TECH 42

Postgres Text-to-SQL MCP Server

Deployment Guide

For PostgreSQL on AWS ECS Fargate

1. Overview

The Tech 42 Postgres Text-to-SQL MCP Server is a production-grade Model Context Protocol (MCP) server for PostgreSQL. It gives AI assistants a safer, more useful database tool surface — enabling natural language SQL generation, schema inspection, query optimization, and database health diagnostics without exposing raw database credentials to client applications.

It supports two operating models:

- **Admin Mode** — for development, DBA work, index tuning, and controlled production diagnostics.
- **Tenant Mode** — for exposing a narrow, tenant-scoped, read-only SQL interface over HTTP with row-level security.

1.1 Key Features

Feature	Description
Schema Discovery	List schemas, tables, views, sequences, columns, constraints, and indexes
SQL Execution	Run SQL in unrestricted mode, or read-only SQL in restricted mode
Query Plans	Run EXPLAIN and test hypothetical indexes with HypoPG
Index Tuning	Analyze queries or workloads and recommend candidate indexes
Database Health	Check index, buffer, connection, vacuum, replication, and sequence health
Top Queries	Inspect slow or resource-intensive queries from pg_stat_statements
Tenant-Scoped SQL	Expose only configured relations with server-side validation and RLS enforcement

1.2 Access Modes

Mode	Flag	Best For
Unrestricted	--access-mode=unrestricted	Local development, disposable databases
Restricted	--access-mode=restricted	Production diagnostics, read-only operations
Tenant Mode	(separate config)	Multi-tenant apps with row-level security

1.3 Required PostgreSQL Extensions

Install these extensions to unlock the full performance-analysis feature set:

```
CREATE EXTENSION IF NOT EXISTS pg_stat_statements;
CREATE EXTENSION IF NOT EXISTS hypopg;
```

- **pg_stat_statements** — required for workload and top-query analysis.
- **hypopg** — required for hypothetical index planning.

1.4 Usage Examples

The following examples show common prompts you can use with your AI assistant once the MCP server is connected.

Get Database Health Overview

```
Check the health of my database and identify any issues.
```

Analyze Slow Queries

```
What are the slowest queries in my database? And how can I speed them up?
```

Get Recommendations On How To Speed Things Up

```
My app is slow. How can I make it faster?
```

Generate Index Recommendations

```
Analyze my database workload and suggest indexes to improve performance.
```

Optimize a Specific Query

```
Help me optimize this query:
SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id
WHERE orders.created_at > '2023-01-01';
```

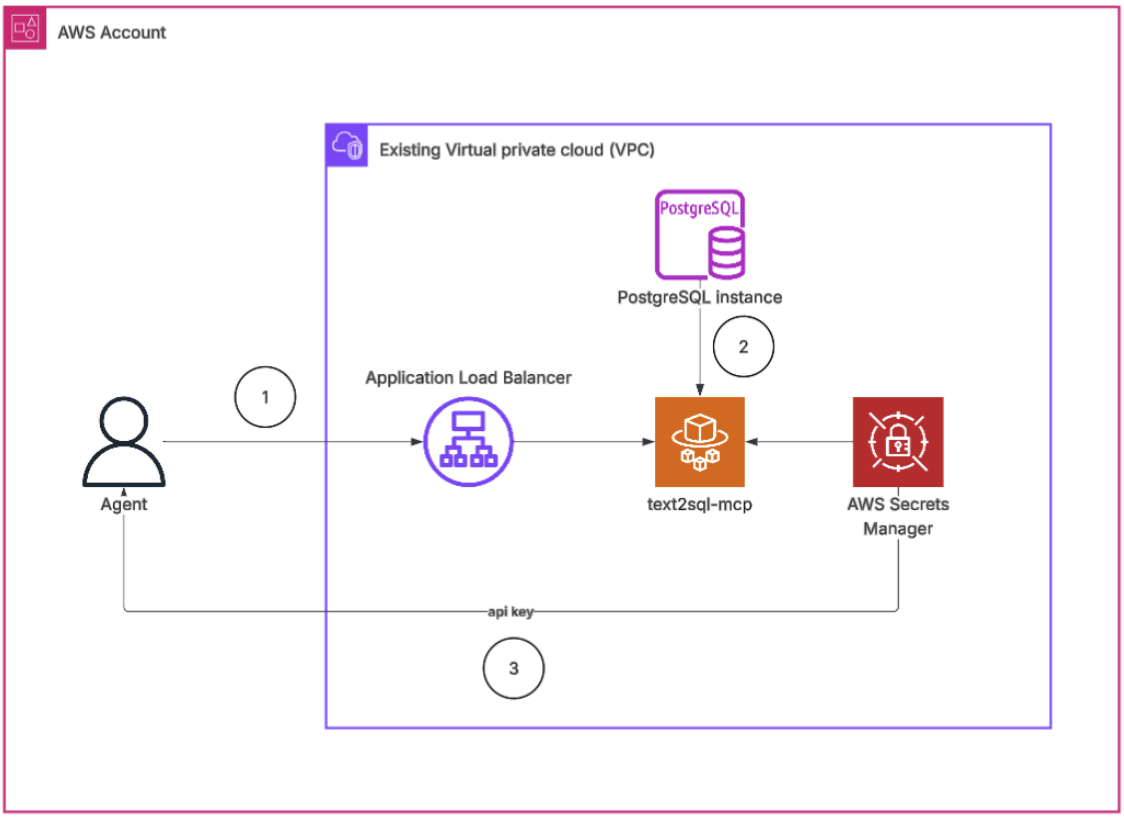
1.5 Tools

The following tools are available:

Tool Name	Description
`list_schemas`	Lists all database schemas available in the PostgreSQL instance.
`list_objects`	Lists database objects (tables, views, sequences, extensions) within a specified schema.
`get_object_details`	Provides information about a specific database object, such as a table's columns, constraints, and indexes.
`execute_sql`	Executes SQL statements on the database, with read-only limitations when connected in restricted mode.
`explain_query`	Gets the execution plan for a SQL query, describing how PostgreSQL will process it and exposing the query planner's cost model. Can be invoked with hypothetical indexes to simulate behavior after adding indexes.
`get_top_queries`	Reports the slowest SQL queries based on total execution time using pg_stat_statements data.
`analyze_workload_indexes`	Analyzes the database workload to identify resource-intensive queries, then recommends optimal indexes for them.
`analyze_query_indexes`	Analyzes a list of specific SQL queries (up to 10) and recommends optimal indexes for them.
`analyze_db_health`	Performs comprehensive health checks including: buffer cache hit rates, connection health, constraint validation, index health (duplicate/unused/invalid), sequence limits, and vacuum health.

2. Deployment Instructions

2.1 Architecture



System Components:

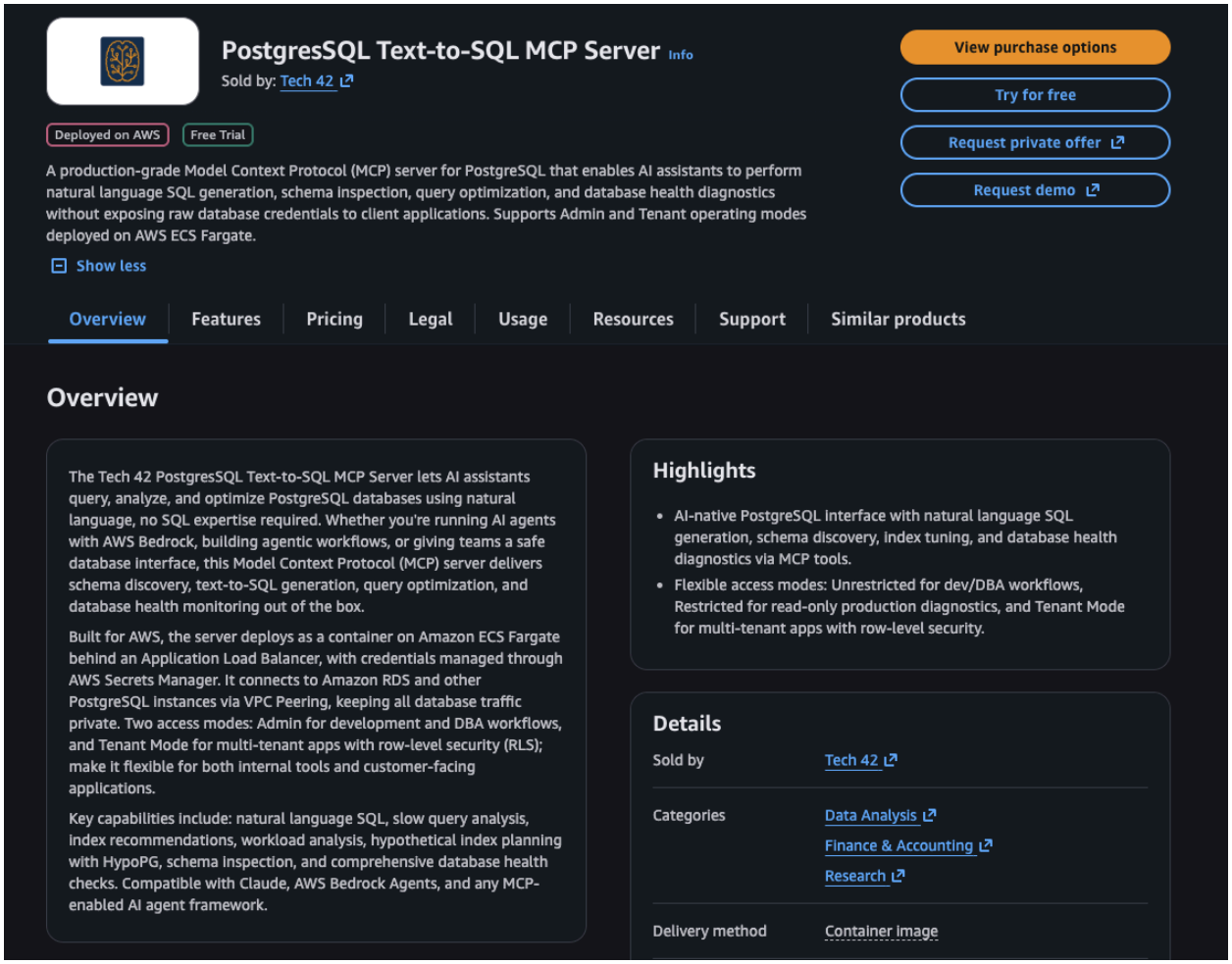
1. **Agent** — the MCP client (AI assistant or developer tool) that initiates requests from outside the VPC.
2. **Application Load Balancer** — routes inbound MCP traffic from the agent directly to the text2sql-mcp container.
3. **text2sql-mcp** — the ECS Fargate task running the MCP server; it manages protocol handling and natural language SQL translation.
4. **PostgreSQL instance** — your internal RDS or PostgreSQL database located within the VPC targeted by the MCP server.
5. **AWS Secrets Manager** — manages the database URI and auto-generated API keys, injecting them securely at task startup.

Logical Data Flows:

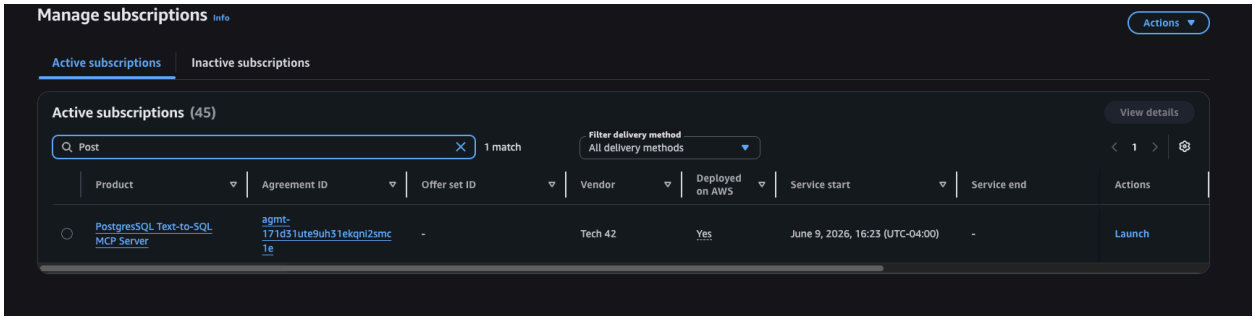
1. The Agent transmits an MCP request through the ALB to the task container using HTTP/SSE.
2. The container establishes a connection to PostgreSQL on port 5432 to execute the generated SQL.
3. The Agent provides the required `x-api-key` header for authentication, retrieved out-of-band from Secrets Manager.

2.1 Subscribe

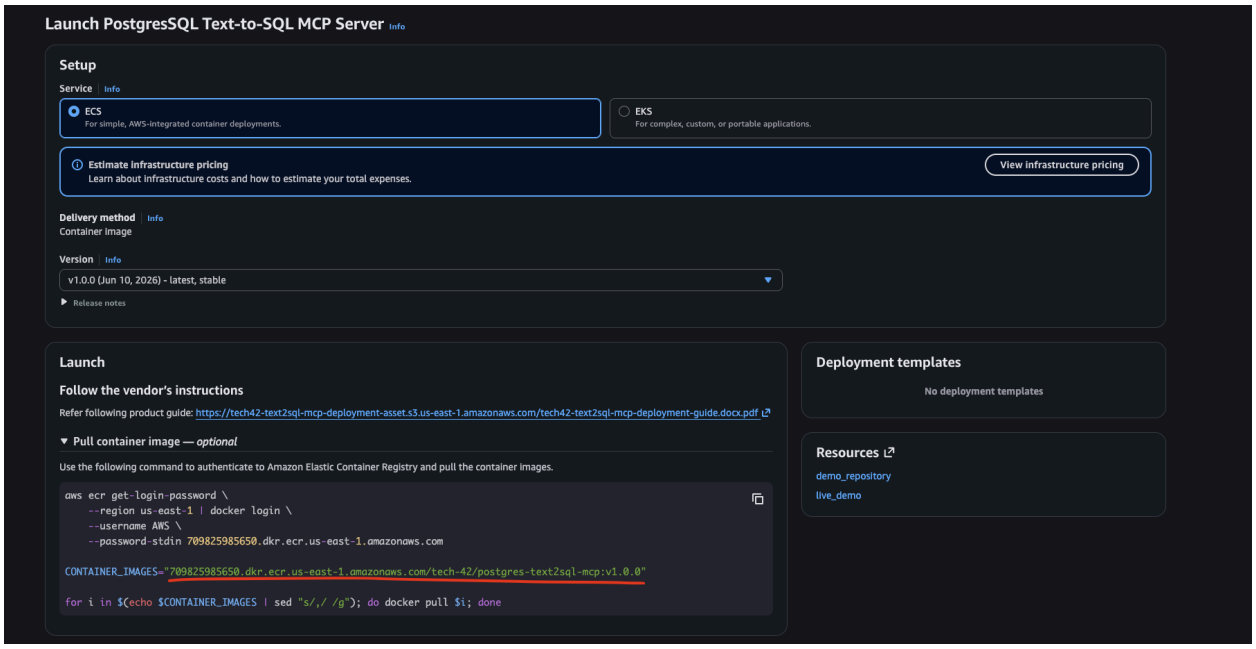
1. Go to https://aws.amazon.com/marketplace
2. Search for PostgresSQL Text-to-SQL MCP Server and subscribe to the product. By default you get a 30 day trial after subscribing to it.



3. After you have subscribed, go to AWS Marketplace inside your AWS account -> Manage subscriptions -> find the product -> Launch



4. Select ECS -> select your desired version -> copy and paste the container image uri



5. Use either the cloudformation template or the example repo provided below guide you on the deployment

2.1 CloudFormation Template

Click the link below to open the CloudFormation template for us-east-1 deployment

Launch Stack in AWS Console

2.3 Example

Reference github.com/tech42-org/postgres-mcp-demo for sample deployment and demo

2.4 Live Demo

Live demo can be found here https://sqlagent.tech42demo.com/