ChatGPT

API pública de Railway (GraphQL) y autenticación

Railway ofrece una **API pública GraphQL** (idéntica a la que usa su panel web) en el endpoint `https://backboard.railway.com/graphql/v2` [1]. Para acceder a ella es necesario un token de API, que puede ser de cuenta personal, de equipo o específico de proyecto [2]. Los tokens de equipo/cuenta se envían en el encabezado `Authorization: Bearer <token>` y otorgan acceso a los recursos correspondientes [3], mientras que los tokens de proyecto (scoped a un entorno) usan `Project-Access-Token: <token>` [4]. Railway provee herramientas como un **playground GraphiQL** (por ejemplo en `railway.com/graphiql` [5]) y colecciones en Postman para explorar el esquema. La documentación oficial describe estos tokens y límites de uso (p.ej. 100 peticiones/hora en plan gratuito) [6].

- **Tokens de acceso**: Se pueden crear *tokens de equipo* (buscan en recursos del equipo) o *tokens personales* (acceso a todos los proyectos del usuario) [3]. También hay *tokens de proyecto* limitados a un único entorno [4].
- **Límite de uso**: Railway aplica límites de tasa (RPH/RPS) según el plan [6]. Por ejemplo, Free=100 RPH; Hobby=1000 RPH; Pro=10000 RPH [6].

Estado de despliegues y del proyecto

La API GraphQL permite consultar el estado de los despliegues de un proyecto. Por ejemplo, una consulta al objeto `environment(id: $envId)` retorna entre otros campos el despliegue más reciente y su `status` (como `"DEPLOYING"`, `"ACTIVE"`, `"FAILED"`, etc.) [7]. En la práctica se puede leer `environment { serviceInstances { node { latestDeployment { status } } } }`. **Railway no tiene un estado "pausado" de proyecto**; la única forma de detenerlo es eliminando el despliegue activo [8]. En la comunidad oficial se indica que no existe función de pausa, y que lo más cercano es remover despliegues activos [8]. Para ver el estado actual de cada servicio se suele usar `latestDeployment.status` [7].

- **Estados posibles**: Los despliegues en Railway pasan por estados como *Initializing*, *Building*, *Deploying*, *Active*, *Failed*, *Crashed*, *Removing*, etc. (ver lista completa en la documentación) [9] [7].
- **Ejemplo de consulta**: En un caso real la consulta devolvió `"status": "DEPLOYING"` para el último despliegue de un servicio [7]. Esto indica que el proyecto está en proceso de desplegar.

Logs de los contenedores

Railway captura los logs estándar de cada build/despliegue para consulta posterior. Programáticamente **se pueden obtener vía GraphQL** usando la consulta `deploymentLogs`. Por ejemplo:

```
query {
  deploymentLogs(deploymentId: "...", filter: "@level:error") {
    message
    severity
```

```
  }
}
```

Esta consulta (mostrada en el playground de Railway) devuelve los mensajes de log de un despliegue dado, permitiendo aplicar filtros (p.ej. `@level:error`) [10]. Además, la CLI oficial de Railway incluye `railway logs` para ver los logs del despliegue más reciente desde la terminal [11]. También existe `buildLogs` para los logs de la etapa de construcción. En resumen, se puede acceder a los registros tanto vía API GraphQL como mediante la CLI (y la interfaz web).

- **Vía GraphQL**: `deploymentLogs(deploymentId: ...)` devuelve todos los registros de ese despliegue [10]. El campo `message` contiene el texto del log. Se admite filtrado por nivel de severidad (warn, error, etc.) usando la sintaxis `@level:error` [10]. También hay soporte para **suscripciones** (WebSocket) para escuchar nuevos logs en tiempo real, por ejemplo:

  ```
  subscription {
    deploymentLogs(deploymentId: "...") {
      message
      severity
    }
  }
  ```

  tal como demuestra el ejemplo de código del equipo de Railway [12].
- **Vía CLI**: Ejecutando `railway logs` en la terminal se muestran los logs del despliegue actual [11]. Para logs de un despliegue pasado específico o de otro servicio, se puede combinar con otros subcomandos del CLI.
- **Web UI**: También existe un explorador de logs en el tablero web, pero la pregunta es sobre acceso programático. En cualquier caso, todo el histórico de logs está disponible para consultas y filtro por atributos [13].

Métricas (CPU, memoria, disco, tráfico)

Railway registra métricas de uso de recursos en cada servicio. La documentación indica que se proveen métricas de **CPU**, **Memoria**, **Disco** y **Red** [14]. Programáticamente, muchas de estas métricas están accesibles vía GraphQL:

- **Uso de CPU/memoria/tráfico**: La consulta `usage` permite obtener, para un proyecto y rango de fechas, métricas de uso agregadas. Por ejemplo, un usuario reportó haber obtenido valores aproximados de *uso de CPU*, *uso de memoria (GB)* y *tráfico de red* desde `usage(projectId, ...)` [15]. En ese caso logró extraer ~11.60 (unidades de CPU) y ~1020 (GB-minutos) de uso de memoria en un periodo dado [15]. Estos valores concuerdan con las gráficas de uso mostradas en el panel de Railway.
- **Uso de disco (volúmenes)**: La consulta GraphQL `volumeInstance(id: $volId)` devuelve la capacidad y uso actual de un volumen asignado. Por ejemplo:

```
query VolumeMetrics($volumeId: String!) {
  volumeInstance(id: $volumeId) {
    currentSizeMB
    sizeMB
  }
}
```

Esto retorna el uso actual (`currentSizeMB`) y el total asignado (`sizeMB`) del volumen [16]. Un ingeniero de Railway confirmó esta consulta en el foro de ayuda [16]. (El `volumeId` se obtiene desde la UI o mediante consultas previas de la API).

- **Otras métricas**: Existe también un objeto `metrics` en la API que puede usarse para obtener métricas históricas por proyecto/servicio. En la interfaz de Postman de Railway aparece una consulta `metrics` descrita como "Get metrics for a project, environment, and service". Aunque no hemos encontrado documentación pública detallada de ella, su presencia indica que sí hay soporte GraphQL para métricas personalizadas. Adicionalmente, Railway ofrece métricas agrupadas en su panel de "Observabilidad" (logs + métricas) tal como muestra la documentación [14].

En resumen, los datos de uso de CPU, memoria, disco y tráfico *sí* son accesibles vía API (a través de las consultas `usage` y `volumeInstance`), además de poder ser vistos en el dashboard web. No hay un endpoint REST tradicional; todo se maneja por GraphQL o herramientas relacionadas.

Limitaciones conocidas y consideraciones

- **Límites de la API**: Como se mencionó, existen límites de tasa (requests per hour/second) según el plan del usuario [6]. Los tokens de proyecto solo validan para el entorno designado [4]. Las consultas GraphQL complejas (p.ej. con muchas entidades) pueden requerir ajustar parámetros de paginación.
- **Estados no representados**: Railway no expone en la API estados como "pausado" o "aparcado" de un proyecto; para detener un entorno hay que eliminar sus despliegues activos manualmente [8]. Tampoco hay un endpoint específico de facturación; los datos de costo se extraen de métricas (p.ej. `usage` / `estimatedUsage`) y convierten externamente.
- **Métodos alternativos**: Además de la API GraphQL oficial, Railway proporciona **webhooks** para integrarse con servicios externos: por ejemplo, se puede configurar un webhook HTTP para que Railway notifique automáticamente cuando cambia el estado de un despliegue [17]. Esto permite reaccionar a eventos sin hacer polling. También se dispone de un **CLI oficial** (`railway`) que expone comandos útiles (`railway status`, `railway logs`, etc. para ver estado y logs desde terminal) [11]. No existe (hasta la fecha) un SDK oficial aparte; usualmente los integradores usan el cliente GraphQL de su preferencia o wrappers no oficiales.
- **Soporte de integraciones externas**: Dado que la API GraphQL es la misma que usa el panel web, ***sí hay soporte real para integraciones externas*** [5] [1]. Todo lo mostrado en la UI (estado de servicios, logs, métricas, límites) es accesible vía API. No es necesario "scrapear" la web; basta usar la API o CLI documentada.

Ejemplos de endpoints/queries:
- Endpoint base: `https://backboard.railway.com/graphql/v2` [1].
- Query de estado:

```
query {
  environment(id: "<environmentId>") {
    serviceInstances {
      edges { node {
        serviceName
        latestDeployment { id status }
      }}
    }
  }
}
```

- Query de logs:

```
query {
  deploymentLogs(deploymentId: "<deploymentId>") {
    timestamp message severity
  }
}
```

- Query de métricas:

```
query {
  usage(projectId: "<projectId>", from: "...", to: "...") {
    cpu
    memoryGB
    networkTxGB
  }
  volumeInstance(id: "<volumeId>") {
    currentSizeMB sizeMB
  }
}
```

En conclusión, Railway sí ofrece acceso programático completo a estado, logs y métricas mediante su API GraphQL pública [1] [10] (junto con herramientas complementarias como webhooks y CLI). Estas capacidades permiten crear dashboards externos o automatizaciones sin depender únicamente de la interfaz web.

Fuentes: Documentación oficial de Railway sobre la API pública [1] [6], guías de logs y métricas [14] [11], y discusiones en el foro/estación de ayuda de Railway [5] [16] [7].

---

[1] [2] [3] [4] [6] Public API Reference | Railway Docs
https://docs.railway.com/reference/public-api

5 15 Loading...
https://station.railway.com/questions/is-there-a-way-to-access-per-project-cdf37adc

7 Update number of instances via API - Railway Help Station
https://station.railway.com/questions/update-number-of-instances-via-api-2499832f

8 Loading...
https://station.railway.com/questions/is-there-a-way-to-pause-stop-a-project-d5174901

9 Deployments | Railway Docs
https://docs.railway.com/reference/deployments

10 12 Loading...
https://station.railway.com/questions/graph-ql-severity-filter-f4da5050

11 13 Viewing Logs | Railway Docs
https://docs.railway.com/guides/logs

14 Metrics | Railway Docs
https://docs.railway.com/reference/metrics

16 Loading...
https://station.railway.com/questions/using-railway-api-to-monitor-current-and-4db97264

17 Webhooks | Railway Docs
https://docs.railway.com/reference/webhooks