A Definitive Guide to Generative AI with Amazon Bedrock

Chapter 7: Overview of Amazon Bedrock Knowledge Base

The ability to augment large language models with external knowledge is crucial for building accurate and contextually-aware AI applications. Amazon Bedrock Knowledge Base provides a seamless solution for implementing retrieval-augmented generation (RAG), allowing models to reference relevant information before generating responses.

This chapter explores how Amazon Bedrock Knowledge Base simplifies the integration of private data sources into generative AI workflows. It covers the end-to-end process of data ingestion, chunking strategies for efficient retrieval, semantic search capabilities, and augmenting model prompts with retrieved information. The knowledge base securely connects to various data sources like Amazon S3, Salesforce, and SharePoint, automatically indexing the content.

Key features like multi-turn conversation support, customized retrieval, source attribution, and cost-effectiveness are highlighted. The chapter also dives into critical aspects such as data security, access control, monitoring, and compliance adherence offered by the service.

Additionally, it provides insights into selecting the right vector database for storing and searching high-dimensional embeddings, a core component of RAG systems. By the end of this chapter, readers will understand how Amazon Bedrock Knowledge Base accelerates the development of generative AI applications enriched with proprietary data.

7.1 Introduction to Amazon Bedrock Knowledge Base

In the last chapter, you learned about Retrieval Augmented Generation (RAG). RAG is a design pattern to improve the output of a large language model by referencing an outside knowledge base before generating a response. RAG makes sure that the model's answers are more accurate and based on more recent, reliable data than just the data it used for training. You will learn how AWS demystify the implementation of RAG. Even, smooth integration with other features of Amazon Bedrock.

Amazon Bedrock Knowledge Bases allow you to fully leverage RAG by providing a simple way to access external data and enrich Large Language Model (LLM) outputs. Application can query this resource connecting your data sources to knowledge base. Then, it gets the right information to add to the context and answer the user's question through direct quotes or natural language. This approach allows you to build applications enriched by the context provided through the knowledge base, speeding up your time to market. By eliminating the need to manually build data pipelines, Amazon Bedrock delivers an out-of-the-box RAG solution, making application development faster and more efficient. Additionally, integrating a knowledge base reduces costs, as there’s no need for continual retraining of the model to incorporate your private data.

**Data Preparation & Organization**

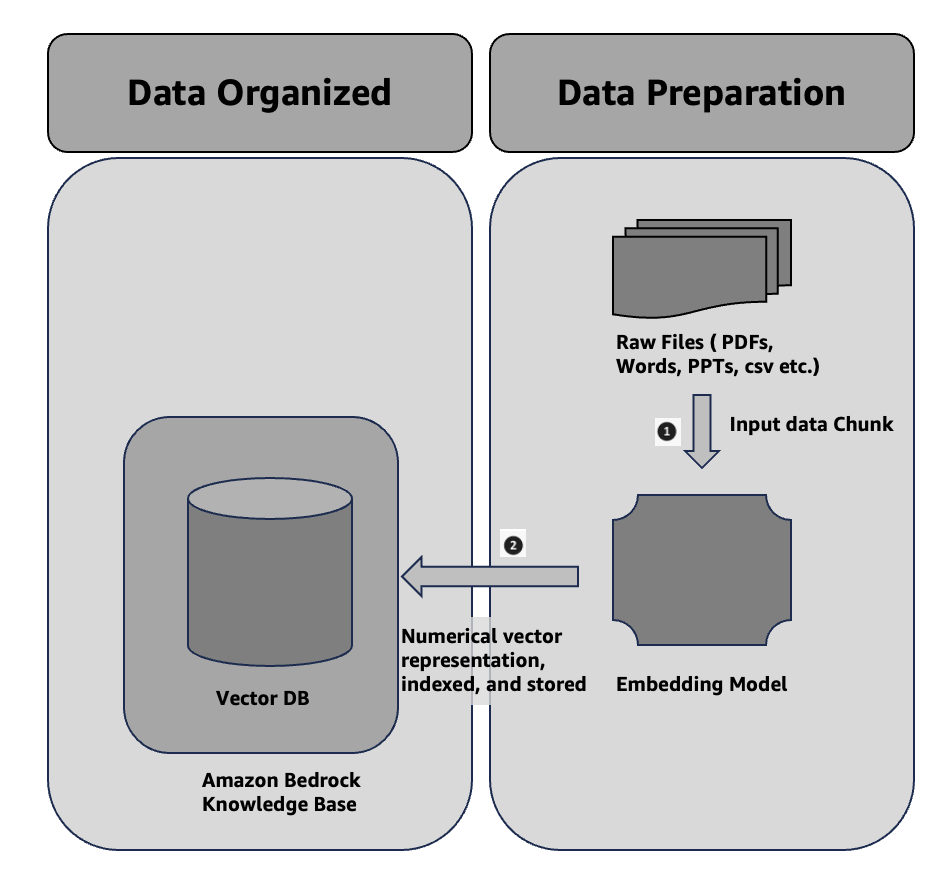


Figure 7.1 Data Preparation & Organization

Even so This is the same as bullet points 1 and 2 in Section 6.4 of the last chapter. To make it easier to get private data, one common design is to first split the documents into chunks for efficient retrieval. Next, you embed the chunks into embeddings and transfer them to a vector index, which maintains their mapping to the original text. You use these embeddings to determine the semantic relationships between queries and text from the data sources. Figure 7.1 shows how data is prepared for the vector database. Instead of the Vector DB you used and managed on your own, Amazon Bedrock Knowledge Base now offers a variety of options for Vector DB.

**Experiences and Information Retrieval & Generation**

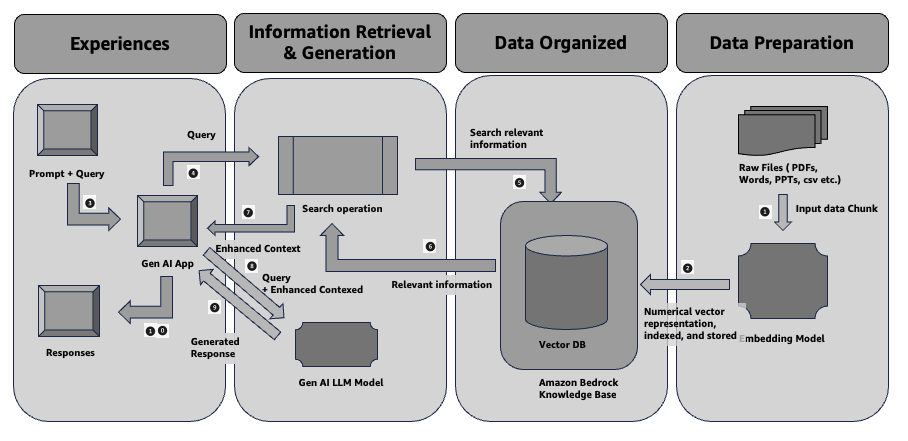
****

Figure 7.2 Experiences and Information Retrieval & Generation

Even so This is the same as bullet points 5–10 in Section 6.4 of the last chapter. You use an embedding model to transform the user's query (prompt) into a vector. You then query the vector index, comparing document vectors to the user query vector, to identify chunks that share semantic similarities with the user's query. The last step improves the user prompt by incorporating additional information from the chunks retrieved from the vector index. We then pass the prompt and additional information to the model, allowing it to generate a response for the user. Figure 7.2 below illustrates how RAG works in runtime to improve responses to user questions. Additionally, in the sections below, you will learn some specific APIs to communicate with the Amazon Bedrock Knowledge Base.

7.2 Why Amazon Bedrock Knowledge Base

In this section, you will discover the significance of Amazon Bedrock Knowledge in the development of your generative AI application.   
  
**Seamless RAG Workflow**: Fully managed support for retrieval augmented generation (RAG), eliminating the need for custom integrations and manual data handling.  
  
**Contextual AI with Proprietary Data**: Enables foundation models (FMs) and agents to access your company’s private data, delivering more relevant, accurate, and customized responses. You will learn agent in the next chapter.  
  
**Secure Data Connectivity**: Securely connects to data sources like Amazon S3, Salesforce, Confluence, and SharePoint, automatically ingesting and indexing content.  
  
**Flexible Data Ingestion**: It supports a number of different ingestion methods, such as handling complex unstructured data (PDFs, images), and you can change the chunking settings to make it easier to find data.  
  
**Supports multi-turn conversations**: Built-in session context management allows your app to handle ongoing conversations, maintaining coherence across interactions.  
  
**Customized Retrieval**: You can improve the accuracy of retrieval by enhanced queries and using advanced processing to make them work best for your business.  
  
**No External Vector Database Needed**: This solution provides a managed vector store (like Amazon OpenSearch Serverless and others ) or the flexibility to connect to your existing vector databases, such as Pinecone or Redis.

**Augmented prompts**: They automatically enrich user queries with relevant, up-to-date information to improve response quality.  
  
**Source Attribution**: It provides citations for retrieved data, ensuring transparency and minimizing AI hallucinations.  
  
**Cost-effective**: Reduces the need for constant retraining of models by dynamically augmenting models with real-time, proprietary data.  
  
**Quick Time-to-Market**: Abstracts the complexity of building pipelines, offering an out-of-the-box RAG solution that accelerates AI application development.

7.3 Sample Applications of Amazon Bedrock Knowledge Base

( Introduction about the chapter )

7.4 Overview of Chunking Strategy

Chunking is a key part of retrieval-augmented generation (RAG). It splits the data into smaller, easier-to-handle "chunks" to make retrieval faster and more accurate. Amazon Bedrock Knowledge Base supports different chunking strategy. You will learn in this section in detail along with their advantages, drawbacks, and use cases in this section.

**Fixed-Size Chunking**

Entire data splits into chunks of a predetermined size such as 500 or 1000 characters or tokens in this approach. Each chunk is treated as a separate unit for embedding and retrieval.

|  |  |  |
| --- | --- | --- |
| Advantages | Drawback | Use cases |
| **Simplicity**: Easy to implement and comprehend. The chunks are uniformly sized. It makes retrieval straightforward.  **Efficient for Structured Data**: Works well when dealing with uniform, structured data, like logs, technical manuals, or data tables. | **Lack of Context**: Fixed-size chunks can cut off meaningful content in the middle of sentences or paragraphs, leading to potential loss of context.  **Reduced Relevance**: The retrieval process may not provide the full picture if pertinent information is dispersed across multiple segments. | This approach is ideal for data with uniformly structured content, like user manuals, FAQs, and log files. In such cases, losing a bit of context between chunks has minimal impact. |

**No Chunking**

This method treats the entire document as a single chunk and indexes it for retrieval.

|  |  |  |
| --- | --- | --- |
| Advantages | Drawback | Use cases |
| **Preserves Full Context:** Processing the document in its entirety eliminates the risk of losing crucial information that could arise from segmenting it into smaller parts.  **Ideal for Short Texts:** Works well with smaller documents, where dividing the content could dilute its meaning or context. | **Inefficient for Large Documents:** Processing or retrieving large documents in this manner can be computationally expensive and hindered by the limited context window of generative AI models with slower retrieval. | This approach highly effective for obtaining summaries from separate documents. It demonstrates significant efficacy for concise documents, including legal contracts. It maintains the complete context for accurate replies. |

**Hierarchical Chunking**

This method systematically arranges segments within a hierarchical framework. The retrieval process considers both specific details and the overarching context by organising smaller components within larger frameworks.

|  |  |  |
| --- | --- | --- |
| Advantages | Drawback | Use cases |
| **Preserves Structure**: The hierarchy ensures the document's logical flow while splitting it for effective retrieval.  **Context-Aware**: The hierarchy allows for retrieval based on both high-level overviews and detailed sections. | **Complex to Implement**: It is more difficult to establish and oversee, necessitating more advanced indexing and retrieval procedures.  **Computational overhead**: Managing hierarchical relationships can increase the system’s computational requirements. | It works best for intricate technical documents, books, or research papers where it's crucial to maintain both a detailed and high-level context. |

**Semantic Chunking**

Semantic chunking divides the content based on meaning, not fixed sizes. The system uses natural language understanding to create chunks that capture complete ideas or sections, such as paragraphs or sections with similar themes.

|  |  |  |
| --- | --- | --- |
| Advantages | Drawback | Use cases |
| **High Relevance**: Each chunk captures a complete thought, ensuring the retrieval of meaningful sections.  **Context Retained**: Semantic chunking preserves the natural flow of information, making it ideal for conversational AI or customer support tools. | **Complex Parsing:** requires advanced natural language processing (NLP) techniques to identify appropriate chunk boundaries.  **Slower Processing:** The system may take more time to parse and chunk documents accurately, leading to slower indexing times. | This feature is particularly useful for customer service databases, knowledge articles, and documents that have complex or varied structures. It ensures that retrieved information is both accurate and contextually relevant. |

**Custom Chunking**

Custom chunking allows you to write your own chunking logic, often using tools like AWS Lambda functions or frameworks like LangChain and LlamaIndex. This allows you complete control over the data splitting process, providing the flexibility to optimize for specific use cases.

|  |  |  |
| --- | --- | --- |
| Advantages | Drawback | Use cases |
| **Highly Tailored**: You can modify chunking to meet specific business requirements, guaranteeing the most effective division of data.  **Flexibility**: Allows the use of advanced techniques and third-party frameworks to achieve optimal chunking for different types of data. | **Requires Expertise**: Custom chunking involves more development work and requires knowledge of how to optimize chunking for retrieval performance.  **Increased Maintenance**: As data structures or use cases change, custom implementations require maintenance and updates. | This approach is best suited for specialized industries like healthcare or finance, where data may have unique formats (e.g., medical records or financial reports), requiring highly specific chunking strategies. |

**Choosing the Right Strategy**

Each chunking strategy has its strengths and weaknesses, and the best choice depends on the type of data you are working with and your desired outcomes. For straightforward, structured data, fixed-size chunking is often sufficient. For more complex documents that require a higher level of contextual awareness, semantic or hierarchical chunking is ideal. Custom chunking is reserved for use cases where a standard approach won’t suffice, and you need complete control over how data is processed.

By understanding the advances and drawbacks of each strategy, organizations can choose the most appropriate chunking method to optimize their Amazon Bedrock Knowledge Bases for efficient and accurate information retrieval.

7.5 Governance & Monitoring

Amazon Bedrock Knowledge Bases offer robust governance and monitoring capabilities to ensure data integrity, security, compliance, and operational efficiency. It manages governance and monitoring in the following way:

**Data Security & Access Control**: Amazon Bedrock Knowledge Bases integrate with AWS Identity and Access Management (IAM), allowing administrators to define detailed permissions for who can access, manage, or retrieve data from the knowledge base. This ensures that only authorized systems and users can access proprietary information, thereby safeguarding sensitive data. (https://docs.aws.amazon.com/bedrock/latest/userguide/security-iam.html)

**Audit Trails**: Amazon Bedrock Knowledge Bases logs all interactions, including data ingestion, retrievals, and modifications. These audit logs help track user activities, providing visibility into who accessed or altered the knowledge base and when. Compliance with industry regulations and internal security policies requires this level of traceability. (https://docs.aws.amazon.com/bedrock/latest/userguide/logging-using-cloudtrail.html)

**Monitoring & Alerts**: AWS CloudWatch provides real-time monitoring of system performance and data flow within Bedrock Knowledge Bases. You can set up custom alerts to notify administrators about any unusual activity, system errors, or performance bottlenecks. Proactive monitoring guarantees the prompt resolution of potential issues. It reduces downtime. (https://docs.aws.amazon.com/bedrock/latest/userguide/knowledge-bases-logging.html)

**Failure Handling**: You can implement automated retries and error recovery mechanisms to effectively manage failures. If there’s an issue during ingestion or retrieval, APIs will attempt to resolve it without disrupting the overall workflow.

**Encryption**: Amazon Bedrock Knowledge Bases uses AWS-managed encryption services to encrypt data stored and transmitted. It supports encryption of data at rest and in transit. (https://docs.aws.amazon.com/bedrock/latest/userguide/encryption-kb.html)

**Compliance**: Amazon Bedrock Knowledge Bases adhere to a variety of compliance standards, such as GDPR, HIPAA, and SOC. This makes it suitable for industries that need to adhere to stringent data protection regulations. (https://docs.aws.amazon.com/bedrock/latest/userguide/compliance-validation.html)

By combining these governance and monitoring features, Amazon Bedrock Knowledge Bases provide organizations with full control, flexibility, and visibility over their data, ensuring secure, compliant, and reliable operations.

7.6 Design principals of right Vector DB

The right choice of vector database is very important during building generative AI applications powered by generative AI models. The database needs to efficiently store and retrieve high-dimensional vector embeddings that represent the semantic meaning of text data, image, video, audio etc. The fundamental design guidelines that will direct the creation or selection of such a vector database are listed below.

Semantic Search Capabilities

The core function of the vector database is to perform semantic search - retrieving the most relevant text chunks or documents based on their vector embeddings' proximity to the query vector in high-dimensional space. This relevant text chunks helps to create enhanced context as an input for generative AI model to generate more accurate and precise outputs.

Scalability for Vector Datasets

Generative AI applications often engage with large datasets including billions of vector embeddings. The database must effectively support continuous ingestion, rebuilding index and search large vector datasets while ensuring high performance and resilience.

High Dimensionality Support

A large number of modern embedding models generate high-dimensional vectors (1024, for example). The database need to enable efficient, large-scale ingesting and searching on these high-dimensional vectors.

Optimized Indexing Techniques

To enable fast nearest neighbor search in high-dimensional spaces, the database should implement advanced indexing algorithms like Hierarchical Navigable Small World (HNSW) or Inverted File with Flat compression (IVFFlat). These methods reduce latency and performance.

Configurable Relevance and Recall

Relevance and recall are critical for generative AI applications. Users should be able to configure the desired trade-off between these factors in the vector database, taking into account their specific needs and preferences and ensures the retrieved results are sufficiently relevant and complete.

Hybrid search and filtering

The vector database should support combining it with traditional search like keyword matching, phrase matching, full-text search and structured filtering in addition to vector similarity search. This enables more refined and targeted retrieval of relevant information.

Strong Integration with ML/LLM Frameworks

To enable generative AI applications to simplify development and deployment, the vector database should effortlessly interface with popular machine learning and big language model frameworks and platforms.

Serverless and Fully Managed

As generative AI workloads can be bursty and difficult to predict. A serverless and fully managed vector database service can be advantageous. It will abstracting away operational complexities of automatically scaling resources up or down based on demand.

Some of the vector database offering from AWS which could be consider during design.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Properties | Memory | Document | Graph | Search | RDBMS |
|  | Amazon MemoryDB | Amazon DocumentDB | Amazon Neptune | Amazon OpenSearch | Amazon Aurora/ RDS with pgvector |
| Index | HNSW, FLAT | IVFFLAT, HNSW | HNSW | IVFFLAT, HNSW | IVFFLAT, HNSW |
| Max Dimensionality | 32768 | 16k, 2k index | 65535 |  | 16k, 2k index |
| Max Vectors | Millions | Billions | Billions | Billions | Billions |
| Serverless | No | No | No | Yes | Yes ( Aurora) |
| Full Text Search | No | No | No | Yes | Yes |
| Hybrid Search | No | No | No | Yes | No |
| Quantization | No | No | No | PQ, SQ | SQ |

By following these design rules, a vector database can become the best way to store and retrieve information for adding relevant external knowledge to generative AI models. This opens up new possibilities for smart and aware applications in many areas.. This enables more refined and targeted retrieval of relevant information.

7.7 Summary

The chapter discusses the importance of Amazon Bedrock Knowledge Base in developing generative AI applications. It explains how Amazon Bedrock Knowledge Base simplifies the process of retrieval-augmented generation (RAG) by providing a managed solution for accessing external data and enriching Large Language Model (LLM) outputs. The chapter also highlights the benefits of using Amazon Bedrock Knowledge Base, such as seamless RAG workflow, contextual AI with proprietary data, secure data connectivity, flexible data ingestion, support for multi-turn conversations, customized retrieval, and no external vector database needed. Additionally, the chapter provides an overview of chunking strategy, which is crucial for efficient retrieval of information from large datasets. The chapter discusses different chunking strategies, such as fixed-size chunking, no chunking, hierarchical chunking, semantic chunking, and custom chunking, along with their advantages, drawbacks, and use cases. It also explains how to choose the right chunking strategy based on the type of data being worked with. The chapter also covers governance and monitoring of Amazon Bedrock Knowledge Base, including data security and access control, audit trails, monitoring and alerts, failure handling, encryption, and compliance. Finally, the chapter discusses the design principles of a right vector database, which is essential for storing and retrieving high-dimensional vector embeddings efficiently. It highlights the importance of semantic search capabilities, scalability for vector datasets, high dimensionality support, optimized indexing techniques, configurable relevance and recall, hybrid search and filtering, and strong integration with ML/LLM frameworks. The chapter also mentions some of the vector database offerings from AWS, such as Amazon MemoryDB, Amazon DocumentDB, Amazon Neptune, Amazon OpenSearch, and Amazon Aurora/RDS with pgvector.