

Optimize performance and budget for AI applications with Microsoft Azure

A single cloud approach on Azure offers performance and cost benefits and the potential for better security and development velocity compared to a multi-cloud approach with AWS

Organizations around the world are embracing the power of AI, especially RAG-based AI apps and agentic AI, to supercharge their business processes and drive innovation. While some organizations may employ a multi-cloud approach to building and hosting these AI apps due to familiarity or legacy that takes advantage of different cloud service provider (CSP) strengths or, this approach often comes with unexpected costs. For example, using Azure to get the latest and most popular OpenAI models from Azure OpenAI in Azure AI Foundry, but hosting your AI workloads on Amazon Web Services (AWS™), might cost you in terms of both performance and budget. Switching to a single cloud approach with Azure for your next OpenAI RAG LLM app can boost performance while saving costs and centralizing key parts of the development workflow.

Using Azure OpenAI in Azure AI Foundry Models allows organizations to also take advantage of robust integrations with the rest of the Azure cloud platform, such as security and development tools, along with the performance and cost benefits of OpenAI's models. We evaluated a single cloud vs multi cloud AI application deployment for key considerations such as performance, total cost of ownership, and security. For performance testing, we built a simple retrieval-augmented generation (RAG) AI application and hosted it on both AWS and Azure using roughly equivalent services, with both applications using the GPT-4o mini model in Azure OpenAI. Our performance tests showed that the Azure application had faster responses, a slightly higher output tokens per second rate, and superior search performance with Azure AI Search compared to the AWS application. This report discusses the benefits of that performance advantage, which include providing a better overall user experience with your next AI app. We also discuss how switching to Azure as a single CSP for an AI application could mitigate costs and security challenges.

Improve user experience for your RAG application with Azure

Up to 59.7% less time to execute application end-to-end compared to an AWS deployment

Get a faster search service layer for your RAG application with Azure

Up to 88.8% less time spent in Azure AI Search compared to Amazon Kendra

Save money and improve security

by choosing to deploy your OpenAI application on Azure vs. choosing a multi-cloud deployment

Comparing ease of deployment

About RAG applications

Natural language processing (NLP) applications are not new in the AI landscape. After all, Apple released their voice assistant, Siri, in 2011, nearly 14 years ago.[1] Chatbots on websites and AI-assisted search engines have also become familiar sights in the online world. They have continued to involve and improve, especially with the introduction of the many chatbots based on large language models (LLMs) such as ChatGPT.[2] Retrieval and context augmentation, by which organizations can augment pre-trained models with their own data, is now an integral part of AI applications that range from simple chat-based apps to more complex agentic AI workflows. Using this approach, businesses can now save time while enhancing the quality of their AI solutions. Use cases for RAG-based applications include chatbots, voice assistants, code generation, and more.[3]

While these applications can help save time, money, and effort, they can be difficult to deploy. Challenges to consider include:

About Azure AI Foundry

Azure AI Foundry is Microsoft's unified platform for designing, customizing, and managing secure generative AI applications and agents. It integrates trusted security, governance, and observability tools, supports 11,000+ models, and enables integrated development with popular tools such as GitHub, Visual Studio, and Copilot Studio. Azure AI Search, part of Azure AI Foundry, enables developers to ground their AI apps and agents in company data, providing context-aware, relevant search results. Azure AI Foundry also Includes Azure AI Foundry Agent Service, which connects Foundry Models and Azure AI Search with Azure AI services and actions into a single runtime. According to Microsoft, Foundry Agent Service "manages threads, orchestrates tool calls, enforces content safety, and integrates with identity, networking, and observability systems to help ensure agents are secure, scalable, and production ready."[4]

- **Selecting the right AI model and framework for the type of AI application.** As AI has exploded in popularity, the number of models and frameworks available has grown exponentially, making the right choice critical.
- **Confirming the tools, software, and services that the app requires.** AI applications are more than just their models and frameworks; they may require a constellation of other software or tools, some of which are available as cloud-native applications. Though our tested application was not agentic in nature, agentic applications, in particular, require a toolchain to build, orchestrate, scale, and operate them effectively.
- **Right-sizing compute, storage, and networking to provide the performance required for the application.** AI applications generally require low latency and strong compute power, with the ability to easily scale up or down based on demand. Choosing cloud instances and data storage options that are optimized for AI workloads will help. You may also wish to consider fully managed services.
- **Determining the right location and proximity of compute and storage resources to optimize application response time.** By keeping data as close as possible to compute resources, whether CPUs or GPUs, developers and data engineers can ensure acceptable communication latency across the application components.
- **Securing all aspects of the application.** Some organizations must confront the compliance requirements of GDPR, HIPAA, or other regulations, but even those that don't must ensure their AI applications and their data are secure. Encrypting data both at rest and in transit and using appropriate access controls can help. More complex agentic applications expand the security footprint, so continuous observability and governance are required to ensure that the applications and resources they access are secure.

CSPs such as Azure and AWS provide tools and guides to help solve some of these challenges. One such platform is Azure AI Foundry, which provides customers with quick and easy access to the latest OpenAI models, including GPT-5, GPT-4.1, o4-mini, o3-deep-research, and more, making it attractive to users hosting on other CSPs. If you plan to leverage Azure OpenAI for your application, it makes sense to host the rest of your application on Azure, as well, rather than linking from AWS. To evaluate how deploying on Azure can optimize your AI project, we tested a RAG-based AI application on both Azure and AWS to compare ease of deployment, costs, service features, and more. In both of our deployments, we used Azure OpenAI.

Microsoft offers a rich complementary set of tools for developers to easily build and deploy applications across the Azure ecosystem. These tools include:

- **Visual Studio Code:** A powerful free code editor
- **GitHub Copilot:** A generative AI coding assistant with access to multiple AI models
- **Approximately 20 Azure extensions for Visual Studio Code:** Includes extensions for the most popular Azure services such as Azure SQL Databases, Azure Functions, Azure App Service, and others
- **Azure console:** The traditional web UI for managing Azure resources
- **Azure CLI:** A cross-platform-compatible command-line interface for accessing and managing Azure resources
- **Microsoft Copilot Studio:** Used for low-code automation and agentic development
- **Compatibility with third-party tools**: Including Terraform, for infrastructure as code

We experienced the ease of developing applications on Azure in this project, using most of the tools in the list above to accelerate our development time and iterate throughout the project. We specifically used Terraform with the Azure provider to provision most of our infrastructure, the Azure CLI for health checks, Visual Studio Code with GitHub Copilot, and Azure Function tools.

Our AI application

We set out to build a simple RAG AI application that used several of each provider's applicable services. The core services that we included were a web app service, a function service, a search service for the RAG context, a data layer, the LLM, and response tracking. We used a sample public dataset with musical instrument reviews for our dataset, and we used AI to analyze the dataset and generate 280 unique questions for our simulated users to ask of the application. We list the services we used for our deployments on Azure and AWS in Table 1, and Figure 1 shows our web application data flow.

Table 1: List of services we used for our LLM application on Azure and AWS. Source: PT.

Application component	Azure	AWS
Web app	Azure Web App Service	AWS Elastic Beanstalk
Function	Azure Function	AWS Lambda
Search	Azure AI Search	Amazon Kendra
Data	Azure SQL Database	Amazon RDS, SQL Server
LLM service	Azure OpenAI in Azure AI Foundry	
Response tracking	Azure Storage Account (Table)	Amazon DynamoDB

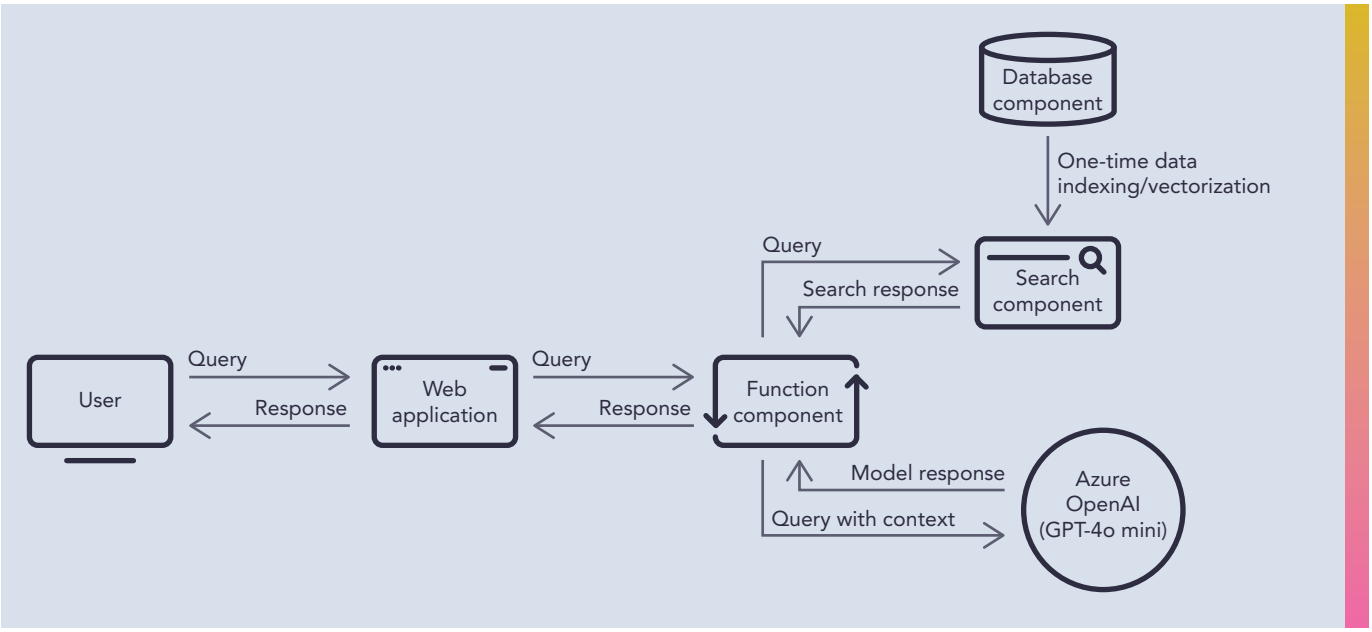


Figure 1: Architecture of our web application data flow. Note that Azure AI Search also uses an embedding model (text-embedding-ada-002) for both inbound queries and one-time vectorization. Source: PT.

The application followed this workflow:

1. A simulated user enters a prompt into the web application.
2. The web application passes the prompt to the function.
3. The function:
   a. Tracks elapsed time of search responses and model responses.
   b. Makes a call to the search service to retrieve relevant context, and returns search results to the function.
   c. Packages the context and the original query into the LLM prompt.
   d. Makes a call to Azure OpenAI containing the original query and augmented context.
   e. Receives the streamed reply to the function
   f. Streams the LLM response and the timing metadata to the web application
4. The web application displays the reply on the screen, and logs the reply and its timings.

We tested three scenarios with different user scales:

- **Single user:** A single user asking 100 questions.
- **100 users:** 100 users, up to 50 concurrent users at a time, each starting 2 seconds apart and asking 70 questions. In this scenario, when the first user finished, user number 51 would start. When the second user finished, user 52 would start, and so on.
- **150 users:** 150 users, up to 100 concurrent users at a time, each starting 2 seconds apart and asking 70 questions. In this scenario, when the first user finished, user number 101 would start. When the second user finished, user 102 would start, and so on.

We tested both Azure and AWS application deployments targeting an Azure OpenAI resource to keep the AI platform constant. We used GPT-4o mini as the model, and we used a Regional Provisioned Throughput deployment with 50 PTUs.

For additional information on our test configurations and scale-point differences per component, see the science behind the report.

Deploying on Azure

Table 2 lists the various components of our application on Azure. We used a combination of Terraform scripts and configuration workflows in the Azure console to deploy and adjust our application. We deployed all components in the same Azure region, US East 2, for consistency and to remove any inter-region latency. We used VS Code for our IDE and used a combination of command line tools and VS Code plugins to build both the web app and the Azure Functions component.

Table 2: The services we used to deploy our LLM application on Azure. Source: PT.

Component	Azure service	Service specifics
Web app	Azure App Service	**Product:** Azure App Service **App service plan pricing tier andinstance:** Premium, P1v3 **Created via:** Terraform Scaling modified in Azure portal
Function	Azure Functions	**Product:** Azure Functions **App service plan pricing tier and instance:** Premium, P1v3 **Created via:** Terraform Scaling modified in Azure portal
Search	Azure AI Search	**Product:** Azure AI Search **Pricing tier:** Standard S1 **Partitions:** 1 **Created via:** Terraform Scaling modified in Azure portal
Data	Azure SQL Database	**Product:** Azure SQL Database **Purchase model:** DTU **Service tier:** S2 (Standard, 50 DTU) **Max storage:** 10GB **Created via:** Terraform
LLM service	Azure OpenAI in Azure AI Foundry	**Model:** GPT-4o-mini **Deployment type:** Regional Provisioned Throughput, 50 PTUs
Embedding model	Azure OpenAI in Azure AI Foundry	**Model:** text-embedding-ada-002 **Deployment type:** Global Standard **Use:** Used in conjunction with Azure AI Search to do embeddings on user prompts
Response tracking	Azure Storage Account (Table)	**Service:** Azure Storage Table

Deploying on AWS

Table 3 lists the various components of our application on AWS. We configured the AWS application to be as similar as possible to the Azure app regarding service levels, quotas, instance types, and application code. We reused the same application code from the Azure function and web app components, changing only the necessary pieces.

We used a combination of Terraform scripts and configuration workflows in the AWS console to deploy and adjust our application. We deployed all components in the same AWS region, US East, for consistency and to remove any inter-region latency. We used VS Code for our IDE and used a combination of command line tools and VS Code plugins to build both the Elastic Beanstalk application and the Lambda Function.

Note that we used Azure OpenAI for our LLM service in both Azure and AWS scenarios. We did this for consistency at the model layer, and to mimic a situation where a business may have a requirement to use OpenAI models.

Table 3: The services used to deploy our LLM application on AWS. Source: PT.

Component	AWS service	Service specifics
Web app	Amazon Elastic Beanstalk	**Product:** Amazon Elastic Beanstalk **Type:** Load balanced **Instance type:** t3.large **Created via:** AWS console Scaling modified in AWS console
Function	AWS Lambda	**Product:** AWS Lambda **Allocated memory:** 256MB **Allocated storage:** 512MB **Created via:** AWS console
Search	Amazon Kendra	**Product:** Amazon Kendra **Edition:** GenAI Enterprise Edition **Storage capacity units:** Default **Created via:** Terraform **Modifications via:** AWS console
Data	Amazon Relational Database Service (RDS) SQL Server	**Product:** Amazon Relational Database Service (RDS) Microsoft SQL Server **Edition:** Express **Instance class:** t3.medium **Allocated storage:** 20GB (minimum for express edition) **Storage type:** gp3 **Created via:** Terraform
LLM service	Azure OpenAI in Azure AI Foundry	**Model:** GPT-4o-mini **Deployment type:** Regional Provisioned Throughput, 50 PTUs
Embedding model	None	Amazon Kendra expects natural language queries, no embedding model required
Response tracking	Amazon DynamoDB	Defaults

Application performance

Why performance matters

One of the challenges in designing and deploying an AI application, whether it is chat-based, an agent, or another AI use case, is ensuring that you size your environment to deliver the performance you need. For a RAG application, the right performance levels could mean the difference between happy or unhappy customers trying to use an online chatbot for help or information. For example, if an AI application that interacts with humans at the end-user stage is slow to respond, users can feel frustrated and may stop using it all together. Customers with an exceptional experience, on the other hand, could lead to increased sales and repeat customers. In addition, performance levels could make a difference in your developers benefiting from AI-assisted coding and their productivity.

There are many ways to tune performance for app components, including compute, storage, and other resources. Keeping applications, models, and data close to each other is vital to keep response times down, so hosting multiple application components in proximity—say in the same geographic region or preferably in the same cloud provider—could help mitigate latency issues. Ultimately, for your users, it comes down to questions such as "How quickly does the application finish what I asked of it?" or "How quickly it is responding?"

Two primary user-facing performance metrics are end-to-end application response time and time between tokens. We measured both metrics in our Azure deployment and AWS deployment. End-to-end application response time, in our case, refers to the length of time between the user submitting the request and the chat returning and completing a response. When measuring end-to-end application response time, lower numbers are better. In our test, time between tokens refers to the rate at which the application stack streamed tokens back to the user. When analyzing time between tokens, lower numbers are better.

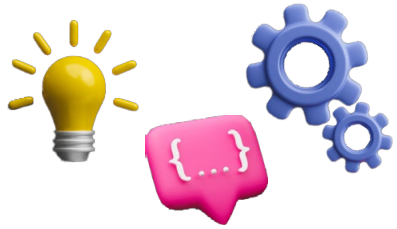
How our application performed

End-to-end application findings

We found that Azure consistently delivered lower end-to-end application response time for our chat application.

For our 1-user, 50-user, and 100-user scenarios, we captured timestamps in our application to measure the following:

- **Total elapsed application time:** The length of time between our web app page load completing and the LLM response completed streaming to the UI
- **Total function time:** The total time spent in the function layer (Azure Function or AWS Lambda), including:
  - **Search time:** The time, from the function's perspective, spent in the call to each provider's search resource (Azure AI Search or Amazon Kendra)
  - **Azure OpenAI time:** The time, from the function's perspective, spent from calling Azure OpenAI to receiving the last token streamed



We ran each scenario three times and selected the median run based on end-to-end elapsed application time. With a single user, the Azure application took 24 percent less time to complete than the AWS application did. At 50 users, the Azure application took 59.7 percent less time and at 100 users, 56.5 percent less time to complete than the application deployed on AWS (see Figure 2).

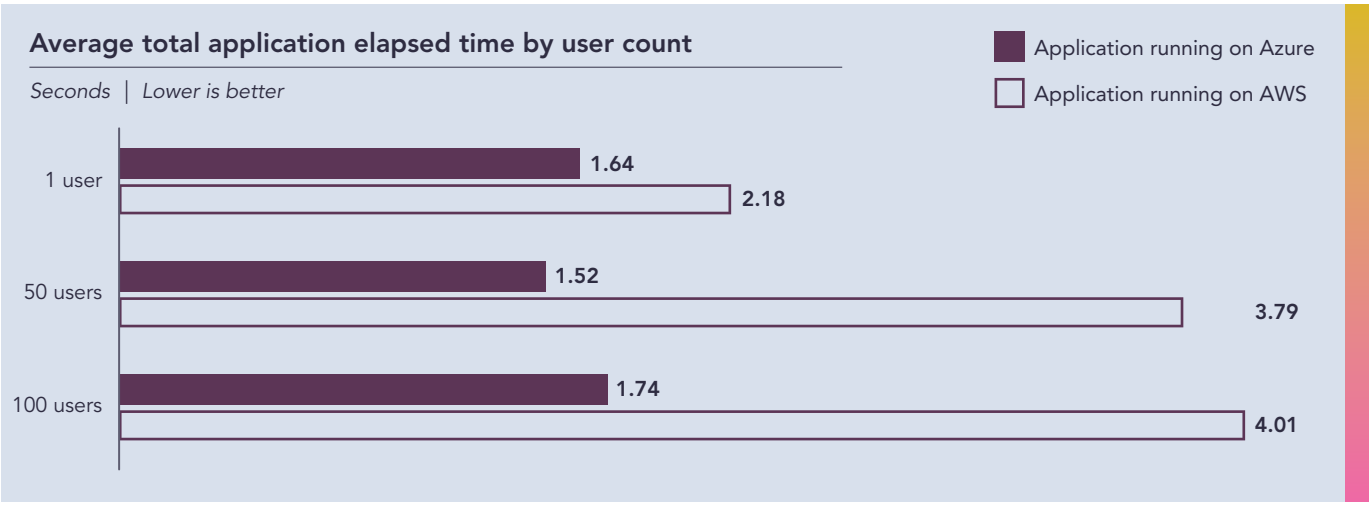


Figure 2: Average application elapsed time for our three user scenarios. Source: PT.

Azure AI Search versus Amazon Kendra

A primary contributor to the lower elapsed time for the Azure app was the fact that Azure AI Search returned context results consistently and significantly faster, especially as we increased user count. Amazon Kendra, by default, allows 10 queries per second (100 query capacity units [QCU]). We attempted to increase query capacity on Amazon Kendra via AWS support but at the time of testing, AWS support informed us that the GenAI Enterprise Edition did not support greater than 100 QCU.

With a single user, Azure AI Search took 53.7 percent less time to complete than Amazon Kendra. With 50 users, Azure AI Search took 88.8 percent less time to complete. At 100 users, Azure AI Search took 82 percent less time to complete than Amazon Kendra (see Figure 3).

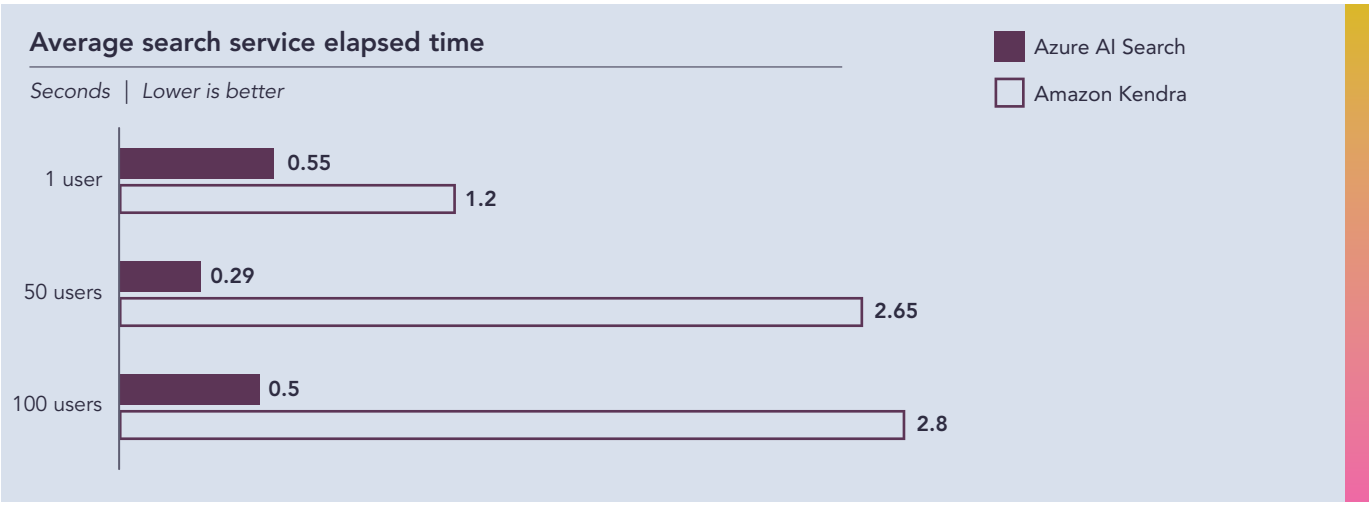


Figure 3: The average time spent in the search layer of the application, in seconds, for our three user count scenarios. Source: PT.

Time between tokens

Any chat application experiences variation in user prompt length, context length and complexity, and response length, and our applications were no different. To evaluate Azure OpenAI performance, we observed time between token performance. We used the same Azure OpenAI resource for both Azure and AWS, demonstrating a scenario where organizations have application components on AWS but wish to use Azure OpenAI.

We used a pool of 280 unique questions contextual to our dataset and allowed each search service to return its top two results, truncated that context string to 600 characters, and then allowed a maximum response token count of 400. Responses generally ranged from 130 tokens to 180 tokens.

In each of our three user count scenarios, we ran the test three times and measured Azure OpenAI time between tokens by dividing the time in the Azure OpenAI call (the time, from the function's perspective, spent from calling Azure OpenAI to receiving the last token streamed) by the number of response tokens. We observed that the Azure application had a better time between tokens rate (see Figure 4). While these sub-millisecond differences are small for our simple application, production AI apps with thousands of users would require many more tokens, compounding the total latency and leading to more noticeable time differences.

For more details on application and scaling adjustments and configurations, please see the science behind the report.

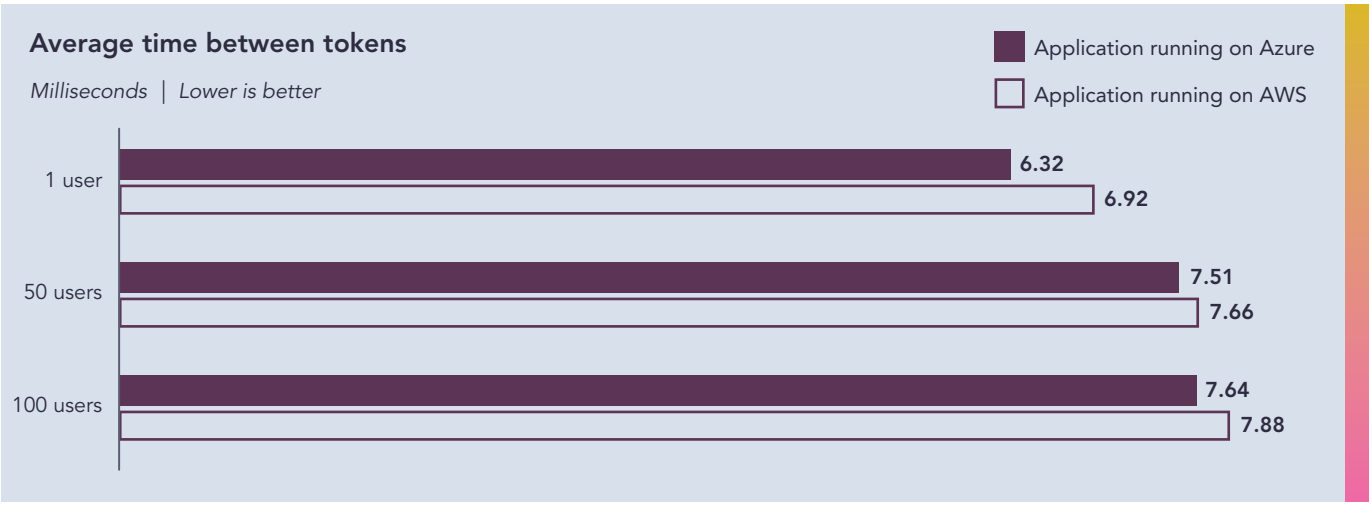


Figure 4: Average time between tokens in our LLM application for our three user scenarios. Source: PT.





The cost impact of multi-cloud applications

Determining the full cost of an AI application deployed on any public cloud will depend on many different factors. CSPs charge for resources beyond just compute and storage costs. If you use the many useful AI services we did in our deployment, you will likely incur additional charges for tokens, number of endpoints, API access, data input/output, and more. These charges are manageable via tools and services from the CSPs. Because these factors vary so widely, we did not conduct a full TCO comparison between Azure and AWS.

Additional costs considerations for deploying AI applications across multiple CSPs can include:

- **Secure networking between the CSPs.** In addition to the different networking costs for each CSP, you may need to pay extra to communicate between them, too. For example, to keep a dedicated, secure interconnect between the two CSPs can cost thousands per month.[5] Using a public network IPsec VPN approach can also accrue data egress costs, though many CSPs are starting to reduce, waive, or remove egress costs.[6] Since Azure OpenAI does not have automatic failover options, you may need to use two regions for high availability, which would mean two of these secure connections between the CSPs.[7]
- **Management overhead.** Using two CSPs means doubling many of the services, tools, and features that need to be learned and managed. Two billing dashboards, two sets of security policies, two IAMs, different APIs...all of these things double the workload of the various administrators and IT employees managing them. This means less time spent on other initiatives and more salary spent on just maintaining your current applications and training to learn two or more platforms.
- **Increased security risks can lead to expensive data breaches.** According to IBM, the average global cost for companies that experienced data breaches in 2024 was $4.88M USD.[8] Deploying across applications with different security standards, tools, and more can increase your risk of having a very costly data breach. In particular, agentic apps are open to wider threat vectors and need protection and control over resources they can access.
- **Complicating optimization means wasted money on underused resources.** When spanning multiple CSPs, visibility into utilization, cost tracking, and more declines. Without robust monitoring or third- party tools to span the gap, it's easy to miss unused capacity or orphaned resources that accrue charges. According to one source, the average cost of cloud management platform tools is $50/mo.[9]

By deploying your Azure OpenAI-dependent app entirely on Azure, you could save money, time, and effort compared to trying to juggle two CSPs at the same time, while also enjoying better performance.

Securing your application

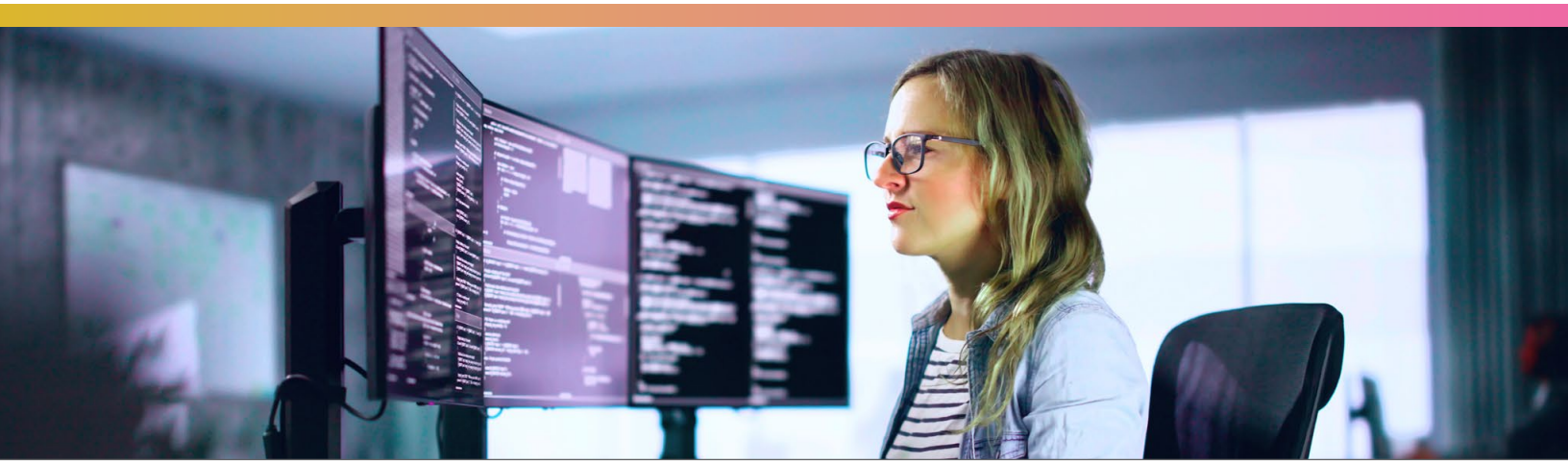
While AI applications deliver benefits, they can also introduce security headaches. In addition to the same security concerns as any other application, the amount and type of data many AI applications use can magnify security flaws. Your data may contain sensitive information, including private user data, business data, government data, or medical data. Whether to protect business and user interests or to conform to data sovereignty and compliance laws, protecting your AI data from breaches is a top priority. You need to ensure that your data is protected at rest, in motion, within the application, in the network, and more.

Though both AWS and Azure provide security features, hosting an application across multiple CSPs can heighten security concerns. Two CSPs means two Identity and Access Management (IAM) systems to configure to limit who has access to sensitive information. It means learning and leveraging two of every security feature relevant to your application and data, making sure that settings are accurate. You may end up with gaps in interoperability or governance and compliance. With two cloud GUIs, you must track and monitor threats across two environments without full visibility from either one. Resolving or avoiding these issues costs time, resources, and money—especially if third-party services to help with visibility or avoiding silos are required. Additionally, Microsoft's security products, including Microsoft Entra ID, Microsoft Defender for Cloud, and Microsoft Purview, integrate with Azure AI Foundry, boosting the security posture of AI applications and agents.[10]

Conclusion

If your organization already hosts application components on AWS, adopting a multi-cloud approach to run AI workloads with Azure OpenAI might make sense. However, this strategy could introduce significant challenges—ranging from managing multi-cloud networking and security to training teams on multiple consoles and APIs. Additionally, stitching together services from multiple cloud providers can result in higher costs, inefficiencies, and increased security risks. Performance could suffer, with your chatbot sluggishly delivering answers that could hurt customer satisfaction.

Adopting a single-cloud strategy with Azure and hosting your RAG AI app on Azure can optimize performance by bringing data closer to compute to give your users answers faster. In fact, running our app on Azure reduced end-to-end execution time by 59.7 percent compared to an AWS deployment. Also, in our tests, Azure provided a faster search service layer for our OpenAI RAG LLM, reducing Azure AI Search time by up to 88.8 percent compared to Amazon Kendra. In application configurations such as ours, the choice is clear: building and hosting your AI app on Azure the better strategy. It reduces complexity—which optimizes performance, saves money, and increases security compared to selecting a multi-cloud deployment. While we used a RAG-based AI app as an example, other more complex agentic AI applications could see similar benefits of a single-cloud strategy.





1. TechRadar, "10 years of Siri: the history of Apple's voice assistant," accessed June 24, 2025, https://www.techradar.com/news/siri-10-year-anniversary.
2. Markovate, "LLM Applications and Use Cases: Impact, Architecture, and More," accessed June 24. 2025, https://markovate.com/blog/llm-applications-and-use-cases/.
3. Markovate, "LLM Applications and Use Cases: Impact, Architecture, and More."
4. Microsoft, "What is Azure AI Foundry Agent Service?" accessed July 10, 2025, https://learn.microsoft.com/en-us/azure/ai-foundry/agents/overview.
5. Cloudverse, "The Hidden Costs of Multicloud Strategies (And How to Avoid Them)," accessed June 24, 2025, https://www.cloudverse.ai/blog/hidden-costs-of-multicloud-strategies/.
6. PacketFabric, "Ways to Connect Multi-cloud: Pros, Cons, and Diagrams," accessed June 24, 2025, https://packetfabric.com/blog/ways-to-connect-multi-cloud-pros-cons-and-diagrams.
7. Microsoft, "Architecture best practices for Azure OpenAI Service – Design Checklist," accessed June 24, 2025, https://learn.microsoft.com/en-us/azure/well-architected/service-guides/azure-openai#design-checklist.
8. IBM, "Cost of a Data Breach Report 2024," accessed June 24, 2024, https://www.ibm.com/reports/data-breach.
9. SaaSworthy, "CloudZero pricing," accessed June 24, 2024, https://www.saasworthy.com/product/cloudzero/pricing.
10. Microsoft, "Enterprise-grade controls for AI apps and agents built with Azure AI Foundry and Copilot Studio," accessed September 5, 2025, https://techcommunity.microsoft.com/blog/microsoft-security-blog/enterprise-grade-controls-for-ai-apps-and-agents-built-with-azure-ai-foundry-and/4414757.

Read the science behind this report ▶

Principled Technologies®

Facts matter.®

Principled Technologies is a registered trademark of Principled Technologies, Inc.
All other product names are the trademarks of their respective owners.
For additional information, review the science behind this report.

This project was commissioned by Microsoft.