Dragonfly

TypeScript Magic: End-to-End Type Safety Across the Full Stack

Developer Advocate @ DragonflyDB

Dragonfly

```
dragonfly> HGETALL joe
1) "name"
2) "Joe Zhou"
3) "role"
4) "Developer Advocate | Dragonfly"
5) "guilty_pleasure"
6) "Attending way too many K-pop concerts."

dragonfly> ZRANGE work_history_joe 2024 2015 BYSCORE REV
1) "Developer Advocate        | Dragonfly"
2) "Senior Software Engineer | Affinity"
3) "Senior Software Engineer | Hopper"
4) "Application Developer     | Bell Canada"
5) "Application Developer     | IBM"
...
```

I MAY LOOK WEIRD AND CRAZY
BUT IM ACTUALLY VERY PASSIONATE ABOUT WHAT I DO

Dragonfly

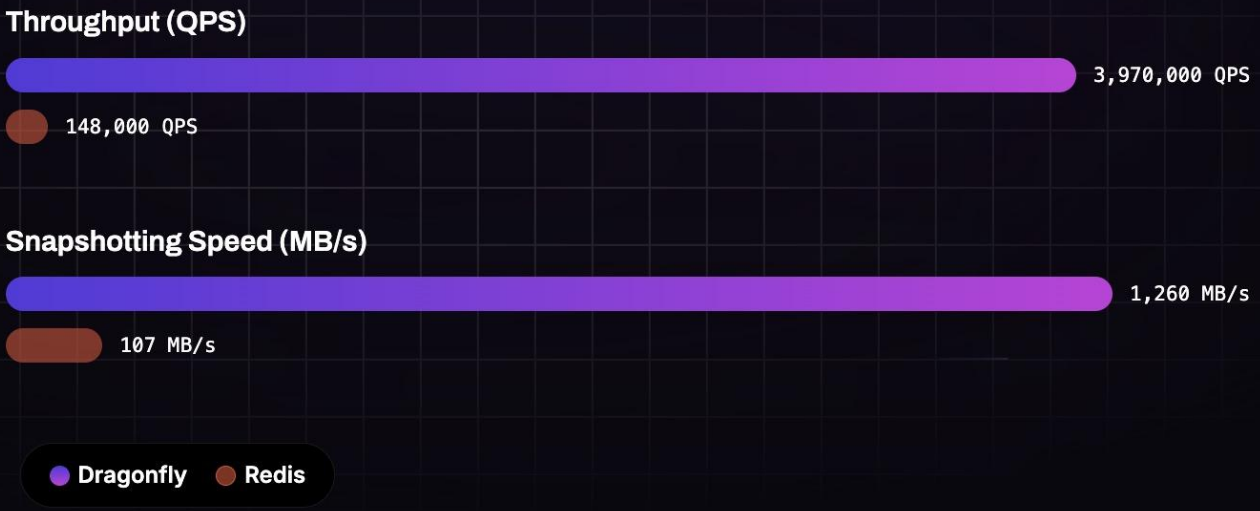
Dragonfly

25x

More QPS than Redis

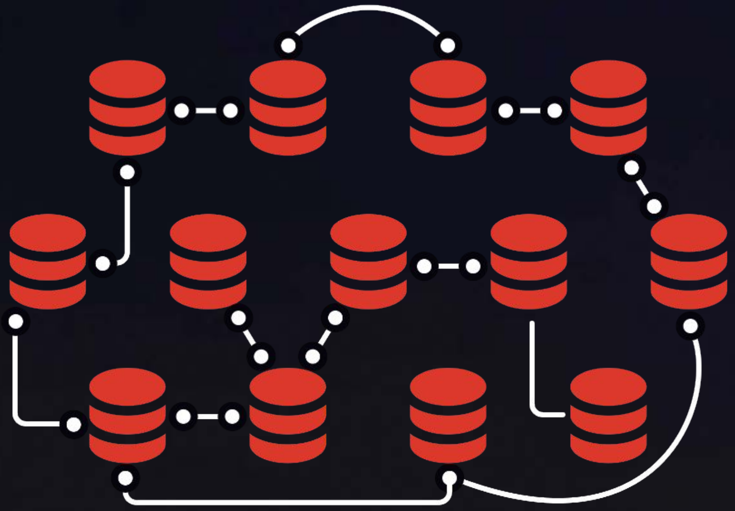
12x

Faster snapshotting than Redis



QPS benchmark on AWS c6gn.16xlarge. Snapshot benchmark on AWS c6gn.4xlarge. Source.

VS


Redis

Dragonfly

```javascript
import { Redis } from 'ioredis';

const client = new Redis({
    port: 6379,
    host: "my.redis.instance.com",
    username: "default",
    password: "top-secret",
    db: 0,
});
```

Dragonfly

```
import { Redis } from 'ioredis';

const client = new Redis({
    port: 6379,
    host: "my-instance.dragonflydb.cloud", 👉
    username: "default",
    password: "top-secret",
    db: 0,
});

client.set("hello", "dragonfly"); 👉
```

Programming Languages

- Right tools for the right things.
- The power of strong & static typing.

Dragonfly

TypeScript Utility Types

Dragonfly

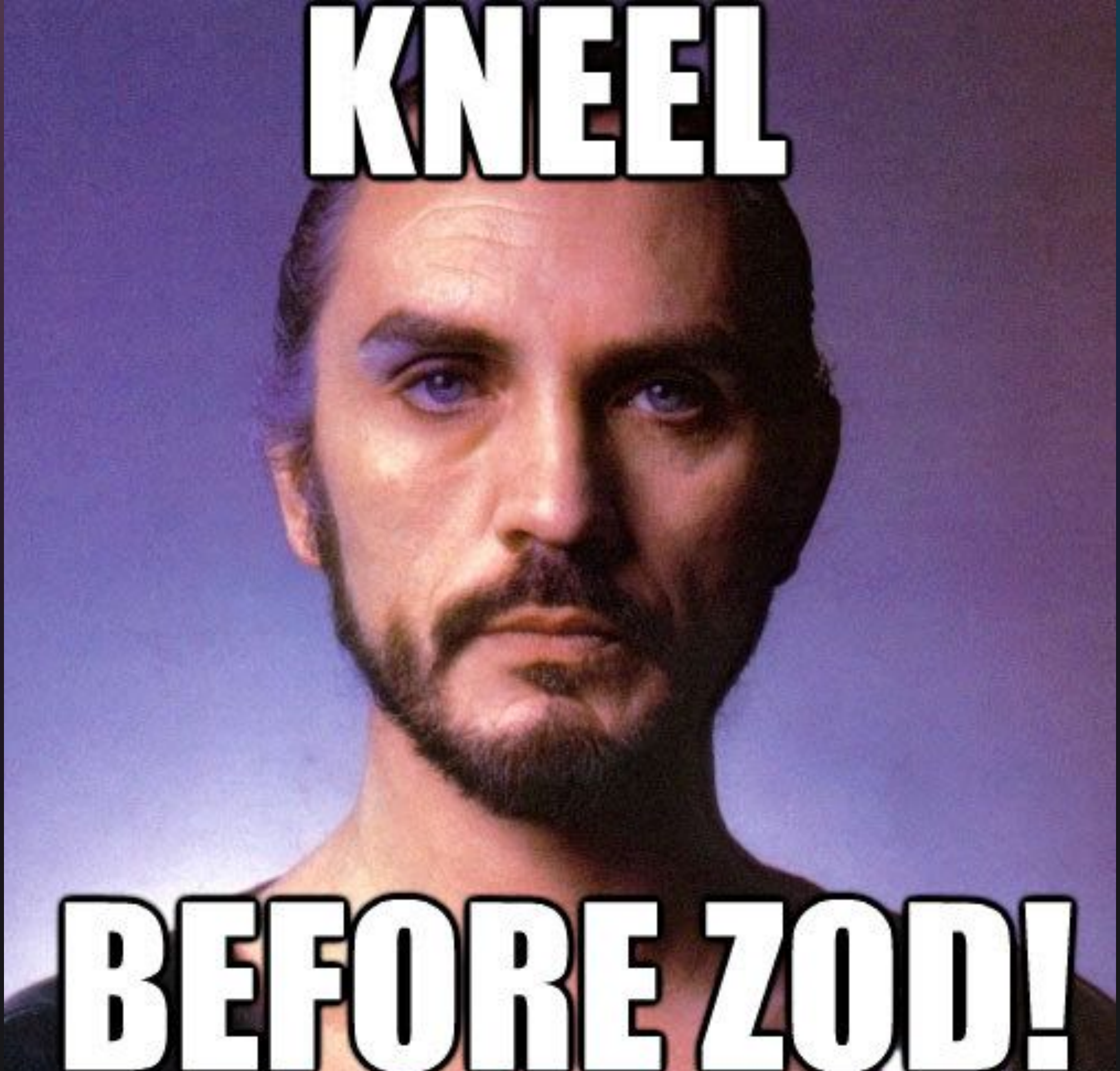
```
type BookRecord = {
    id: string;
    title: string;
    authorId: string;
    publicationDate: Date;
    ISBN: string;
    stock: bigint;
}
```

```
type BookRecord = {
   id: string;
   title: string;
   authorId: string;
   publicationDate: Date;
   ISBN: string;
   stock: bigint;
}

type BookRequest = Omit<BookRecord, 'id'>; 👈
```

Dragonfly

```
type BookPreview = Pick<BookRecord, "title">;

type BookPreview = {
  title: string;
}
```

```
type BookRO = Readonly<BookRecord>;

type BookRO = {
  readonly id: string;
  readonly title: string;
  readonly authorId: string;
  readonly publicationDate: Date;
  readonly ISBN: string;
  readonly stock: bigint;
}
```



Dragonfly

Dragonfly

```
import { z } from 'zod';

const bookRecord = z.object({
   id: z.string().uuid(),
   title: z.string(),
   authorId: z.string().uuid(),
   publicationDate: z.date(),
   ISBN: z.string(),
   stock: z.bigint(),
});
```

Dragonfly

```
import { z } from 'zod'; 👈

const bookRecord = z.object({
  id: z.string().uuid(),
  title: z.string(),
  authorId: z.string().uuid(),
  publicationDate: z.date(),
  ISBN: z.string(),
  stock: z.bigint(),
});
```

```
import { z } from 'zod';

const bookRecord = z.object({
   id: z.string().uuid(),
   title: z.string(),
   authorId: z.string().uuid(),
   publicationDate: z.date(),
   ISBN: z.string(), 👉
   stock: z.bigint(),
});
```

```
import { z } from 'zod';

const bookRecord = z.object({
  id: z.string().uuid(),
  title: z.string(),
  authorId: z.string().uuid(),
  publicationDate: z.date(),
  ISBN: z.string().regex(
    /^(?=(?:\D*\d){10}(?:(?:\D*\d){3})?$)[\d-]+$/,
    {
      message: 'ISBN must be either 10 or 13 digits long'
    },
  ),
  stock: z.bigint(),
});
```

```
import { z } from 'zod';

const bookRecord = z.object({ ... });

bookRecord.parse({
   authorId: "123",
});
```

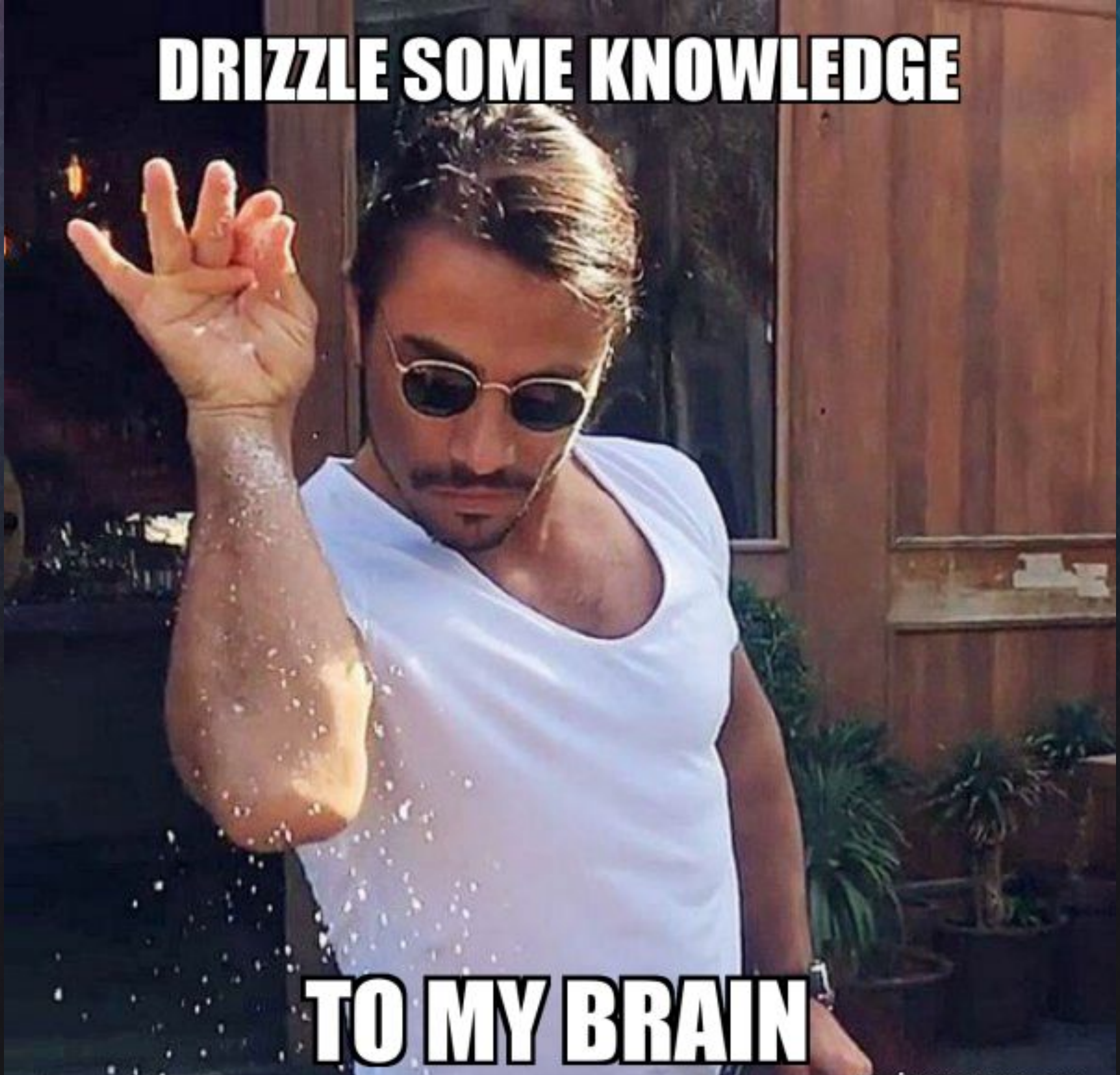
Dragonfly

```
import { z } from 'zod';

const bookRecord = z.object({ ... });

type BookRecord = z.infer<typeof bookRecord>;
type BookRecord = {
   id: string;
   title: string;
   authorId: string;
   publicationDate: Date;
   ISBN: string;
   stock: bigint;
}
```

```
import { z } from 'zod';

const bookRecord = z.object({ ... });

const bookRequest = bookRecord.omit({ id: true });

type BookRequest = z.infer<typeof bookRequest>;
type BookRequest = {
   title: string;
   authorId: string;
   publicationDate: Date;
   ISBN: string;
   stock: bigint;
}
```

Request Validator Object

Request Type Def



Dragonfly

Dragonfly

Drizzle

Dragonfly

```
import {
   pgTable, text, timestamp,
   uuid, bigint
} from 'drizzle-orm/pg-core';

const books = pgTable('books', {
   id: uuid().primaryKey(),
   title: text().notNull(),
   authorId: uuid('author_id').notNull(),
   publicationDate: timestamp('publication_date', { mode: 'date' }).notNull(),
   ISBN: text('isbn').notNull(),
   stock: bigint({ mode: 'number' }).notNull(),
});
```

Dragonfly

```
import { createInsertSchema } from 'drizzle-zod';
import { uuidv7 } from 'uuidv7';

const books = pgTable('books', { ... });

const bookInsertSchema = createInsertSchema(books, {
  id: (schema) => schema.id.default(uuidv7),
  title: (schema) => schema.title.min(1),
  authorId: (schema) => schema.authorId.uuid(),
  publicationDate: (schema) => schema.publicationDate.min(new Date('2000-01-01')),
  ISBN: (schema) => schema.ISBN.regex(/^(?=(?:\D*\d){10}(?:(?:\D*\d){3})?$)[\d-]+$/),
  stock: (schema) => schema.stock.gte(0),
});
```

```
import { createInsertSchema } from 'drizzle-zod';
import { uuidv7 } from 'uuidv7';


const books = pgTable('books', { ... });


const bookInsertSchema = createInsertSchema(books, {
  id: (schema) => schema.id.default(uuidv7), 👈
  title: (schema) => schema.title.min(1),
  authorId: (schema) => schema.authorId.uuid(),
  publicationDate: (schema) => schema.publicationDate.min(new Date('2000-01-01')),
  ISBN: (schema) => schema.ISBN.regex(/^(?=(?:\D*\d){10}(?:(?:\D*\d){3})?$)[\d-]+$/),
  stock: (schema) => schema.stock.gte(0),
});
```

```
import { createInsertSchema } from 'drizzle-zod';
import { uuidv7 } from 'uuidv7';


const books = pgTable('books', { ... });


const bookInsertSchema = createInsertSchema(books, {
  id: (schema) => schema.id.default(uuidv7),
  title: (schema) => schema.title.min(1),
  authorId: (schema) => schema.authorId.uuid(),
  publicationDate: (schema) => schema.publicationDate.min(new Date('2000-01-01')), 👈
  ISBN: (schema) => schema.ISBN.regex(/^(?=(?:\D*\d){10}(?:(?:\D*\d){3})?$)[\d-]+$/),
  stock: (schema) => schema.stock.gte(0),
});
```

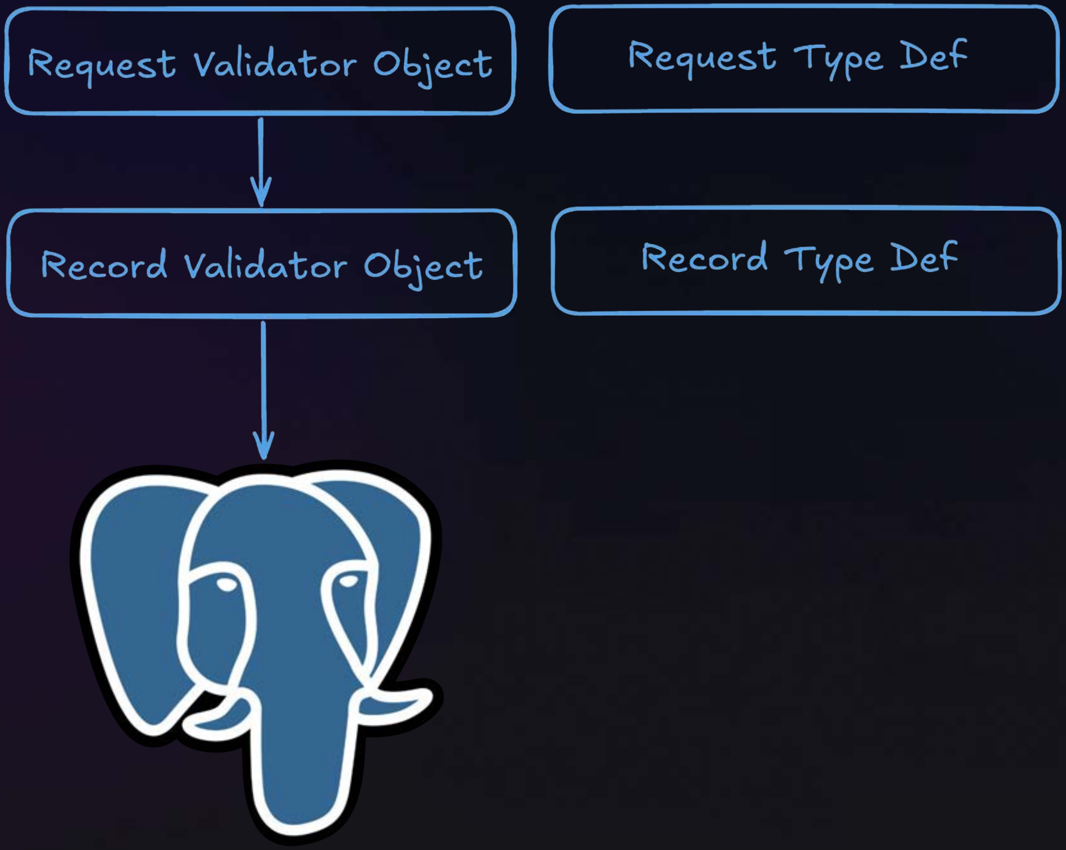
```
const books = pgTable('books', { ... });
const bookInsertSchema = createInsertSchema(books, { ... });

const bookRequestSchema = bookInsertSchema
   .omit({ id: true, publicationDate: true })
   .setKey('publicationDate',
       z.string().transform((val, ctx) => {
           // Parse the date in a way you like.
           return date;
       }),
   )
```

Dragonfly

```
const books = pgTable('books', { ... });
const bookInsertSchema = createInsertSchema(books, { ... });
const bookRequestSchema = bookInsertSchema.omit({ ... }).setKey( ... );

type BookInsert = z.infer<typeof bookInsertSchema>;
type BookRequest = z.infer<typeof bookRequestSchema>;
```

```
type BookRequest = {
   title: string;
   authorId: string;
   publicationDate: Date;
   ISBN: string;
   stock: number;
}
```

Dragonfly


Request Validator Object
Request Type Def
Record Validator Object
Record Type Def

Dragonfly

Hono

Dragonfly

```
import { Hono } from 'hono';
import { zValidator } from '@hono/zod-validator';

const app = new Hono();
const route = app.post('/books', zValidator('json', bookRequestSchema),
    (c) => {
        // Request is already validated by 'bookRequestSchema'.
        const validatedRequest: BookRequest = c.req.valid('json');
        // Use 'bookInsertSchema' to validate the request again.
        const bookToInsert: BookInsert = bookInsertSchema.parse(validatedRequest);
        console.log('new book is saved in database')
        return c.json(bookToInsert, 201);
    }
);
```

Dragonfly

```
import { Hono } from 'hono';
import { zValidator } from '@hono/zod-validator';

const app = new Hono();
const route = app.post('/books', zValidator('json', bookRequestSchema),
   (c) => { ... }
);

export type ServerType = typeof route;
```

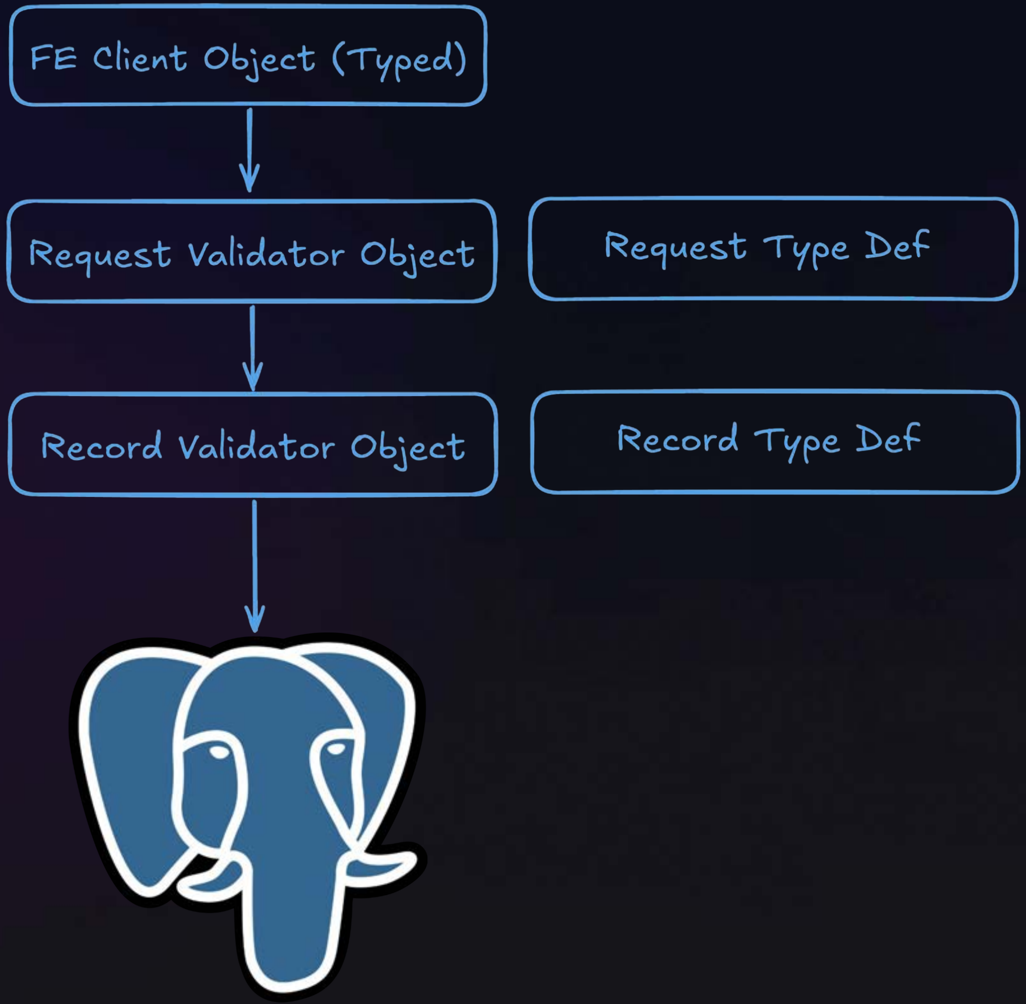
Dragonfly

```
import { type ServerType } from './server'; 👈
import { hc } from 'hono/client';

const client = hc<ServerType>('http://localhost:3000/');
client.books.$post({
    json: {
        title: 'Harry Potter',
        authorId: '0192bc31-747e-7b2a-b157-6a964de146a7',
        publicationDate: '2001-12-31',
        ISBN: '0-061-96436-0',
        stock: 5
    }
}).then(async (response) => { ... })
  .catch((error) => { ... });
```

Dragonfly

```
import { type ServerType } from './server';
import { hc } from 'hono/client';

const client = hc<ServerType>('http://localhost:3000/'); 👈
client.books.$post({
   json: {
       title: 'Harry Potter',
       authorId: '0192bc31-747e-7b2a-b157-6a964de146a7',
       publicationDate: '2001-12-31',
       ISBN: '0-061-96436-0',
       stock: 5
   }
}).then(async (response) => { ... })
  .catch((error) => { ... });
```

Dragonfly

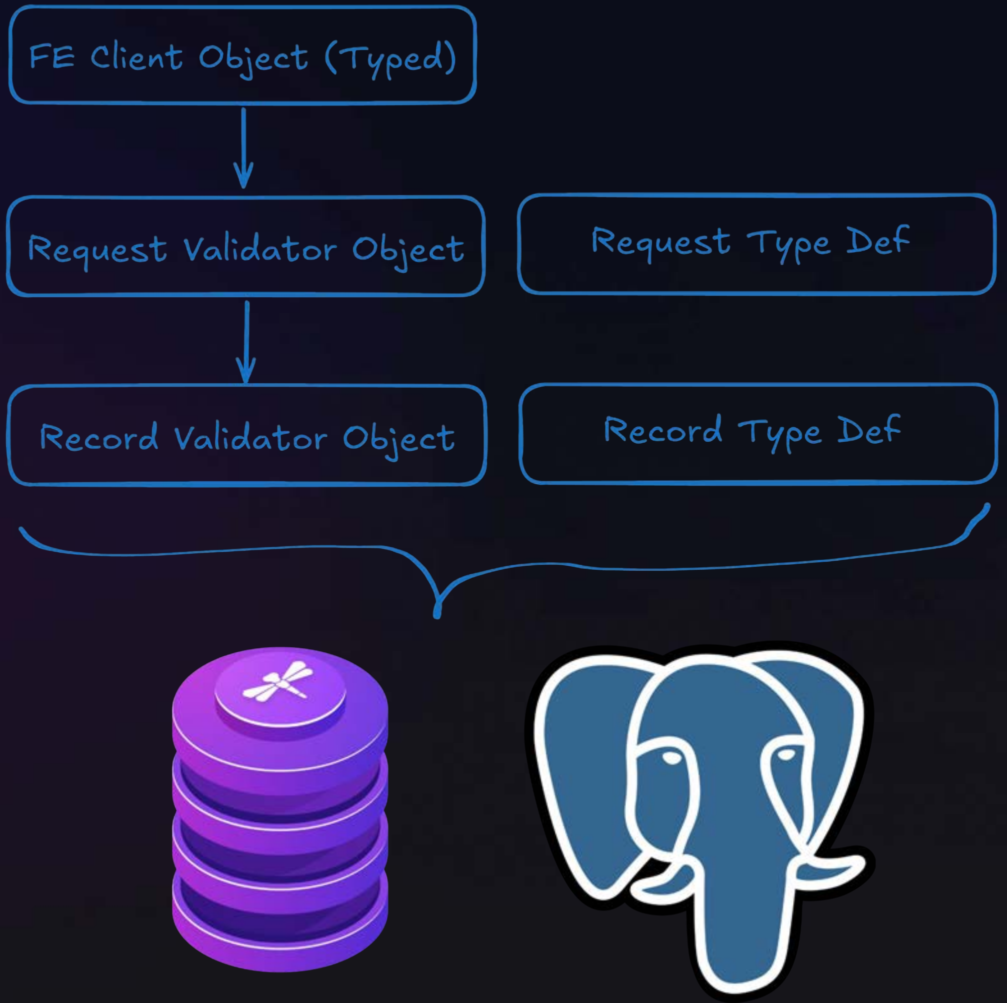

FE Client Object (Typed)
Request Validator Object
Request Type Def
Record Validator Object
Record Type Def

Dragonfly

What about Cache?

Dragonfly

```
import { Schema } from 'redis-om';

const books = new Schema('books', {
   id: { type: 'string' },
   title: { type: 'string' },
   authorId: { type: 'string' },
   publicationDate: { type: 'date' },
   ISBN: { type: 'string' },
   stock: { type: 'number' }
}, {
   dataStructure: 'JSON',
});
```


FE Client Object (Typed)
Request Validator Object
Request Type Def
Record Validator Object
Record Type Def

Compared to Other Solutions

- GraphQL, Swagger
- tRPC

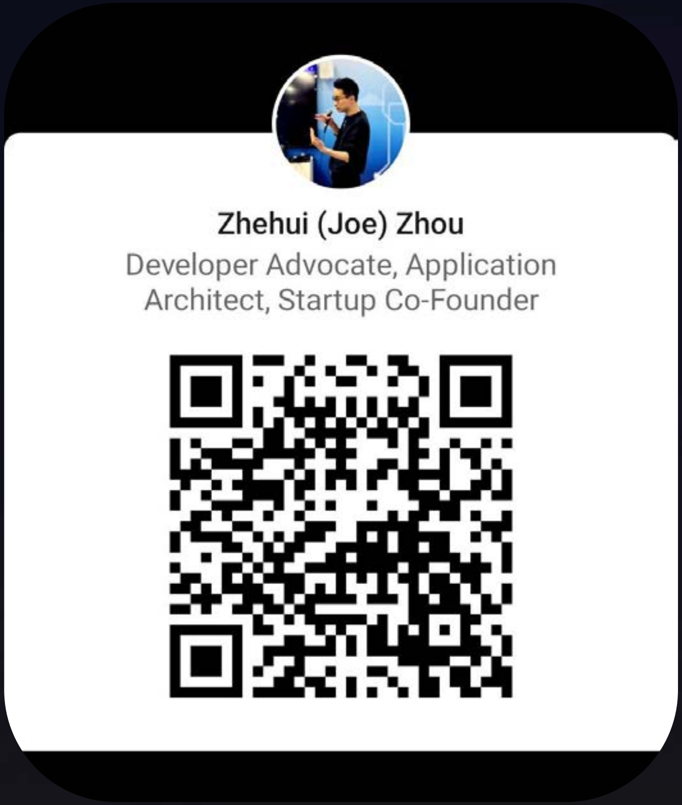
Dragonfly

Full-Stack Type Safety

Dragonfly

Thanks!

Please visit → **dragonflydb.io**



Zhehui (Joe) Zhou

Developer Advocate, Application Architect, Startup Co-Founder