

warning "Backend update (October 2026)" GitHub Models inference was retired on July 30, 2026. Any references below are retained only for historical context while this lab is migrated. Use Lab 013 ? Free LLM backends to configure Foundry Local or Ollama before continuing.

Lab 031: pgvector Semantic Search on Azure

Level: L300 Path: RAG Time: ~60 min ? Cost: ~\$15/month (PostgreSQL Flexible Server Burstable B1ms)

warning "Azure subscription required" This lab deploys Azure PostgreSQL Flexible Server. ?

Prerequisites guide

What You'll Learn

Deploy Azure PostgreSQL Flexible Server with pgvector enabled via one click Connect securely with SSL and firewall rules Migrate the RAG pipeline from Lab 022 to Azure Use Azure AI Services for embeddings (or keep using GitHub Models ? free) Expose the vector store as an MCP tool for agents

Deploy Infrastructure

Option A ? Deploy to Azure (one click)

Deploy to Azure

This deploys: PostgreSQL Flexible Server (Burstable B1ms, ~\$15/month) pgvector extension enabled Database ragdb pre-created Azure services firewall rule

Parameters you'll set: administratorLoginPassword ? your database password (note it down!)

location ? choose a region close to you

Option B ? Azure CLI (Bicep)

[diagram or code sample]

? Supporting Files

note "Download these files before starting the lab" Save all files to a lab-031/ folder in your working directory.

File | Description | Download | 001_init.sql | Database migration | ? Download |

Lab Exercise

Step 1: Set connection environment variables

After deployment, save your connection details:

[diagram or code sample]

Step 2: Connect and verify pgvector

[diagram or code sample]

Step 3: Create schema

[diagram or code sample]

Step 4: Ingest the sample dataset

Download and ingest the OutdoorGear sample data:

[diagram or code sample]

Reuse the chunking and embedding contract from Lab 022, then replace only the in-memory storage adapter with Azure Database for PostgreSQL:

[diagram or code sample]

Step 5: Expose as an MCP tool

Wrap the search function as an MCP tool so any agent can query your knowledge base:

[diagram or code sample]

Add to .vscode/mcp.json: [diagram or code sample]

Cleanup

[diagram or code sample]

? Knowledge Check

question "Q1 (Run the Lab): After running 001_init.sql against your database, how many rows are in the products table? Run SELECT COUNT(*) FROM products; to verify."

Run the migration then query the table.

??? success "?" Reveal Answer" 7 rows

The migration seeds 7 OutdoorGear products: P001 (TrailBlazer Tent 2P), P002 (Summit Dome 4P), P003 (TrailBlazer Solo), P004 (ArcticDown -20°C), P005 (SummerLight +5°C), P006 (Osprey Atmos 65L), P007 (DayHiker 22L). Run SELECT id, name, category FROM products ORDER BY id; to see them all.

question "Q2 (Run the Lab): What type of index is created on the product_embeddings table, and which column does it index?"

Open lab-031/migrations/001_init.sql and find the CREATE INDEX statement.

??? success "?" Reveal Answer" IVFFlat index on the embedding column

The migration creates: CREATE INDEX ON product_embeddings USING ivfflat (embedding vector_cosine_ops) WITH (lists = 100);

IVFFlat (Inverted File with Flat quantization) is an approximate nearest neighbor index ?

it's faster than exact search but sacrifices a small amount of recall. The lists = 100

parameter defines the number of Voronoi cells for clustering. vector_cosine_ops means

distances are computed using cosine similarity.

question "Q3 (Multiple Choice): You run SELECT * FROM search_products_by_vector(\$1::vector, 3) and get 3 results. What do the results represent?"

- A) The 3 most recently inserted products - B) The 3 products with the highest price - C)

The 3 products whose embedding vectors are most similar (closest cosine distance) to the

query vector - D) 3 randomly selected products from the database

??? success "?" Reveal Answer" Correct: C

The search_products_by_vector() function performs an approximate nearest neighbor (ANN)

search using the IVFFlat index. It computes cosine distance between the query vector and

all stored product embeddings, then returns the k products with the smallest distance (=

highest semantic similarity). The result represents the most semantically relevant products

for the user's query.

Next Steps

Row Level Security for multi-tenant agents: ? Lab 032 ? Row Level Security Agentic RAG on

top of this: ? Lab 026 ? Agentic RAG Pattern

Supporting kit

The site build generates stable PDF and ZIP downloads. The ZIP contains the lab guide and every supporting file declared for this lab.

Download generated PDF Download generated ZIP