MCKNIGHT CONSULTING GROUP

Data Platform for RAG-based Agentic AI: Total Cost of Ownership

EnterpriseDB DIY AWS Stack

RELATED RESEARCH:

Self-Hosted Database NoSQL Performance Testing: EnterpriseDB and MySQL
Redpanda vs. Confluent: A Performance and TCO Benchmark Report

William McKnight
Jake Dolezal

JUNE 2025

Table of Contents

Executive Summary

When developing generative and agentic AI solutions for the enterprise, it is crucial to choose the right data platform for operations like vector search and Retrieval-Augmented Generation (RAG) as it directly impacts performance, efficiency, and effectiveness. A well-designed AI data pipeline ensures optimal retrieval, accurate generation, and efficient processing, enabling scalability, flexibility, and customization. By selecting the right data platform, you can improve user experience, enhance system performance, and increase adoption, ultimately unlocking AI's full potential and achieving desired outcomes.

Companies are comparing on-premises and cloud-based solutions for enterprise GenAI use cases like RAG. This study compares the Total Cost of Ownership (TCO) of deploying PostgreSQL pipelines supporting a modern AI workload including RAG between two data platform approaches:

- **Sovereign, on-premises solution**: Using EDB' integrated platform, EDB Postgres AI on customer-owned hardware
- **Public cloud solution**: A do-it-yourself (DIY) approach on AWS with managed services.

The study evaluates costs, development complexity, and operational effort for building and managing PostgreSQL deployments for AI-enabled applications with vector embeddings. It provides insights into cost trade-offs and complexity between a platform solution with EDB Postgres AI and a DIY public cloud approach.

We found that EDB Postgres AI is an extremely functional and cost-effective option for an enterprise data and AI platform. EDB Postgres AI offers **67% lower development complexity and 38% lower maintenance complexity, enabling development three times faster.** Additionally, EDB Postgres AI **reduces total cost of ownership by 51%** compared to a DIY cloud-based solution, requiring a smaller team and **over 50% less people costs**. Although the on-premises EDB Postgres AI solution requires upfront hardware costs, it proves prudent, making EDB Postgres AI a sovereign, efficient and cost-effective choice for enterprise GenAI applications.

With these advantages, EDB Postgres AI stands out as a compelling solution for companies seeking to optimize their RAG deployments and accelerate their time-to-value for AI applications. By leveraging EDB Postgres AI, businesses can streamline development, reduce costs, and improve overall efficiency.

Introduction

This Total Cost of Ownership study evaluated the financial implications for an enterprise that is fairly dedicated to PostgreSQL. We developed a test that is representative of a modern enterprise use of a data pipeline solution for generative AI.

We used all six components crucial for building a robust and scalable Retrieval-Augmented Generation (RAG) system: Database & AI for knowledge storage and retrieval, Data Lake for storing and processing large datasets, Security & Compliance for ensuring data protection and regulatory adherence, Observability & Monitoring for tracking performance and identifying issues, Distributed HA Microservices for scalability and reliability, and Message Queue for efficient data processing and communication between services.

We calculated the full costs of our tests and projected that to an annual level to achieve annual TCO.

We chose 2 competitors to evaluate, including the most common competitor - a highly architected, public cloud deployment using AWS managed services – and the emerging possibility of a sovereign, on-premises platform solution using EDB Postgres AI, including its built-in Database, Hybrid Management, and AI Factory capabilities.

For the workload focus, we selected RAG, which is gaining popularity in natural language processing (NLP) and artificial intelligence (AI). Its importance lies in its potential to improve the accuracy and relevance of generated text by combining retrieval-based methods with generative models. RAG's ability to integrate external knowledge sources enables it to provide more informed and contextually relevant responses. This makes it a valuable tool for various NLP tasks, such as question answering, text summarization, and dialogue generation.

Companies are exploring both on-premises and cloud-based solutions for RAG, weighing the benefits of each. On-premises solutions offer enhanced data security, governance, and control over data access, suiting organizations with strict regulatory requirements. In contrast, cloud-based solutions generally provide scalability, flexibility, and are easier to implement.

Our test involved deploying multiple PostgreSQL pipelines—one supporting RAG applications with built-in monitoring, observability, and security.

Since vector embeddings have become a core component of modern AI-enabled applications, including RAG architectures, this study incorporates a comprehensive evaluation of time, effort, and ownership costs required for the build and ongoing management required to support these deployments. Here we provide a comprehensive view of the cost trade-offs (and development and operational complexity) between EDB Postgres AI and the DIY approach on cloud-managed platforms for scalable, secure, and intelligent AI workloads with PostgreSQL.

Platform Summary

Do-it-Yourself (DIY) AWS Stack

At the core of the DIY stack, Amazon RDS for PostgreSQL provides a managed relational database with support for AI workloads such as our vector search and RAG use case. Amazon S3 serves as the data lake, storing raw and processed data at scale, including documents and other staged data. AWS IAM (Identity and Access Management) enforces fine-grained security and compliance through identity and access controls across all services. Amazon CloudWatch provides centralized observability and monitoring for infrastructure and application metrics, enabling alerting and diagnostics. Our applications are deployed as highly available, distributed microservices on Amazon ECS (Elastic Container Services), allowing for flexible scaling and isolation of compute workloads. Inter-service communication and event-driven workflows are orchestrated using Amazon MSK (Managed Streaming for Apache Kafka), supporting messaging between components.

EnterpriseDB (EDB) Postgres AI

For our EnterpriseDB stack, we are using the single, integrated solution of EDB Postgres AI, which includes their Enterprise Postgres database, the Hybrid Manager for orchestrating, monitoring, and observing our Kubernetes-based databases and applications, plus an AI Factory for vector search and RAG. AI Factory is a bundle that includes a vector database extension (pgvector), automated AI pipelines (AIDB), an external data plugin for file-system document storage, Kserv model serving runtime, and a robust, low-code/no-code GenAI application builder for building AI workflows using large language models (LLMs).

Component	DIY	EDB
1. Database & AI	RDS PostgreSQL	EDB Postgres AI
2. Data Lake	Amazon S3	
3. Security & Compliance	AWS IAM	
4. Observability & Monitoring	Amazon CloudWatch	
5. Distributed HA Microservices	Amazon ECS	
6. Message Queue	Amazon MSK	

Table 1: DIY and EDB Stacks

Infrastructure and Licensing Costs

Pricing the DIY cloud solution is all usage-based, with ongoing monthly charges tied to compute, storage, data transfer, and service utilization, offering flexibility but potentially variable long-term costs. The one exception being that we used discounted all-upfront reserved instance pricing whenever possible, which is typical, as it saves on compute costs. In contrast, EDB Postgres AI involves an upfront capital expense for hardware (rack servers) and predictable annual licensing fees (e.g., per-core database licenses), providing cost stability but requiring an initial investment. One goal of the benchmark was to see if this initial investment was prudent, given the zero cost entry point of the DIY stack.

Pricing Assumptions

To price out the totals for each stack, we needed to apply several assumptions and constants to our calculations. To mimic a typical enterprise's use of these products, we used:

Databases	**10**
Transactional/Analytical	9
Vector/RAG	1
RDS/K8s DB Replication	2x
Database Size	1,024 GB
RDS Provisioned IOPS	6,000
vCPU/Cores per Database	32/16

Other Assumptions	
Raw Data/Document Corpus Size	1,024 GB
Parquet Compression	3x
S3 Put Size	4 MB
CloudWatch Monitoring Logs	100 GB
CloudWatch Dashboards	10
MSK Messaging Storage	100 GB
MSK Replication Factor	3x
ECS Embedding Service Replication	3x

Table 2: Database Assumptions and Constants

If your use case is more or less, you can take the costs up or down in an according percentage. The ratios should not change based on scale.

Totals Per Year

After figuring in our assumptions, the following table breaks down the annual costs of these stacks.

Stack	Component	Service	Annual Cost
DIY	**Database**	RDS PostgreSQL	$706,521.60
		Amazon KMS	$2,808.00
	AI + Distributed HA	Amazon ECS	$0.00
		EC2 Instance (Embedding)	$34,347.96
	Message Queue	Amazon MSK	$325,267.20
	Object Storage	Amazon S3	$9,690.08
	Observability and Monitoring	Amazon CloudWatch	$89,880.00
	Security and Compliance	Amazon IAM	$0.00
	Support	Support	$116,851.48
		DIY Total	**$1,285,366.32**
EDB	**Rack Servers**	On-premise Compute	*
	Database + Hybrid Control Plane	Hybrid Management Full + HA Distri	$311,040.00
	Database + Hybrid Control Plane + AI	AI Factory + GenAI Builder	$157,632.00
		EDB Total	**$468,672.00**

*One-time/first-year cost of $179,102.85

Table 3: Yearly stack cost breakdown

DIY

The pricing of this AWS-based stack is consumption-based and modular, with each service billed independently according to usage metrics. Amazon RDS for PostgreSQL charges based on instance type, storage provisioned, and I/O operations, while Amazon S3 charges for data stored, requests made (e.g., PUT/GET), and data transferred out.

AWS IAM is typically free of charge but incurs indirect costs through API request usage and associated services. Amazon CloudWatch pricing is based on metrics collected, logs ingested and stored, custom dashboards, and alarms. Amazon ECS incurs costs for the underlying compute (viz., EC2). Finally, Amazon MSK is priced by broker instance hours, storage per GB-month, and data transfer, with additional charges for provisioned throughput or custom configurations. We used the AWS US East 2 (Ohio) as the region for pricing purposes.

Component	Service	SKU	Rate	Unit
Database	RDS PostgreSQL	db.m7g.8xlarge 32vCPU	$1.97	Per Hour (Reserved 1-Year All Upfront)
		io2 Storage	$0.25	Per GB-Month
		io2 IOPS	$0.20	Per IOPS-month
		Snapshot Exports	$0.01	Per GB Per Snapshot

	Amazon KMS	KMS Requests	$0.03	Per 10,000 Requests
AI + Distributed HA	Amazon ECS	EC2 Model (No separate fee)		
	EC2 Instance (for embedding)	r7i.8xlarge 32vCPU	$1.31	Per Hour
Message Queue	Amazon MSK	express.m7g.large	$0.408	Per Broker
		Storage Per Broker	$0.10	Per GB-Month
Object Storage	Amazon S3	S3 Standard Storage	$0.023	Per GB-Month
		S3 Requests PUTs	$0.005	Per 1,000 PUTs
		S3 Requests GETs	$0.0004	Per 1,000 GETs
Observability and Monitoring	Amazon CloudWatch	Dashboards	$3.00	Per Dashboard
		Database Insights	$0.0125	Per vCPU-hour
		Log Export to S3	$0.25	Per GB-Month
		Standard Logs Ingested	$0.50	Per GB-Month
Security and Compliance	Amazon IAM	(No separate fee)		
Support	Support	Enterprise On-Ramp Tier	10%	of Monthly Spend

Table 4: DIY Unit Charges

EDB Postgres AI

Using EDB Postgres AI on-premises requires an initial capital investment in hardware (with rack servers) along with EDB annual licensing fees based on per-core usage, figuring in Enterprise tier and Production support, along with a negotiated discount from EDB. This offered us lower costs and higher cost predictability over time but necessitated buying some hardware upfront. The pricing for our solution was as follows:

Component	Service	Rate	Unit
Rack Servers	Dell PowerEdge R760 2x16-core 64GB RAM	$35,820.57	Per Server
Databases + HCP	Hybrid Management Full + HA Distributed*	$2,160	Per Core**
Database + HCP with AI	Hybrid Management Full + HA Distributed + AI Factory*	$4,320	Per Core**
	+ GenAI Builder	$88,512	Per DB**

*includes Enterprise tier and Production support
**per year

Table 5: EDB Postgres AI Unit Charges

Time, Effort, and Complexity

Infrastructure, database, and software costs are only a portion of the overall total cost of platform ownership equation. Time, effort, and complexity must be considered. From the constants used in our pricing model above, we assumed 10 distinct databases. Nine of these are pre-existing databases with transactional and analytical workloads. If we migrated these databases to EDB Postgres AI from RDS or other PostgreSQL databases, we did not include time and effort to port them over, since EDB Postgres AI is fully PostgreSQL-compliant; it is built on the open-source PostgreSQL core and maintains full compatibility with PostgreSQL features and extensions. The tenth database is a new AI-enabled build that we did consider.

This study examines the time, effort, and architectural complexity involved in building the vector embedding database and Retrieval-Augmented Generation (RAG) application using PostgreSQL as the core data platform. The implementation encompassed several key components:

- **Source Document Ingest** - responsible for collecting and storing raw content
- **Document Preprocessing and Chunking** - cleans, segments, and prepares data for embedding
- **Embedding Microservice** - generates high-dimensional vectors from text using a machine learning mode
- **Retriever Service** - enables similarity search over the vector database, feeding relevant chunks into the Large Language Model
- **LLM Generation Service** - produces final responses

Supporting features included:

- Logs for traceability
- Metrics, Dashboards, and Alarms for monitoring and observability
- Access Control, Secrets & Encryption mechanisms to ensure secure, compliant operations

We considered these elements together to form the foundation for assessing the practical time and effort demands and technical complexity of developing a RAG system for GenAI on the 1) EDB Postgres AI and 2) DIY PostgreSQL on AWS stacks, and administering and maintaining the system while in production.

Vector Search and Retrieval-Augmented Generation (RAG) Program

The Vector Search and RAG program we designed is an enterprise knowledge assistant designed to help users quickly find answers buried in vast amounts of internal documentation—whether they be operational procedures, HR policies, technical manuals, or legal contracts. This RAG application goes beyond traditional keyword-based searches that often fall short in this context because they cannot effectively interpret varied terminology or retrieve contextually relevant passages based on natural language queries. Our RAG-based solution semantically understands user questions, retrieving the most relevant text chunks using vector similarity, and generating accurate, grounded answers.

Our process began with document ingestion and preprocessing. This involved collecting files from various sources, such as PDFs, DOCX files, emails, intranet sites, and databases. For our study we used a curated dataset from HuggingFace (KShivendu/dbpedia-entities-openai-1M).

The documents were already cleaned, normalized, and split into smaller semantically coherent chunks, mostly less than 100. Each chunk was enriched with metadata such as source, author, and date to support fine-grained filtering during retrieval. These text chunks were then converted into dense vector embeddings using a pre-trained language model. Specifically, we used all-MiniLM-L6-v2 for a 384-dimension embedding. The HuggingFace dataset also came with 1,536-dimension OpenAI embeddings which increased the representational capacity of the embeddings, allowing the model to capture finer-grained and more nuanced semantic distinctions between document chunks. The embeddings captured the semantic essence of its corresponding text and stored them in the competitors' PostgreSQL databases with the pgvector (DIY) and aidb (EDB Postgres AI) extensions, along with the original text and metadata.

When a user submits a question, the query is converted into a vector embedding using the same model. The system then performs a similarity search in the vector store to find the most semantically relevant chunks of text. These results can also be filtered based on metadata. The top N matching chunks are assembled into a context window, which is passed along with the original question to a large language model. The LLM can then be prompted to generate an answer based solely on the retrieved context, producing meaningful responses.

The architecture included several integrated components - storage for raw and processed documents, pipelines for text cleanup and chunking, vector embedding services, and vector databases for similarity search. The LLM or retriever service was hosted and accessed via API, and observability tools were used to track retrieval accuracy, monitor model drift, and ensure system performance. Security and compliance are also critical in these applications so we had data encrypted at rest and in transit, access role-controlled, sensitive information masked before being sent to LLMs, and all queries and responses logged for auditing.

This RAG-based approach provides numerous benefits. It increases accuracy by grounding answers in trusted internal content, improves efficiency by drastically reducing search time, and offers flexibility by understanding questions posed in natural language. For organizations with large and complex knowledge bases, a RAG-enabled assistant like we built for this test[1] becomes a valuable tool for knowledge workers, enabling scalable, accurate, and explainable decision support.

Methodology

To quantitatively measure time, effort, and complexity, we built the vector database and RAG application using EDB Postgres AI and the DIY stack, and used agile project management techniques to capture and break down the time and effort expended in completing the work.

Our agile project management techniques for this study included T-shirt sizing as a technique used to quantify the relative effort and complexity of development tasks using sizes from extra-small (XS) to 4X-large (4XL). These qualitative sizes were then converted to story points based on a modified Fibonacci sequence—for example, XS task = 1 point and 4XL task = 35 points—to provide a scalable, quantitative measure of effort. To estimate total work, we multiplied the story points by the number of expected iterations or repetitions of the

[1] Please contact us if we can build one for you.

task. Finally, by dividing the total estimated story points by the number of story points a developer can complete per week (team velocity), we forecasted delivery timelines and resource requirements accurately. This method,which we have used repeatedly in our field consulting practice rolling out projects of this kind, blends intuitive sizing with quantitative planning to support sprint-level and roadmap-level estimation.

Size	Points
XS	1
S	2
M	3
L	5
XL	8
2XL	13
3XL	21
4XL	35

Table 6: Story-sizing conversion chart used

Projecting to Production

The time and effort calculation does not stop once development is complete.

Development tasks typically involve a finite number of iterations to reach completion—each iteration representing a cycle of building, testing, refining, or reviewing a component until it meets the required functionality and quality standards. This phase is focused on feature creation and may span days or weeks depending on complexity. Once our RAG components are in production, it transitions from active development to steady-state operations, where the work shifts to ongoing administration, troubleshooting, and maintenance. These tasks are usually shorter in duration but occur at a regular frequency, such as applying patches, monitoring performance, resolving incidents, or responding to change requests. We also use story sizes and points to quantify this operational work that is more reactive and routine, aimed at ensuring stability, reliability, and compliance over time.

Time and Effort

Once we calculate all task story points, multiply development points by their iterations, and multiply production tasks by two-week sprints or intervals per year (26), we are able to estimate the total time, effort, and complexity of building our RAG solution with bothEDB Postgres AI and the DIY cloud stack and maintaining them once they were in production.

In the following table, the Development story point totals are a one-time, initial rollout effort for our single RAG application and vector database, which can require either professional services or in-house IT developers with the expertise to build such a solution. The Production story points are an ongoing effort for all 10 databases, which will require a headcount (in-house or otherwise) to handle.

DIY	Development Story Points (one-time)	Production Story Points (per year)
Source Document Ingest	13	78
Document Preprocessing and Chunking	13	52
Embedding Microservice	21	52
LLM Generation Service	63	78
Retriever Service	39	52
Logs	32	52
Metrics, Dashboards, and Alarms	13	26
Access Control, Secrets & Encryption	5	26
DIY Total	**199**	**416**

EDB	Development Story Points (one-time)	Production Story Points (per year)
Source Document Ingest	5	52
Document Preprocessing and Chunking	5	26
Embedding Microservice	8	26
LLM Generation Service	15	52
Retriever Service	15	26
Logs	12	26
Metrics, Dashboards, and Alarms	2	26
Access Control, Secrets & Encryption	3	26
EDB Total	**65**	**260**

Table 7: Development and Production Story Points

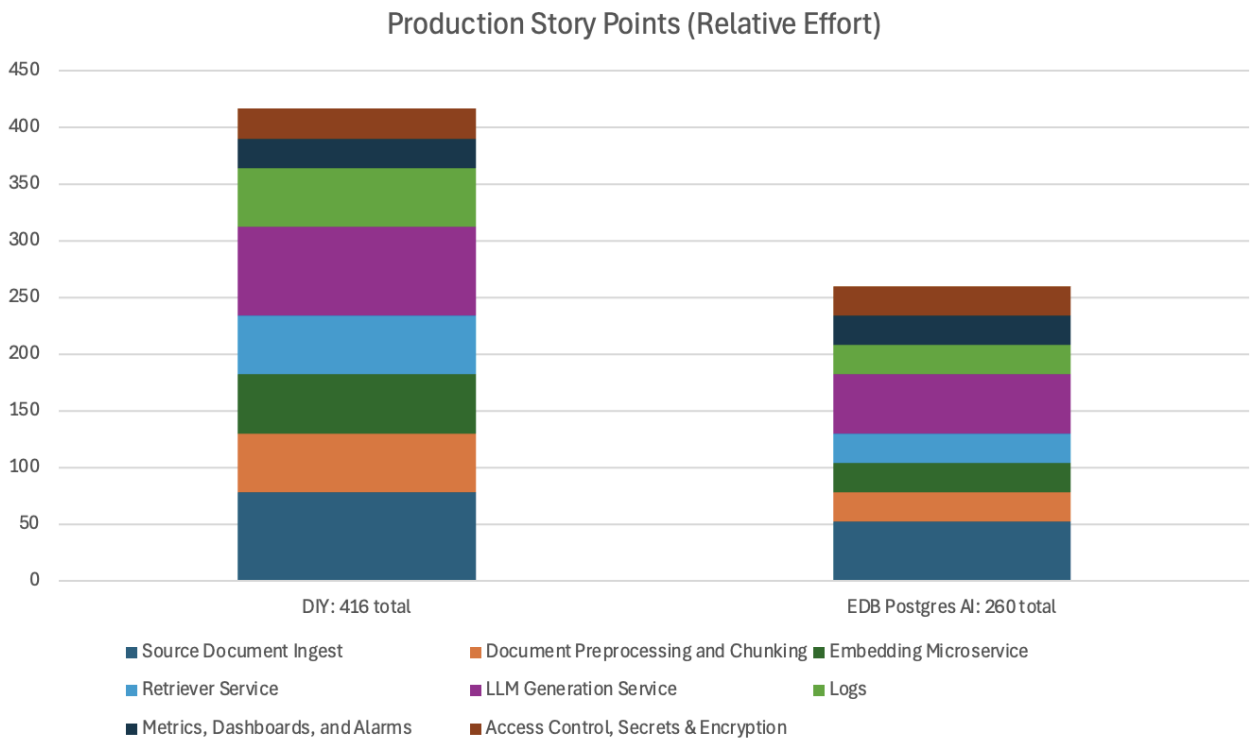


Figure 1: Production Story Points

EDB Postgres AI has a 67% lower development complexity and 38% lower complexity to maintain once in production.

Development and Production Full Time Employee (FTE) Headcounts

Once we know the total story points for development and production, we project:

- The estimated time-to-production development timeline, given a fixed developer team size, assuming you have a fixed budget for professional services (likely) or a fixed number of available IT developers with this expertise (less likely)
- The production Full Time Employee (FTE) headcount needed to maintain, troubleshoot, and administer all 10 databases in your data estate

Here are our assumptions:

Story Points Per Developer FTE Per Week	10
Development Team FTE (Headcount)	7
Development Professional Services Hourly Rate	$175
Production Inhouse IT Salary	$150,000
Employee Benefits	22%
Work Hours Per Week	40

Table 8: Assumption in Labor Cost Conversions

Given these values, here is the comparison between EDB and DIY:

Totals	**DIY**	**EDB Postgres AI**
Time to Production	28 weeks	9 weeks
Production FTE	8.0	5.0
Development Time & Effort Costs (one-time)	$1,393,000	$455,000
Production Time & Effort Costs (per year)	$1,233,000	$783,000

Table 9: Labor Cost Comparison

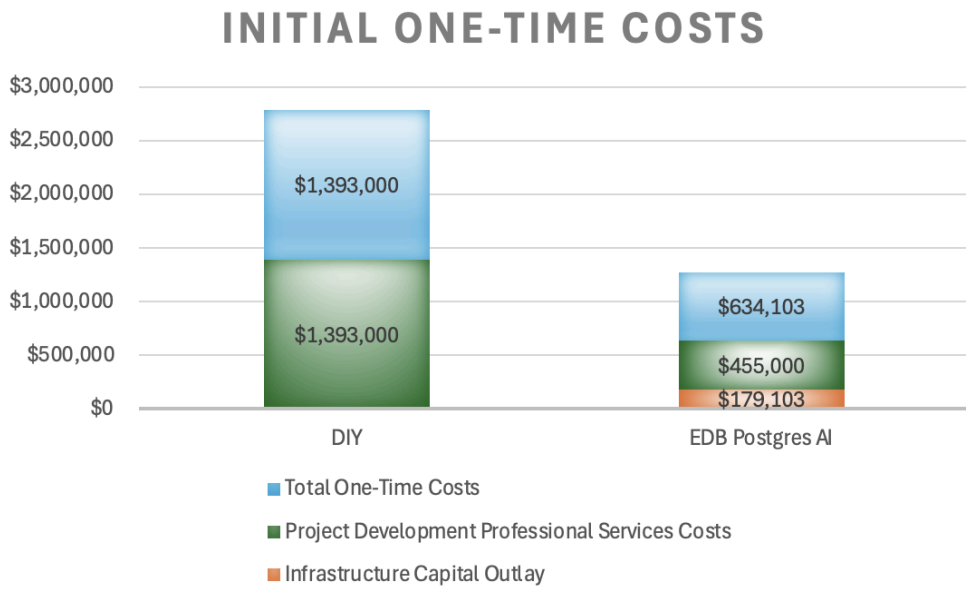
You can be up and running 3 times faster with EDB Postgres AI. Develop your RAG application and maintain your data estate with a smaller post-production team, reducing your people costs by over 50%.

Three-Year Total Cost of Ownership

Bringing it all together, this table shows the total cost of ownership to procure infrastructure, develop a vector database and RAG application, and maintain a data estate with nine other transactional and analytical databases.

Initial One-Time Costs	DIY	EDB Postgres AI
Infrastructure Capital Outlay	$0	$179,103
Project Development Professional Services	$1,393,000	$455,000
Total One-Time Costs	**$1,393,000**	**$634,103**
Weeks to Production	**28**	**9**

Table 10: One-Time Costs



EDB Postgres AI, with its advanced tooling, makes it 3 times faster to develop AI-enabled applications like RAG.

Annual Ongoing Costs	DIY	EDB Postgres AI
Infrastructure Per Year	$1,285,366	$468,672
Production FTE Salary & Benefits Per Year	$1,233,000	$783,000
Total Annual Costs	**$2,518,366**	**$1,251,672**

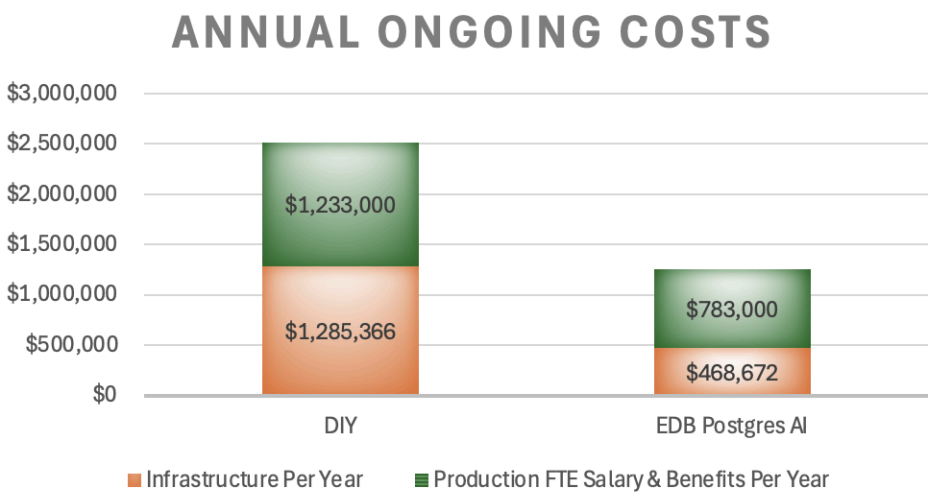


Figure 2: Annual Ongoing Costs

We estimate the DIY cloud stack requires approximately double the three-year total cost of ownership, compared to using EDB Postgres AI.

	DIY	EDB Postgres AI
Total Cost of Ownership	**$8,948,098.97**	**$4,389,118.85**

Table 11: Total Cost of Ownership

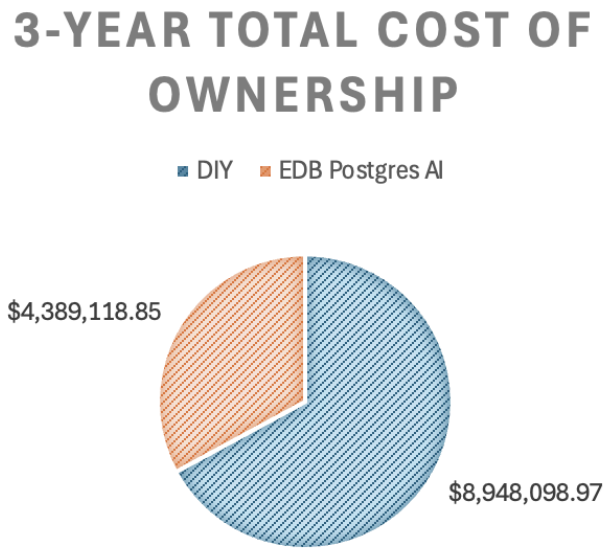


Figure 3: 3-Year Total Cost of Ownership

Overall, there is a 51% reduction in total cost of ownership with EDB Postgres AI over a do-it-yourself solution in the cloud.

Conclusion

Companies are comparing on-premises and cloud-based solutions for their pipeline solution including Retrieval-Augmented Generation (RAG), with EDB Postgres AI) emerging as a cost-effective platform option. EDB Postgres AI offers **67% lower development complexity** and **38% lower maintenance complexity**, enabling **development three times faster**. Additionally, EDB Postgres AI **reduces total cost of ownership by 51%** compared to a DIY cloud-based solution on AWS, requiring a smaller team and **over 50% less people costs**. Although deploying EDB Postgres AI on-premises requires upfront hardware costs, it proves prudent, making EDB Postgres AI a more efficient and cost-effective choice for RAG-based agentic and generative AI applications.

About EDB

EDB Postgres® AI is the first open, enterprise-grade sovereign data and AI platform, with a secure, compliant, and fully scalable environment, on premises and across clouds. Supported by a global partner network, EDB Postgres AI unifies transactional, analytical, and AI workloads, enabling organizations to operationalize their data and LLMs where, when, and how they need it.

About McKnight Consulting Group

Information Management is all about enabling an organization to have data in the best place to succeed to meet company goals. Mature data practices can integrate an entire organization across all core functions. Proper integration of that data facilitates the flow of information throughout the organization which allows for better decisions – made faster and with fewer errors. In short, well-done data can yield a better run company flush with real-time information… and with less costs.

However, before those benefits can be realized, a company must go through the business transformation of an implementation and systems integration. For many that have been involved in those types of projects in the past – data warehousing, master data, big data, analytics - the path toward a successful implementation and integration can seem never-ending at times and almost unachievable. Not so with McKnight Consulting Group (MCG) as your integration partner, because MCG has successfully implemented data solutions for our clients for over a decade. We understand the critical importance of setting clear, realistic expectations up front and ensuring that time-to-value is achieved quickly.

MCG has helped over 100 clients with analytics, big data, master data management and "all data" strategies and implementations across a variety of industries and worldwide locations. MCG offers flexible implementation methodologies that will fit the deployment model of your choice. The best methodologies, the best talent in the
industry and a leadership team committed to client success makes MCG the right choice to help lead your project.

MCG, led by industry leader William McKnight, has deep data experience in a variety of industries that will enable your business to incorporate best practices while implementing leading technology. See www.mcknightcg.com.

Disclaimer

McKnight Consulting Group (MCG) runs all its tests to strict ethical standards. The results of the report are the objective and unbiased results of the application of queries to the simulations described in the report. The report clearly defines the selected criteria and process used to establish the field test. The report also clearly states the data set sizes, the platforms, the methods, etc. that were used. The reader is left to determine for themselves how to qualify the information for their individual needs. The report does not make any claims regarding third-party certification and presents the objective results received from the application of the process to the criteria as described in the report. The report strictly measures TCO and does not purport to evaluate other factors that potential customers may find relevant when making a purchase decision. This is a sponsored report. The client chose its configuration, while MCG chose the test, configured the database and testing application, and ran the tests. MCG also chose the most compatible configurations for the other tested platforms. Choosing compatible configurations is subject to judgment. The information necessary to replicate this test is included. Readers are encouraged to compile their own representative configuration and test it for themselves.