Hive

GraphQLConf 2026

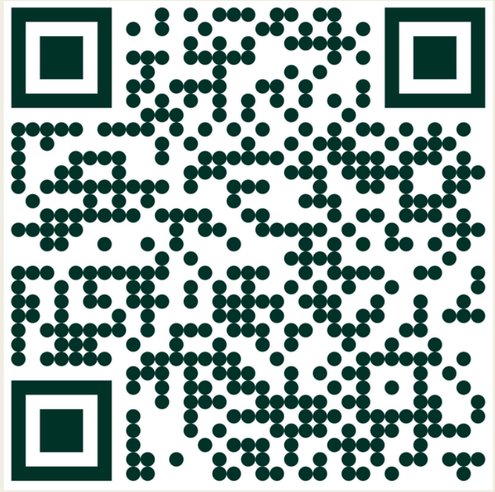
yelp

The Biggest Change to GraphQL Codegen in 10 years

Eddy Nguyen
The Guild, SEEK
eddeee888
eddeee888

Igor Kusakov
Yelp
ikusakov

1. Client plugins v6 redesign
2. Migration path from Apollo Tooling



Agenda

1. GraphQL Codegen history

Agenda

1. GraphQL Codegen history
2. Past designs & issues

Agenda

1. GraphQL Codegen history
2. Past designs & issues
3. Client plugins redesign

Agenda

1. GraphQL Codegen history
2. Past designs & issues
3. Client plugins redesign
4. Apollo Tooling migration

Agenda

1. GraphQL Codegen history
2. Past designs & issues
3. Client plugins redesign
4. Apollo Tooling migration
5. The collaboration

Agenda

1. GraphQL Codegen history
2. Past designs & issues
3. Client plugins redesign
4. Apollo Tooling migration
5. The collaboration
6. The future

Hive

GraphQL Codegen history

Mar 2019

@graphql-codegen/typescript
@graphql-codegen/typescript-operations
@graphql-codegen/typescript-resolvers
...

Nov 2016

First commit

Goal: GraphQL end-to-end type-safety

Nov 2016

First commit

Mar 2019

@graphql-codegen/typescript
@graphql-codegen/typescript-operations
@graphql-codegen/typescript-resolvers
...

Aug 2022

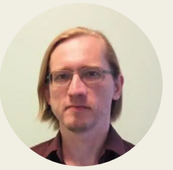
Client Preset

Oct 2022

Server Preset

Sept 2025

Server v5
Federation



Apr 2026

Client v6
Redesign &
Apollo Tooling migration

Goal: GraphQL end-to-end type-safety

Goal: Streamlined GraphQL DX

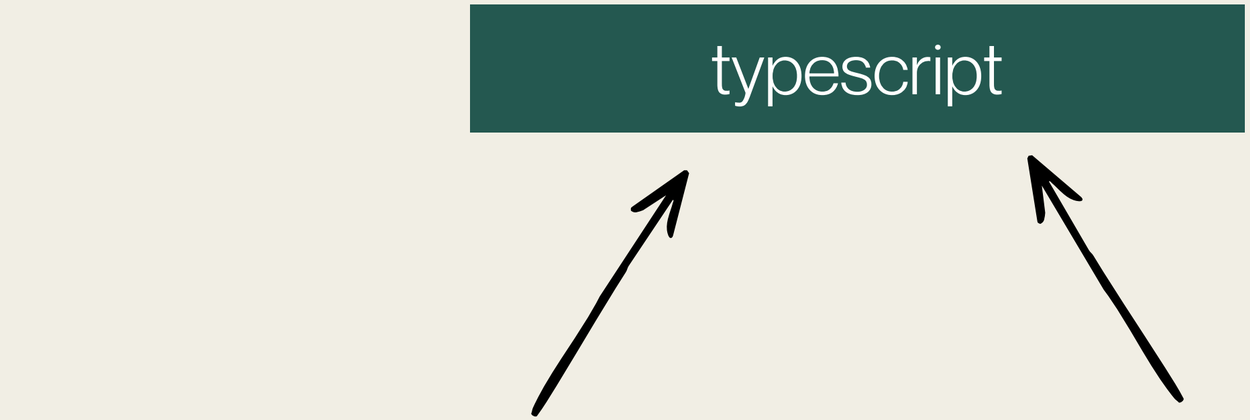
Hive

Past designs and issues

Base schema types

typescript

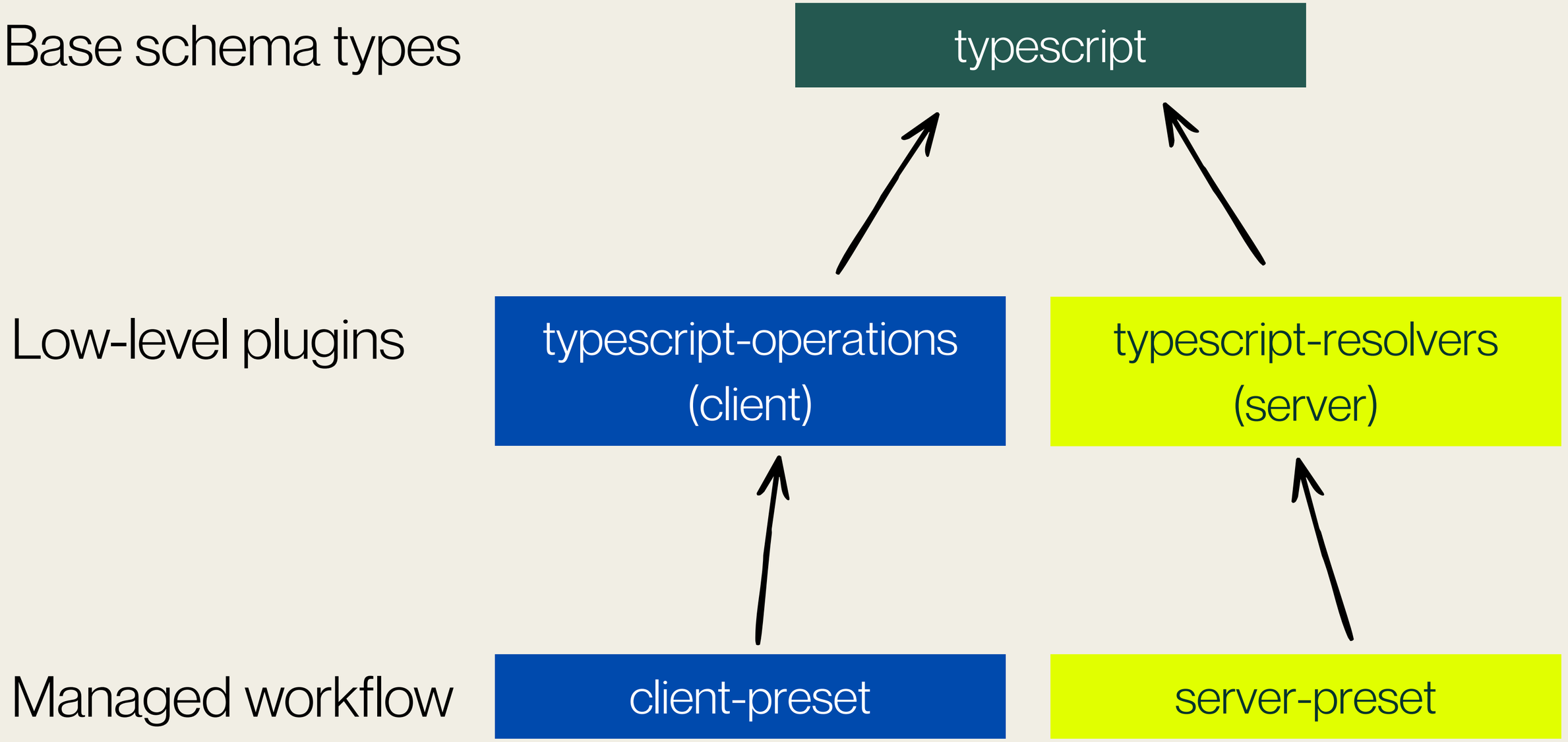
Base schema types



Low-level plugins

typescript-operations
(client)

typescript-resolvers
(server)


Base schema types
typescript
Low-level plugins
typescript-operations
(client)
typescript-resolvers
(server)
Managed workflow
client-preset
server-preset

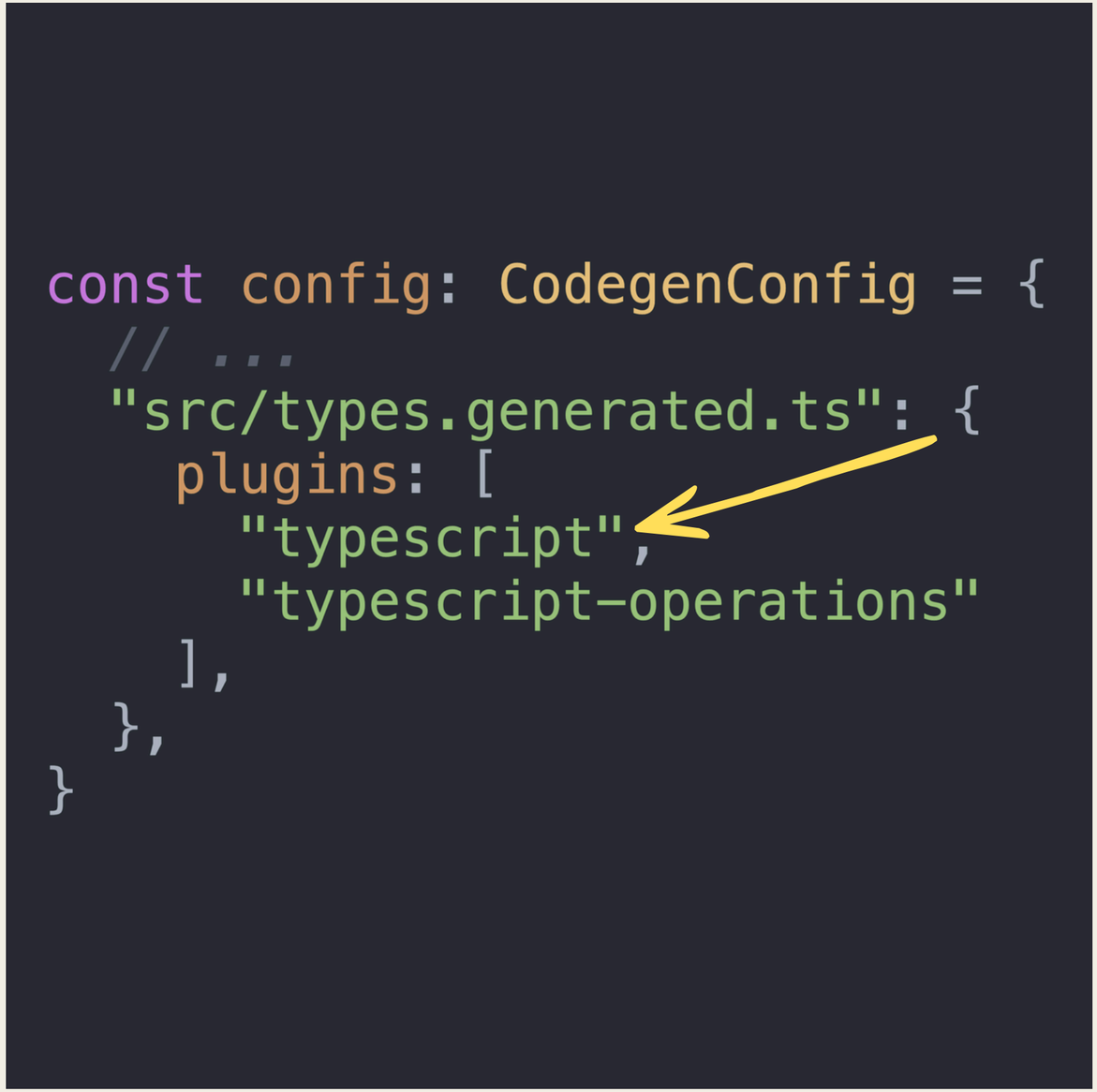
Designs / Issues

Designs / Issues

Plugin
interoperability

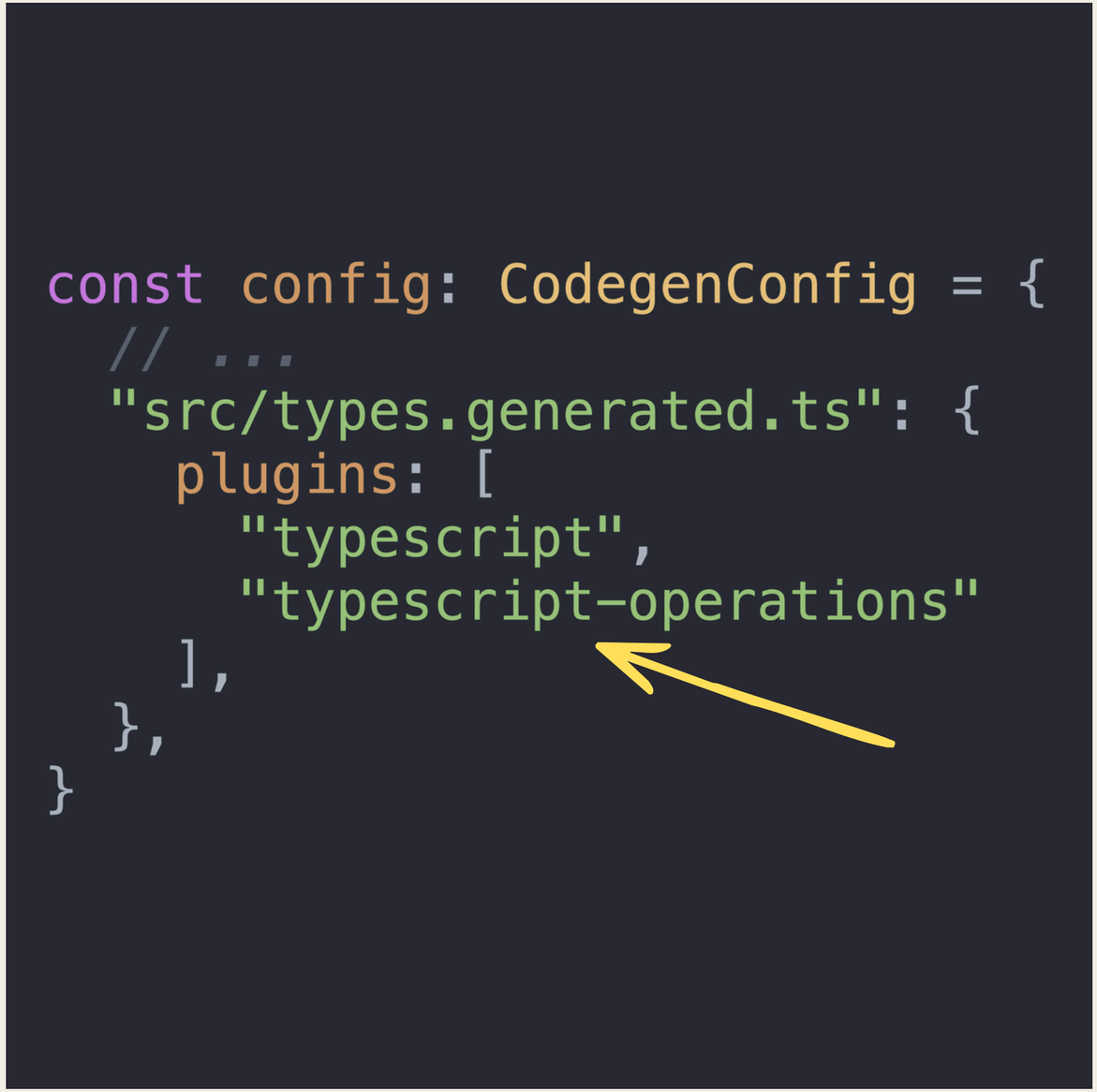
Designs / Issues

Plugin interoperability



Designs / Issues

Plugin interoperability



Designs / Issues

Unnecessary
types

Designs / Issues

Unnecessary types

```
type Query {
  user(input: UserInput!): User
}

input UserInput {
  id: ID!
}

type User {
  id: ID!
  name: String!
  role: UserRole!
}

enum UserRole {
  USER
  ADMIN
}
```

Designs / Issues

Unnecessary types

```
query User {
  user(input: { id: "1" }) {
    id
  }
}
```

Designs / Issues

Unnecessary types

```ts
export type Scalars = {
  ID: { input: string; output: string; }
  String: { input: string; output: string; }
  Boolean: { input: boolean; output: boolean; }
  Int: { input: number; output: number; }
  Float: { input: number; output: number; }
};

export type Query = {
  __typename?: 'Query';
  user: Maybe<User>;
};


export type QueryUserArgs = {
  input: UserInput;
};

export type User = {
  __typename?: 'User';
  id: Scalars['ID']['output'];
  name: Scalars['String']['output'];
  role: UserRole;
};

export type UserInput = {
  id: Scalars['ID']['input'];
};

export enum UserRole {
  Admin = 'ADMIN',
  User = 'USER'
}
```

```ts
export type UserQueryVariables = Exact<{ [key: string]: never; }>;


export type UserQuery = { user: { id: string } | null };
```

Designs / Issues

Unnecessary types

Common issue 1

```ts
export type Scalars = {
  ID: { input: string; output: string; }
  String: { input: string; output: string; }
  Boolean: { input: boolean; output: boolean; }
  Int: { input: number; output: number; }
  Float: { input: number; output: number; }
};

export type Query = {
  __typename?: 'Query';
  user: Maybe<User>;
};


export type QueryUserArgs = {
  input: UserInput;
};

export type User = {
  __typename?: 'User';
  id: Scalars['ID']['output'];
  name: Scalars['String']['output'];
  role: UserRole;
};

export type UserInput = {
  id: Scalars['ID']['input'];
};

export enum UserRole {
  Admin = 'ADMIN',
  User = 'USER'
}

export type UserQueryVariables = Exact<{ [key: string]: never; }>;


export type UserQuery = { user: { id: string } | null };
```

Designs / Issues

Unnecessary types

Common issue 2

```ts
export type Scalars = {
  ID: { input: string; output: string; }
  String: { input: string; output: string; }
  Boolean: { input: boolean; output: boolean; }
  Int: { input: number; output: number; }
  Float: { input: number; output: number; }
};

export type Query = {
  __typename?: 'Query';
  user: Maybe<User>;
};


export type QueryUserArgs = {
  input: UserInput;
};

export type User = {
  __typename?: 'User';
  id: Scalars['ID']['output'];
  name: Scalars['String']['output'];
  role: UserRole;
};

export type UserInput = {
  id: Scalars['ID']['input'];
};

export enum UserRole {
  Admin = 'ADMIN',
  User = 'USER'
}

export type UserQueryVariables = Exact<{ [key: string]: never; }>;


export type UserQuery = { user: { id: string } | null };
```

Designs / Issues

Unnecessary types

Common issue 2

```ts
export type Scalars = {
  ID: { input: string; output: string; }
  String: { input: string; output: string; }
  Boolean: { input: boolean; output: boolean; }
  Int: { input: number; output: number; }
  Float: { input: number; output: number; }
};

export type Query = {
  __typename?: 'Query';
  user: Maybe<User>;
};


export type QueryUserArgs = {
  input: UserInput;
};

export type User = {
  __typename?: 'User';
  id: Scalars['ID']['output'];
  name: Scalars['String']['output'];
  role: UserRole;
};

export type UserInput = {
  id: Scalars['ID']['input'];
};

export enum UserRole {
  Admin = 'ADMIN',
  User = 'USER'
}

export type UserQueryVariables = Exact<{ [key: string]: never; }>;


export type UserQuery = { user: { id: string } | null };
```

Designs / Issues

Unnecessary types

```ts
export type Scalars = {
  ID: { input: string; output: string; }
  String: { input: string; output: string; }
  Boolean: { input: boolean; output: boolean; }
  Int: { input: number; output: number; }
  Float: { input: number; output: number; }
};

export type Query = {
  __typename?: 'Query';
  user: Maybe<User>;
};


export type QueryUserArgs = {
  input: UserInput;
};

export type User = {
  __typename?: 'User';
  id: Scalars['ID']['output'];
  name: Scalars['String']['output'];
  role: UserRole;
};

export type UserInput = {
  id: Scalars['ID']['input'];
};

export enum UserRole {
  Admin = 'ADMIN',
  User = 'USER'
}

export type UserQueryVariables = Exact<{ [key: string]: never; }>;


export type UserQuery = { user: { id: string } | null };
```

Designs / Issues

Universal options

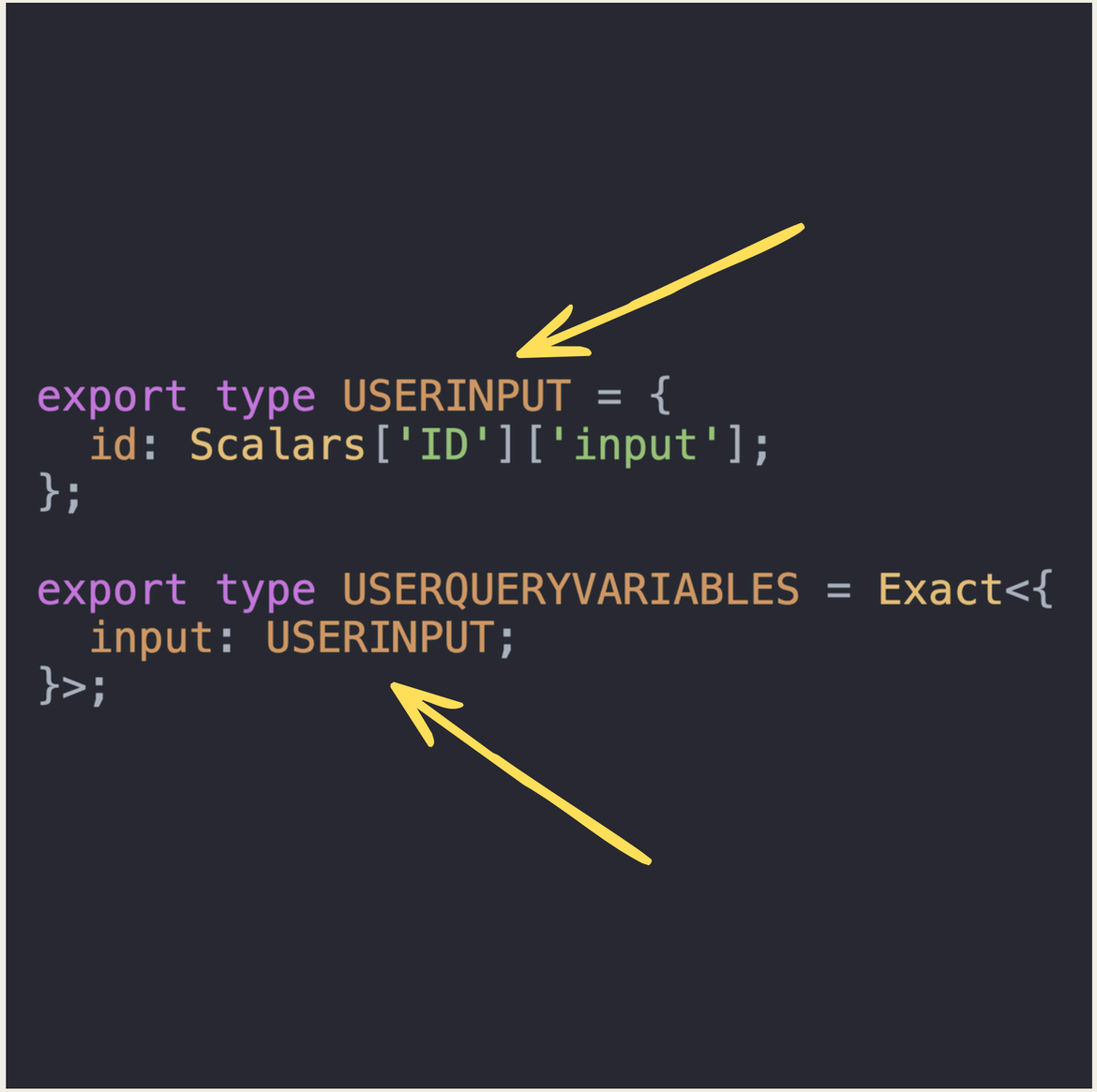
Designs / Issues

Universal options

```
const config: CodegenConfig {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
      "typescript-operations",
    ],
    config: {
      namingConvention:
        "change-case-all#upperCase"
    }
  }
}
```

Designs / Issues

Universal options



Designs / Issues

No-op
options

Designs / Issues

No-op options

avoidOptionals

- defaultValue
- object
- field
- inputValue
- resolvers
- query
- mutation
- subscription

Designs / Issues

No-op options

avoidOptionals

- defaultValue operations
- object operations
- field
- inputValue operations
- resolvers
- query
- mutation
- subscription

Designs / Issues

Generalised
defaults

Designs / Issues

Verbose
setup

Designs / Issues

Verbose setup

```
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
      "typescript-operations"
    ],
  },
}
```

Designs / Issues

Verbose setup

```
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
      "typescript-operations",
    ],
    config: {
      scalars: {
        ID: {
          input: "string | number",
          output: "string",
        },
        enumAsTypes: true,
        avoidOptionals: {
          field: true,
          inputValue: false,
        },
        defaultScalarType: "unknown",
      },
    },
  },
}
```

Designs / Issues

Verbose setup

```
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
      "typescript-operations",
    ],
    config: {
      scalars: {
        ID: {
          input: "string | number",
          output: "string",
        },
        enumAsTypes: true,
        avoidOptionals: {
          field: true,
          inputValue: false,
        },
        defaultScalarType: "unknown",
      },
    },
  },
}
```

Hive

Client plugins redesign

Design Principles

01

02

03

Design Principles

01

Generate necessary types

02

03

```graphql
query User {
  user(input: { id: "1" }) {
    id
  }
}
```

```typescript
export type Scalars = {
  ID: { input: string; output: string; }
  String: { input: string; output: string; }
  Boolean: { input: boolean; output: boolean; }
  Int: { input: number; output: number; }
  Float: { input: number; output: number; }
};

export type Query = {
  __typename?: 'Query';
  user: Maybe<User>;
};


export type QueryUserArgs = {
  input: UserInput;
};

export type User = {
  __typename?: 'User';
  id: Scalars['ID']['output'];
  name: Scalars['String']['output'];
  role: UserRole;
};

export type UserInput = {
  id: Scalars['ID']['input'];
};

export enum UserRole {
  Admin = 'ADMIN',
  User = 'USER'
}

export type UserQueryVariables = Exact<{ [key: string]: never; }>;


export type UserQuery = { user: { id: string } | null };
```

```graphql
query User {
  user(input: { id: "1" }) {
    id
  }
}
```

```typescript
export type UserQueryVariables =
  Exact<{ [key: string]: never; }>;

export type UserQuery = {
  user: {
    id: string
  } | null
};
```

```
query User {
  user(input: { id: "1" }) {
    id
    role
  }
}
```

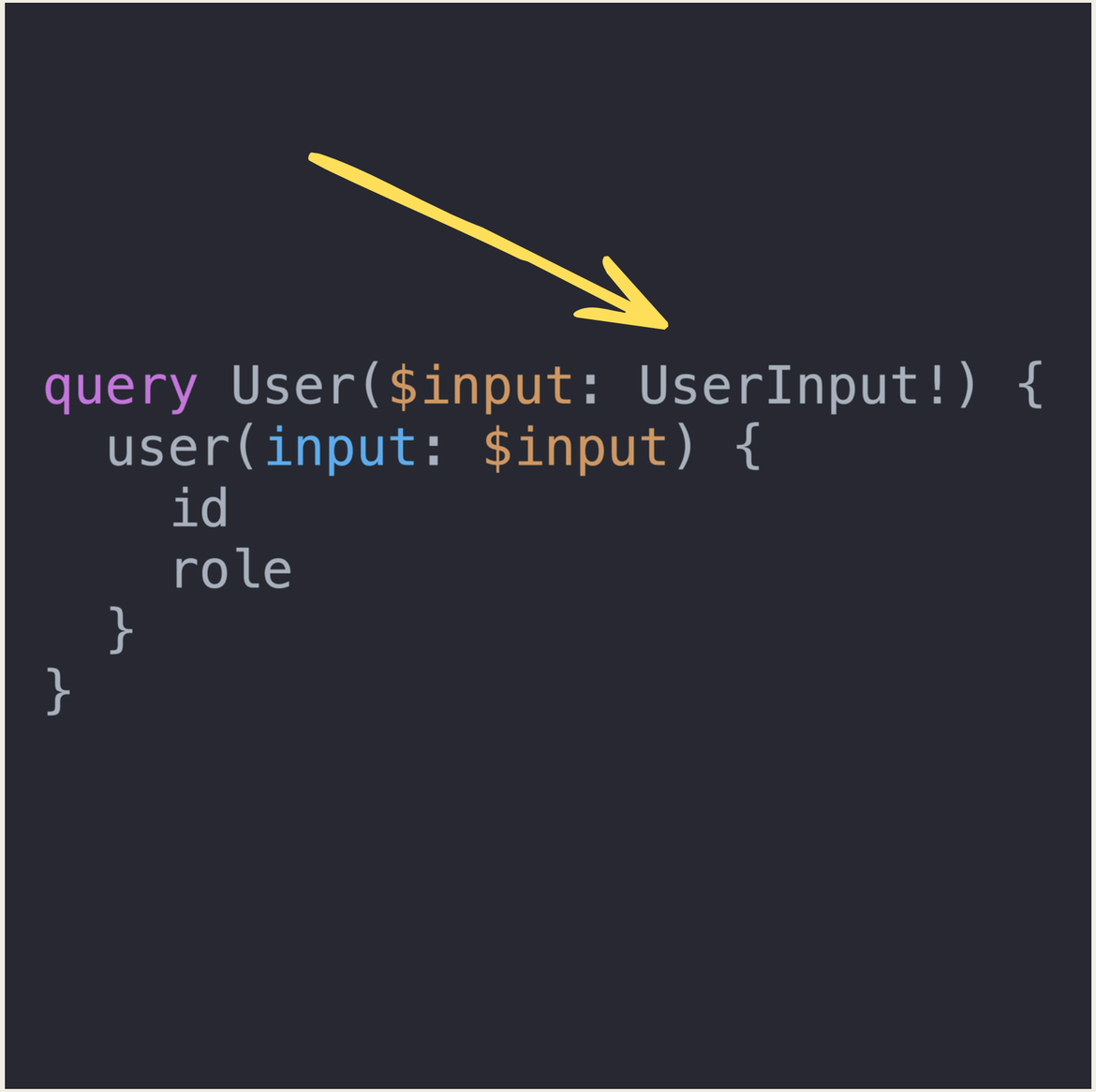
```graphql
query User {
  user(input: { id: "1" }) {
    id
    role
  }
}
```

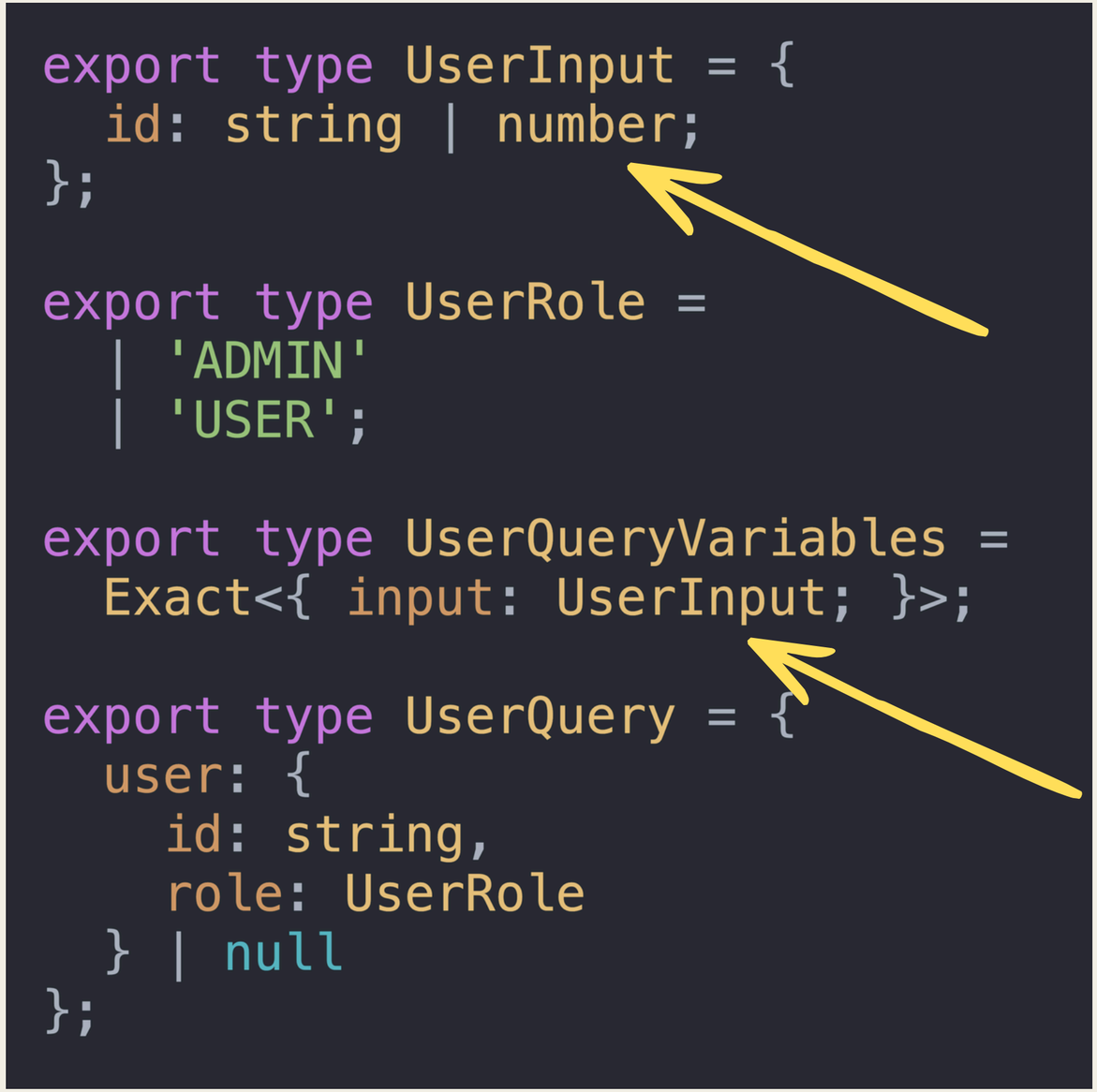
```typescript
export type UserRole =
  | 'ADMIN'
  | 'USER';

export type UserQueryVariables =
  Exact<{ [key: string]: never; }>;

export type UserQuery = {
  user: {
    id: string,
    role: UserRole
  } | null
};
```

```
query User($input: UserInput!) {
  user(input: $input) {
    id
    role
  }
}
```


query User($input: UserInput!) {
user(input: $input) {
id
role
}
}


export type UserInput = {
id: string | number;
};
export type UserRole =
| 'ADMIN'
| 'USER';
export type UserQueryVariables =
Exact<{ input: UserInput; }>;
export type UserQuery = {
user: {
id: string,
role: UserRole
} | null
};

Design Principles

01

Generate necessary types

02

03

Design Principles

01

Generate necessary types

02

Specialized options

03

```
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
      "typescript-operations",
    ],
  }
}
```

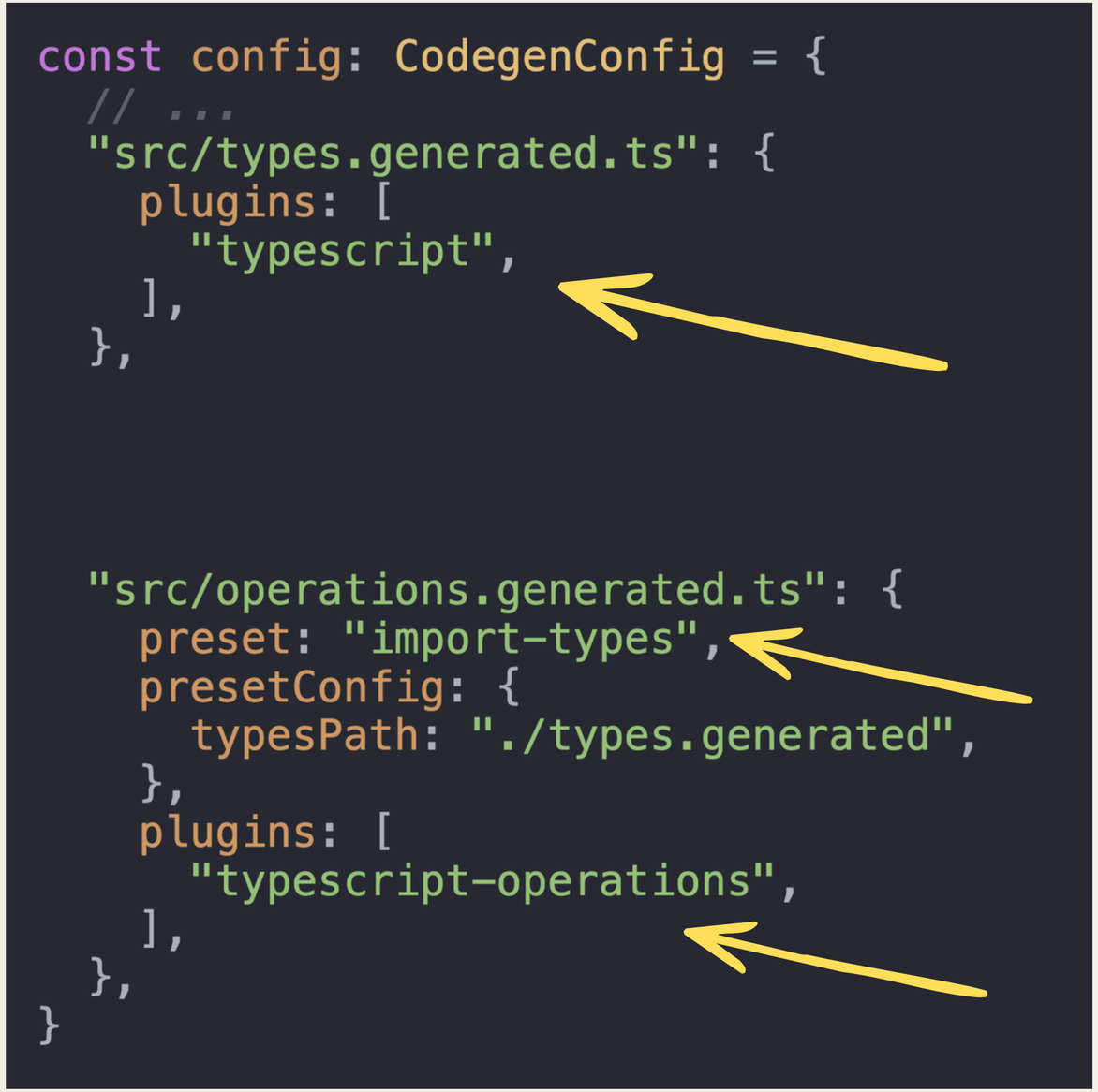
```ts
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
      "typescript-operations",
    ],
  }
}
```


```ts
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript-operations",
    ],
  }
}
```

```ts
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
    ],
  },



  "src/operations.generated.ts": {
    preset: "import-types",
    presetConfig: {
      typesPath: "./types.generated",
    },
    plugins: [
      "typescript-operations",
    ],
  },
}
```


const config: CodegenConfig = {
// ...
"src/types.generated.ts": {
plugins: [
"typescript",
],
},
"src/operations.generated.ts": {
preset: "import-types",
presetConfig: {
typesPath: "./types.generated",
},
plugins: [
"typescript-operations",
],
},
}

```ts
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript",
    ],
  },

  "src/operations.generated.ts": {
    preset: "import-types",
    presetConfig: {
      typesPath: "./types.generated",
    },
    plugins: [
      "typescript-operations",
    ],
  },
}
```

```ts
const config: CodegenConfig = {
  // ...
  "src/types.generated.ts": {
    plugins: [
      "typescript-operations",
    ],
    config: {
      generateOperationTypes: false,
    },
  },

  "src/operations.generated.ts": {
    plugins: [
      "typescript-operations",
    ],
    config: {
      importSchemaTypesFrom:
        "src/types.generated.ts",
    },
  },
}
```


WHY WASTE TIME USE LOT PLUGIN
WHEN FEW PLUGIN DO TRICK

Design Principles

01

Generate necessary types

02

Specialized options

03

Design Principles

01

Generate necessary types

02

Specialized options

03

Community-driven defaults

Result

-95%

Lines of generated code reduced

In Hive Console codebase

14

Notable breaking changes

Options, CLI improvements, ESM-first, etc.

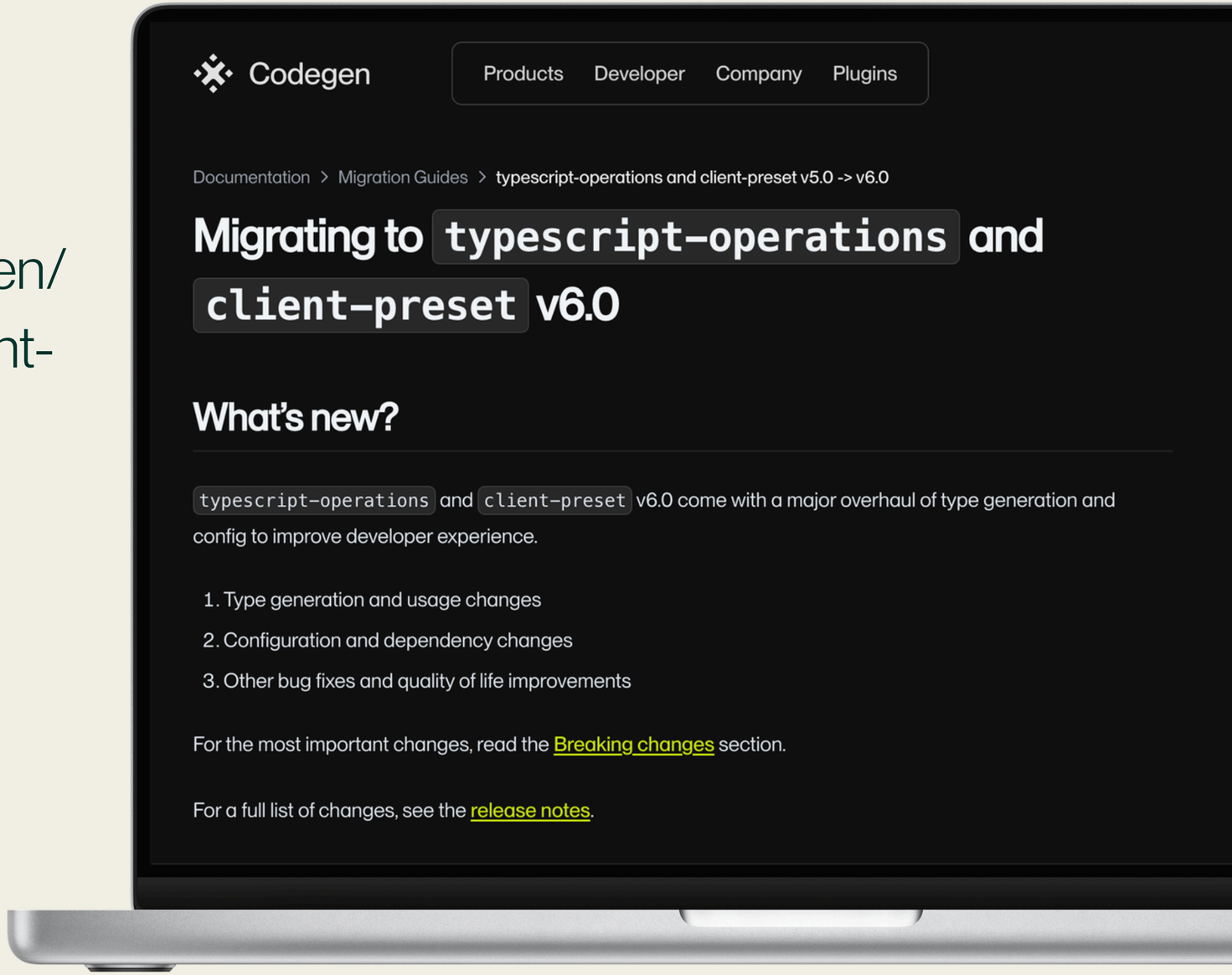
3

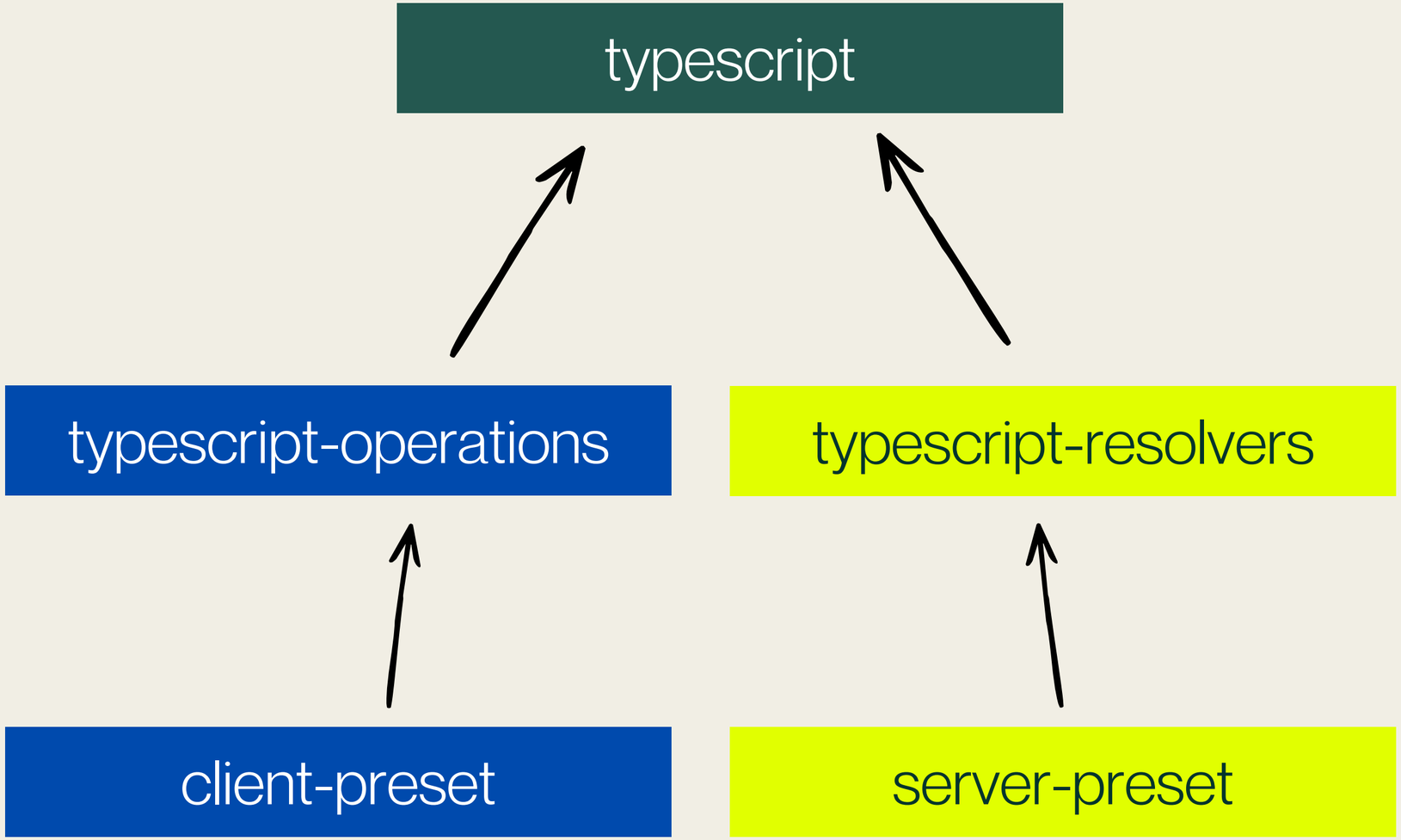
Curated setup

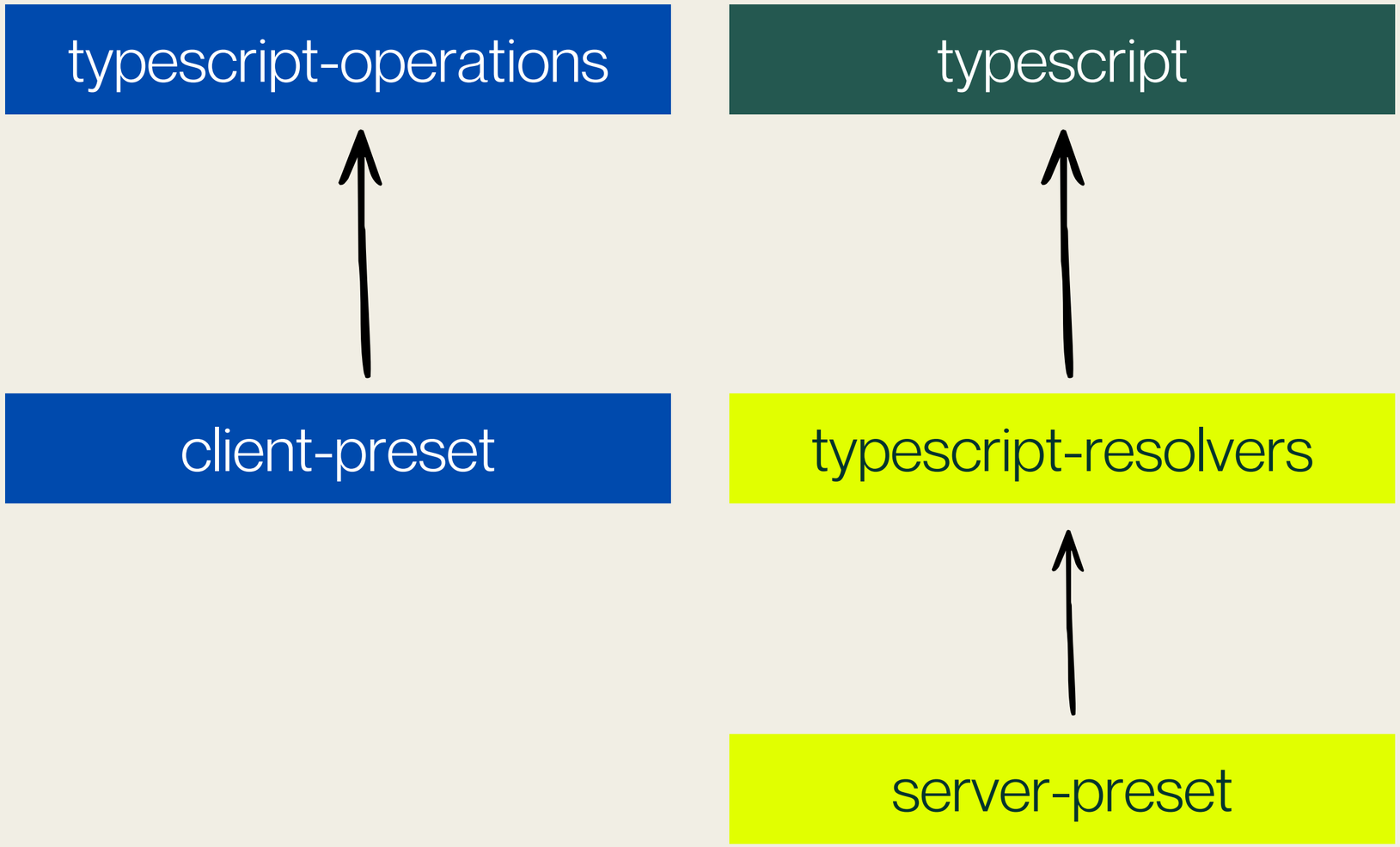
One-file, multi-file, near-operation-file

Migration Guide

https://the-guild.dev/graphql/codegen/docs/migration/operations-and-client-preset-from-5-0




typescript
typescript-operations
typescript-resolvers
client-preset
server-preset


typescript-operations
typescript
client-preset
typescript-resolvers
server-preset

MIGRATING APOLLO TOOLING TO GRAPHQL CODEGEN AT YELP

yelp

Overview

- At Yelp, we rely heavily on **GraphQL**
- Yelp frontend monorepo: **hundreds of packages**, **thousand of source files**
- We use codegen to produce TypeScript types
- We initially used **apollo-tooling** in 2020
- Apollo-tooling got deprecated in favor of **GraphQL Codegen**

```
src/components/UserProfile/
├── UserProfile.tsx          ← contains the GetUser query
└── __generated__/
    └── GetUser.ts           ← auto-generated types: GetUser, GetUserVariables
```

Differences

Difference	Apollo-tooling	GraphQL Codegen
Generated file name	`./__generated__/GetUser.ts`	`./__generated__/UserProfile.ts`
Nested types names	`GetUser_user_account`	`GetUser_user_User_account_Account`
Custom types	none	`Maybe<>, InputMaybe<>, Exact<>, Scalar[...]`
Includes from the global schema: enums, input types	inline	From external file (globalTypes.ts)

Implementation

- “**Black box**” approach: a lot of fragile code that would be difficult to maintain
- We discovered a **concurrency issue** - which could not be fixed with a “black box” approach.
- Next approach: we **forked and modified** the relevant GraphQL Codegen plugins:
  - **`typescript-operations`** `plugin`
  - **`near-operation-file`** `preset`
- We wanted to upstream our modifications to the official GraphQL Codegen project.

Apollo-Tooling Compatible Config for Codegen 6.0

```ts
const config: CodegenConfig = {
  schema: …GraphQLSchema,
  documents: ['./src/**/*.{ts,tsx}', …],
  externalDocuments: [
    '../other-package/src/**/*.{ts,tsx}', …],
  generates: {
    ‘./src/': {
      preset: 'near-operation-file',
      presetConfig: {
        extension: '.ts',
        folder: '__generated__',
        filePerOperation: true
      },
    plugins: ['typescript-operations'],
  },
  config: {
    extractAllFieldsToTypesCompact: true,
    namingConvention: 'keep',
    printFieldsOnNewLines: true,
    enumType: 'native',
    nonOptionalTypename: true,
    skipTypeNameForRoot: true,
    omitOperationSuffix: true,
    fragmentSuffix: '',
    defaultScalarType: 'any'
  }
}
```

https://the-guild.dev/graphql/codegen/docs/migration/apollo-tooling

yelp

Hive

yelp

The collaboration

You want to upstream your changes, how would you do that?

How to contact the maintainers:

- Check the repo’s contribution guidelines
- Write first messages as good as you can:
  - Write clearly
  - Structure the message well
  - Provide the simplest code examples
  - Don’t write too much
  - Don’t send right away, wait at least a day
- Be open to different outcomes (it's not your project, and you don't have the full picture)
- Look for other contact methods (Discord, Twitter/X, etc.)

How we collaborate:

- Write a proposal in a GitHub Discussion
- Get approval
- Create an initial PR and request a review
- Eddy will then either:
  - Approve and merge the PR,
  - Make changes and merge it, or
  - Implements a completely different solution.
- Keep the same attitude:
  - Write clearly
  - Be polite and friendly
  - Be open to different outcomes (it's not your project, and you don't have the full picture)

Collaboration benefits:

For Yelp:

- All the features that we proposed got upstreamed! (we don’t need to maintain them in-house)

For GraphQL Codegen:

- Ideas, discussions, and contributions
- A large Yelp codebase for testing

For the community:

- All the improvements!

Thanks Eddy
for the great collaboration!

yelp

Hive

Let's talk!

Hive

The future

1. Correct types
2. Fast
3. Great DX

Upcoming focus

01

Improve Client Preset

02

Improve CLI and Watcher

03

Improve Server Preset

04

Improve docs and discoverability

Hive

yelp

THANK YOU