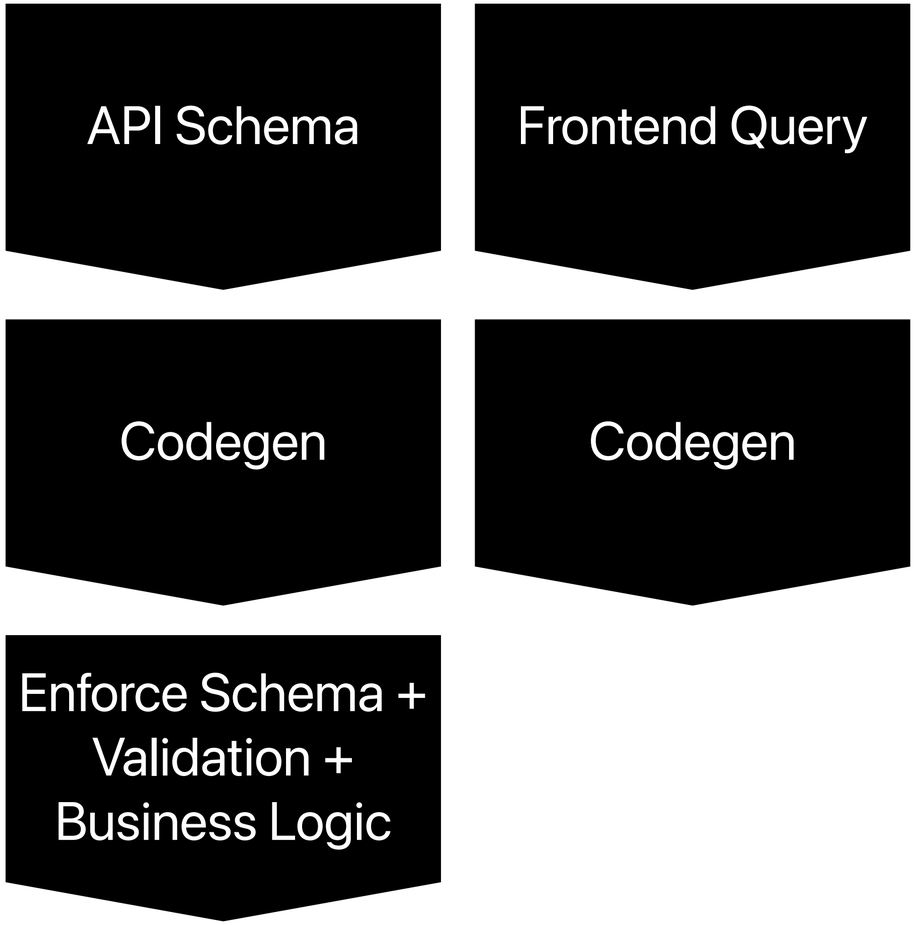
Anatomia de una API



Angular

APOLLO

Anatomia de una API

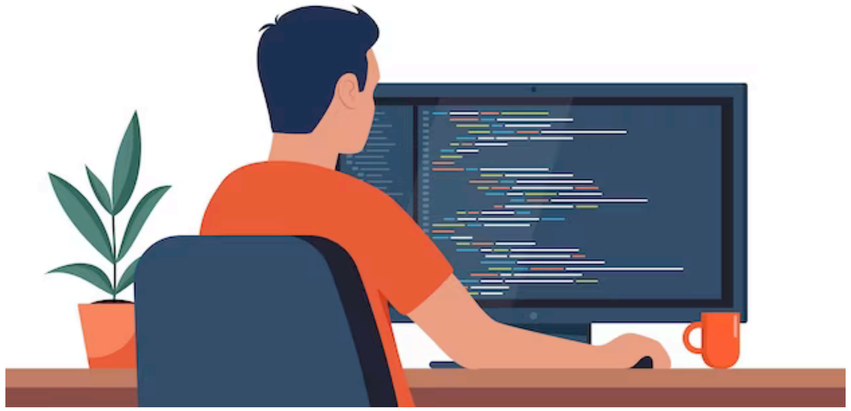


TS-REST

APOLLO

Anatomia de una API

API Schema	Frontend Query
Codegen	Codegen
Enforce Schema + Validation + Business Logic	fetchMyData()



Porque no podemos compartir funciónes ?

Creamos una función
en nuestro backend

Como hacer una API en tRPC

Como hacer una API en tRPC

Creamos una función en nuestro backend

```ts
export const appRouter = router({
  greeting: procedure.query(() ⇒ "Hello" as const),
});
```

Como hacer una API en tRPC

Creamos una función en nuestro backend

```ts
export const appRouter = router({
  greeting: procedure.query(() ⇒ "Hello" as const),
});
```

Usamos la función en nuestro frontend

Como hacer una API en tRPC

Creamos una función en nuestro backend

```ts
export const appRouter = router({
  greeting: procedure.query(() ⇒ "Hello" as const),
});
```

Usamos la función en nuestro frontend

```ts
const res = await trpc.greeting.query();
//    ^? = const res: "Hello"
```

tRPC en acción

```ts
const t = initTRPC.create();

const router = t.router;
const publicProcedure = t.procedure;

const appRouter = router({
  userList: publicProcedure.input(z.string()).query(async (opts) => {
    const { input } = opts;
    const user = await db.user.findById(input);

    return user;
  }),
  userCreate: publicProcedure
    .input(z.object({ name: z.string() }))
    .mutation(async (opts) => {
      const { input } = opts;
      const user = await db.user.create(input);

      return user;
    }),
});

export type AppRouter = typeof appRouter;

const server = createHTTPServer({
  router: appRouter,
});

server.listen(3000);
```

```ts
import { createTRPCClient, httpBatchLink } from "@trpc/client";
import type { AppRouter } from "./server";
//     👆 **type-only** import
// Pass AppRouter as generic here. 👇 This lets the `trpc` object know
// what procedures are available on the server and their input/output types.
const trpc = createTRPCClient<AppRouter>({
  links: [
    httpBatchLink({
      url: "http://localhost:3000",
    }),
  ],
});


// Inferred types
const user = await trpc.userById.query('1');
//    ^? const user: {
//         name: string;
//         id: string;
//       } | undefined


const createdUser = await trpc.userCreate.mutate({ name: 'sachinraja' });
//    ^? const createdUser: {
//         name: string;
//         id: string;
//       }
```

Metemos a las bases de datos en la ecuación

Prisma

```ts
const postRouter = router({
    hello: publicProcedure
      .input(z.object({ text: z.string() }))
      .query(({ input }) => {
        return {
          greeting: `Hello ${input.text}`,
        };
      }),

    create: publicProcedure
      .input(z.object({ name: z.string().min(1) }))
      .mutation(async ({ ctx, input }) => {
        return ctx.db.post.create({
          data: {
            name: input.name,
          },
        });
      }),

    getLatest: publicProcedure.query(async ({ ctx }) => {
      const post = await ctx.db.post.findFirst({
        orderBy: { createdAt: "desc" },
      });

      return post ?? null;
    }),
  });
```

```
generator client {
    provider = "prisma-client-js"
}

datasource db {
    provider = “mysql"
    url      = env("DATABASE_URL")
}

model Post {
    id        Int      @id @default(autoincrement())
    name      String
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt

    @@index([name])
}
```

Drizzle

```ts
export const postRouter = router({
  hello: publicProcedure
    .input(z.object({ text: z.string() }))
    .query(({ input }) => {
      return {
        greeting: `Hello ${input.text}`,
      };
    }),

  create: publicProcedure
    .input(z.object({ name: z.string().min(1) }))
    .mutation(async ({ ctx, input }) => {
      await ctx.db.insert(posts).values({
        name: input.name,
      });
    }),

  getLatest: publicProcedure.query(async ({ ctx }) => {
    const post = await ctx.db.query.posts.findFirst({
      orderBy: (posts, { desc }) => [desc(posts.createdAt)],
    });

    return post ?? null;
  }),
});
```

```ts
export const posts = createTable(
  "post",
  {
    id: bigint("id", { mode: "number" })
        .primaryKey()
        .autoincrement(),
    name: varchar("name", { length: 256 }),
    createdAt: timestamp("created_at")
      .default(sql`CURRENT_TIMESTAMP`)
      .notNull(),
    updatedAt: timestamp("updated_at").onUpdateNow(),
  },
  (example) => ({
    nameIndex: index("name_idx").on(example.name),
  })
);
```

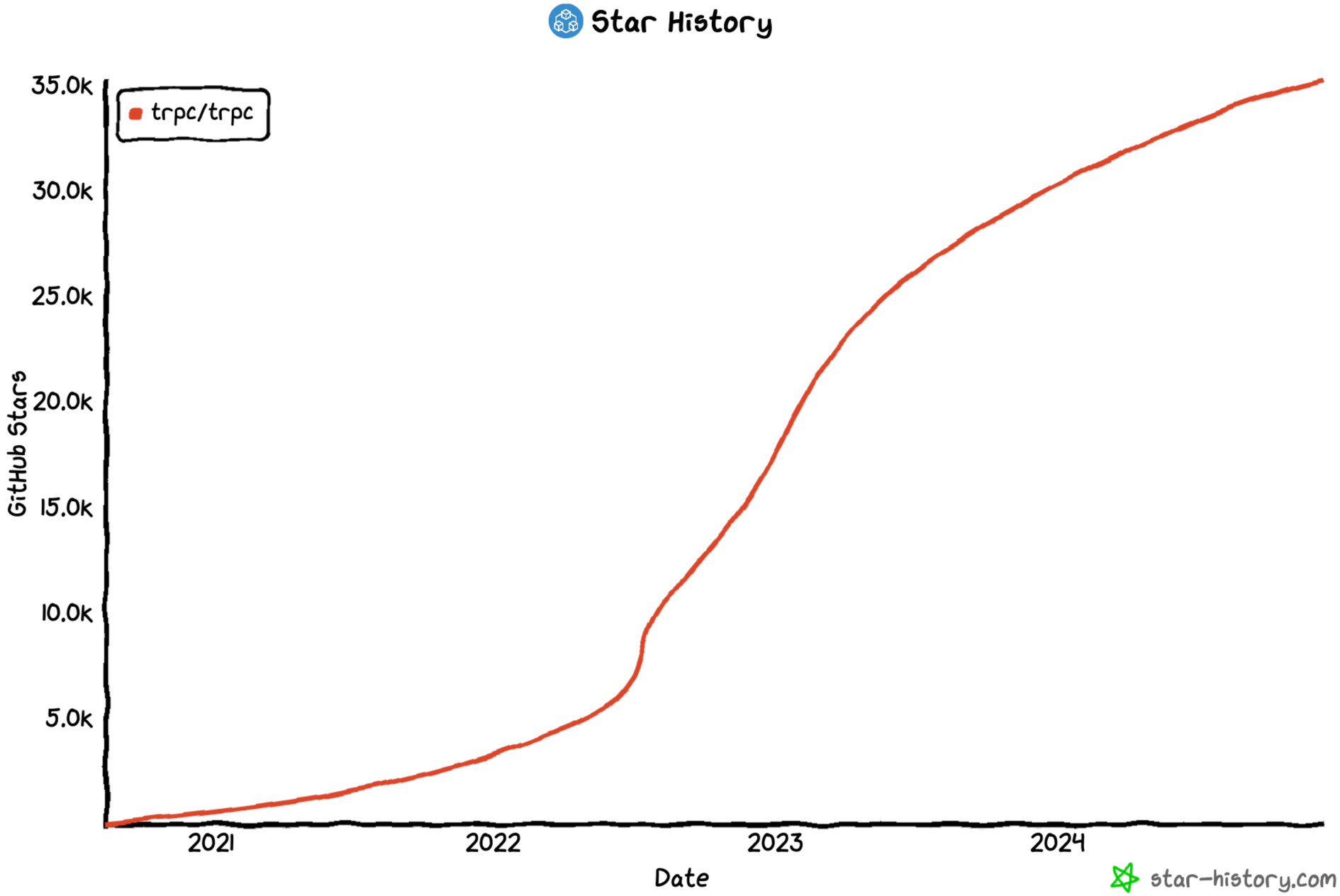
TypeScript

Un lenguaje para backend, frontend & mobile

tRPC is HUGE

+35,000 stars

+725,000 weekly npm downloads



“Aprended a dejar de tener miedo
a los garabatos rojos”

Entonces recomiendas usar tRPC para todo ?

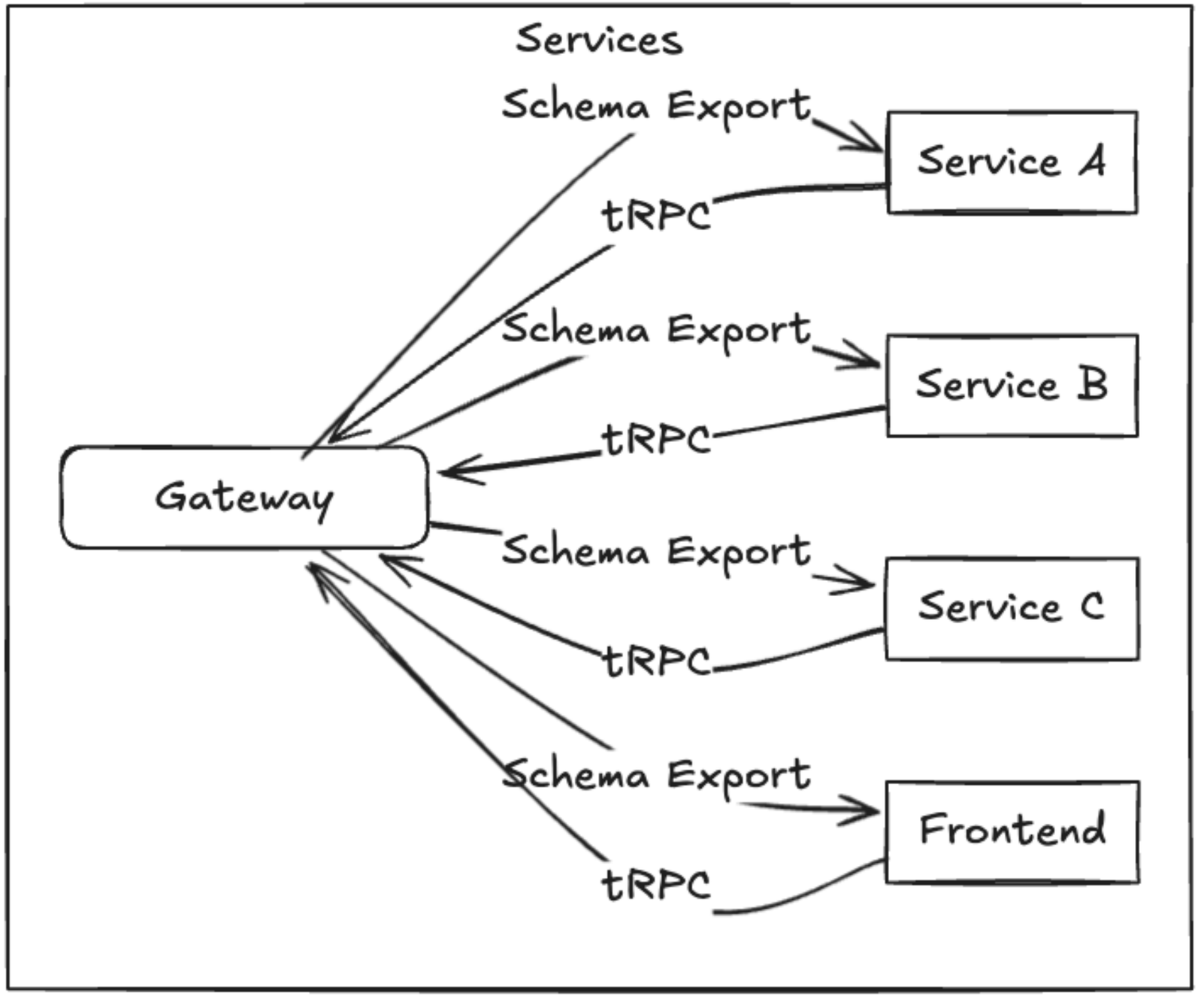
no

When to use each type of API ???

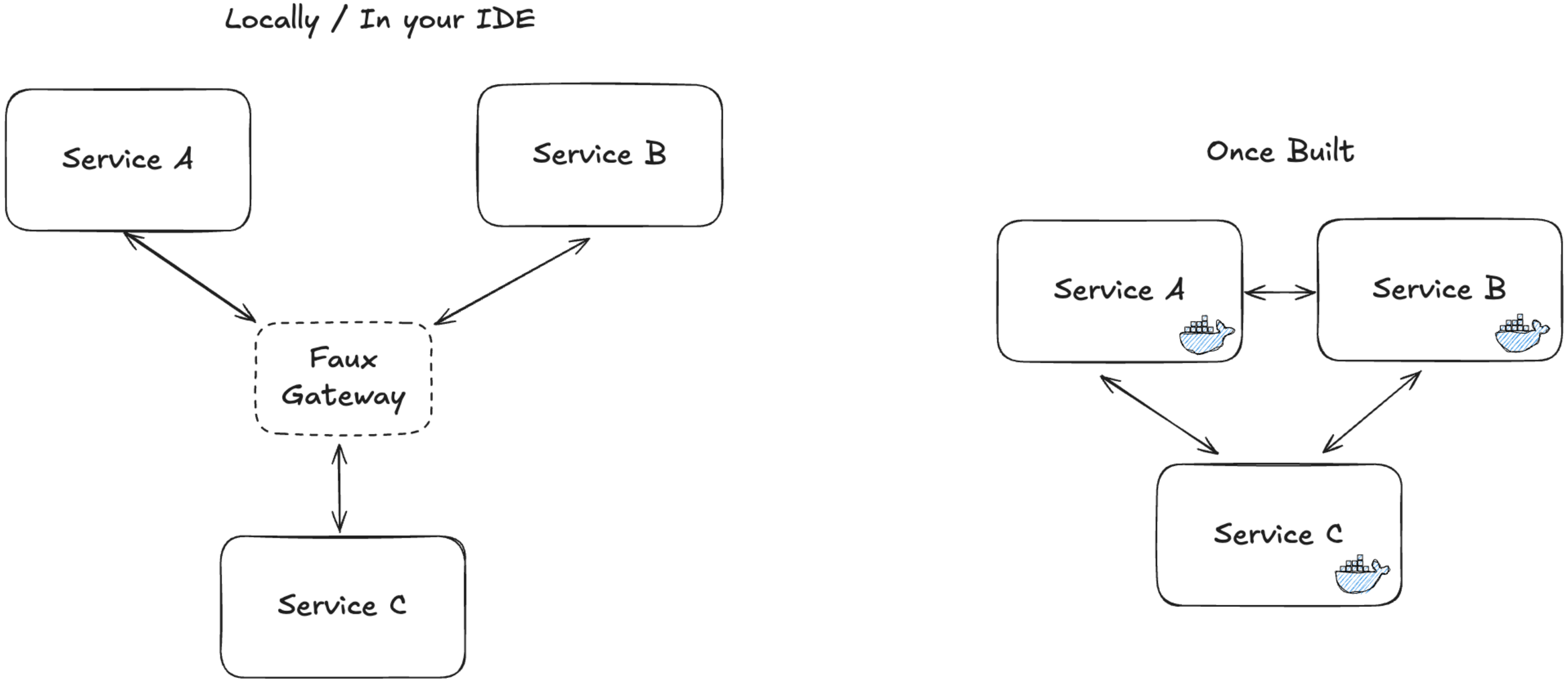
client/Server	Backend/Frontend: ...SAME Team	...DIFFERENT Teams	...DIFFERENT Companies / Public
...ARE BOTH Typescript	tRPC	GraphQL	REST
...AREN'T Typescript	GraphQL	GraphQL	REST

> If you already have a custom GraphQL-server for your project, you may not want to use tRPC. GraphQL is amazing; it's great to be able to make a flexible API where each consumer can pick just the data they need.
>
> The thing is, GraphQL isn't that easy to get right - ACL is needed to be solved on a per-type basis, complexity analysis, and performance are all non-trivial things.
>
> We've taken a lot of inspiration from GraphQL. If you've previously built GraphQL servers, you'll be familiar with the concepts of input types and resolvers.
>
> tRPC is a lot simpler and couples your server & website/app more tightly together (for good and for bad). It allows you to move quickly, make changes without having to update a schema, and avoid thinking about the ever-traversable graph.

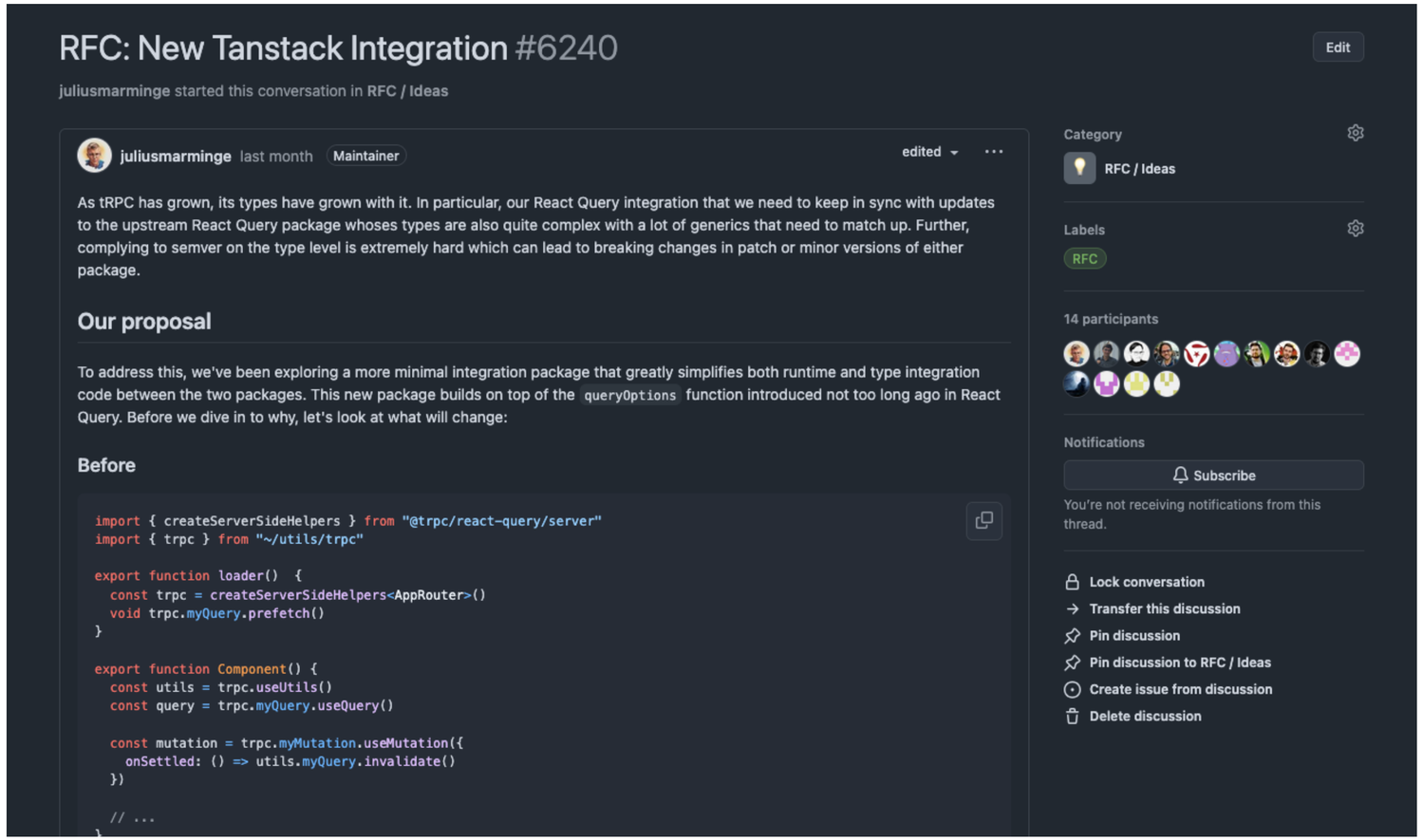
Y SOA ?


Services
Schema Export
Service A
tRPC
Schema Export
Service B
tRPC
Gateway
Schema Export
Service C
tRPC
Schema Export
Frontend
tRPC

Demo


Locally / In your IDE
Service A
Service B
Faux Gateway
Service C
Once Built
Service A
Service B
Service C

Ayudadnos con un RFC



RFC: New Tanstack Integration #6240

Edit

juliusmarminge started this conversation in RFC / Ideas

juliusmarminge last month Maintainer

edited

As tRPC has grown, its types have grown with it. In particular, our React Query integration that we need to keep in sync with updates to the upstream React Query package whoses types are also quite complex with a lot of generics that need to match up. Further, complying to semver on the type level is extremely hard which can lead to breaking changes in patch or minor versions of either package.

Our proposal

To address this, we've been exploring a more minimal integration package that greatly simplifies both runtime and type integration code between the two packages. This new package builds on top of the `queryOptions` function introduced not too long ago in React Query. Before we dive in to why, let's look at what will change:

Before

```
import { createServerSideHelpers } from "@trpc/react-query/server"
import { trpc } from "~/utils/trpc"

export function loader()  {
  const trpc = createServerSideHelpers<AppRouter>()
  void trpc.myQuery.prefetch()
}

export function Component() {
  const utils = trpc.useUtils()
  const query = trpc.myQuery.useQuery()

  const mutation = trpc.myMutation.useMutation({
    onSettled: () => utils.myQuery.invalidate()
  })

  // ...
}
```

Category

RFC / Ideas

Labels

RFC

14 participants

Notifications

Subscribe

You're not receiving notifications from this thread.

Lock conversation

Transfer this discussion

Pin discussion

Pin discussion to RFC / Ideas

Create issue from discussion

Delete discussion

https://github.com/trpc/trpc/discussions/6240

Gracias

¿Preguntas?

GitHub: nsttt

Twitter: @nstlopez

BlueSky: https://nst.blue/

Néstor López

Software Engineer

CNCF Galicia

14 Nov 2024 @ A Coruña