DevhubService

September 5, 2026

DevhubService

Kind: Service

Source: `atloria-monorepo/apps/api/src/reader-account/devhub.service.ts`

C3 owner "Developers" dashboard — org-side aggregate view over the reader plane (signups, TTFC funnel, top consumers). Row-level identity stays limited to the reader's email; call rows are only ever aggregated here (the detailed status/path log is the READER'S view of their own traffic).

Access control lives at the controller: JwtAuthGuard + ResourceOrgGuard (project must belong to the caller's org) + PlanGuard

`DevhubService` powers the C3 owner “Developers” dashboard by aggregating reader-plane activity for an organization’s projects. It provides signup and TTFC funnel metrics plus top consumer summaries, while preserving privacy boundaries by exposing reader identity only as email and returning call activity only in aggregated form.

Methods

Method	Signature	Returns
`summary`	`summary(projectId: string)`	`Promise<DevhubSummary>`
`readers`	`readers(projectId: string, opts: { limit?: number; offset?: number })`	`Promise<{ total: number; rows: DevhubReaderRow[] }>`

Dependencies

- `PrismaService`

Diagram

```
sequenceDiagram
    participant Owner as Organization Owner
    participant Controller as Devhub Controller
    participant Guards as JWT / Org / Plan Guards
    participant Service as DevhubService
    participant ReaderDB as Reader-plane Data

    Owner->>Controller: Request dashboard summary or readers
    Controller->>Guards: Validate JWT, org project access, plan
    Guards-->>Controller: Authorized request
    Controller->>Service: summary() / readers()
    Service->>ReaderDB: Query signup, TTFC, and consumer aggregates
    ReaderDB-->>Service: Aggregated reader-plane data
    Service-->>Controller: DevhubSummary or reader rows
    Controller-->>Owner: Organization dashboard response
```

Usage

```
import { DevhubService } from './devhub.service';

@Injectable()
export class DevhubController {
  constructor(private readonly devhubService: DevhubService) {}

  @Get('summary')
  async getSummary() {
    // Controller guards should validate JWT, organization ownership,
    // and plan access before this service is called.
    return this.devhubService.summary();
  }

  @Get('readers')
  async getReaders() {
    const { total, rows } = await this.devhubService.readers();

    return {
      total,
      readers: rows,
    };
  }
}
```

AI Coding Instructions

- Keep dashboard queries aggregate-focused; do not expose detailed reader call status or request-path logs through this service.

- Preserve the privacy boundary: reader-level identity is limited to email, and call activity must remain summarized.
- Assume authorization is enforced by the controller's `JwtAuthGuard`, `ResourceOrgGuard`, and `PlanGuard`; do not duplicate unrelated guard logic in service methods.
- When adding metrics, ensure they are scoped to the authorized organization/project context and use reader-plane data sources consistently.
- Maintain stable response shapes for `DevhubSummary` and `DevhubReaderRow`, since they support the organization-facing Developers dashboard.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `DevhubController` (DEPENDS_ON)
- `ReaderAccountModule` (MODULE_PROVIDES)