Technology Stack Analysis for an Intelligent Microsoft Teams Planner Management System

DATE: 2025-10-06

REPORT OBJECTIVE: Provide comprehensive analysis and recommendations for technology stack components (databases, RAG frameworks, document processing, web crawling, document generation) for building an intelligent Microsoft Teams Planner management system. The report should include detailed comparisons, feature matrices, integration patterns, Docker deployment considerations, and actionable recommendations for technical developers building a containerized, single-user RAG-enabled system.

Executive Summary

This report presents a detailed analysis and strategic recommendations for the technology stack required to build a sophisticated, containerized, and RAG-enabled management system for Microsoft Teams Planner. The objective is to provide technical developers with a clear, evidence-based roadmap for selecting components that ensure scalability, flexibility, and high performance for a single-user environment. The analysis covers five critical domains: databases, Retrieval-Augmented Generation (RAG) frameworks, document processing, web crawling, and document generation.

For the database layer, this report recommends a hybrid model. **PostgreSQL**, augmented with the **pgvector** extension, is advised as the primary relational and vector database. Its robust support for complex queries, advanced data types like JSONB, and strong performance in mixed workloads make it superior to MySQL for this application. For modeling complex, interconnected data, such as task dependencies or organizational relationships that extend beyond Planner's native capabilities, **Neo4j** is recommended as a specialized graph database. This dual-database approach provides both structured data integrity and the ability to perform deep relational analysis.

In the domain of RAG frameworks, **LangChain** is identified as the optimal choice for orchestration. Its modularity and extensive integration ecosystem provide the necessary flexibility to chain together various components, from data ingestion to LLM interaction. While LlamaIndex offers superior performance in pure retrieval tasks, LangChain's strength in building complex, agentic workflows is better suited to the multifaceted nature of a Planner management system.

For document ingestion and processing, **Unstructured.io** is recommended over alternatives like Apache Tika and IBM's Docling. Its focus on transforming complex, unstructured documents into formats specifically optimized for LLM and RAG pipelines provides a significant advantage. It excels at preserving document structure, such as tables and lists, which is critical for generating high-quality embeddings.

To acquire external data for augmenting Planner tasks, **Playwright** is the recommended web crawling tool. Its browser automation capabilities are essential for reliably scraping modern, JavaScript-heavy websites, a common weakness of simpler libraries like BeautifulSoup. For developers seeking to minimize maintenance and leverage AI, **Firecrawl** presents a compelling, low-code alternative.

Finally, for generating reports and documents from Planner data, a multi-tool approach is advised. **WeasyPrint** is recommended for generating high-fidelity PDF documents from HTML and CSS tem-

plates, offering a familiar workflow for web developers. For native Microsoft Office formats, the combination of **python-docx** and **python-pptx** provides direct, programmatic control over Word and PowerPoint file generation.

The report concludes with detailed integration patterns, advocating for a microservices-inspired architecture deployed via **Docker Compose**. This approach containerizes each component—database, RAG API, document processor—ensuring modularity, portability, and simplified dependency management, which is ideal for a single-user, developer-focused system.

Database Technology Analysis

The selection of a database architecture is a foundational decision that will profoundly impact the system's ability to store, retrieve, and analyze data related to Planner tasks. The ideal solution must handle structured relational data, efficiently store and query high-dimensional vector embeddings for RAG, and potentially model complex relationships between tasks, users, and projects. This analysis evaluates relational, vector, and graph database options to recommend a comprehensive data layer.

Relational Database: PostgreSQL vs. MySQL

The choice between PostgreSQL and MySQL, the two leading open-source relational database management systems (RDBMS), is critical for the core data persistence layer. While both offer ACID compliance and robust SQL support, their feature sets, performance characteristics, and recent development trajectories diverge in ways that are significant for this project. Based on recent trends and technical capabilities, PostgreSQL emerges as the superior choice for an intelligent Planner management system.

PostgreSQL has demonstrated significant momentum in recent years, surpassing MySQL in developer preference according to the 2023 Stack Overflow Developer Survey and winning the DB-Engines DBMS of the Year award in 2023. This growing adoption is driven by its extensibility and advanced feature set, which are particularly relevant for AI and analytics workloads. PostgreSQL supports a richer variety of data types out of the box, including arrays, JSONB (a binary JSON format), and geometric types. The JSONB type is especially valuable for this project, as it allows for the efficient storage and indexing of semi-structured data that might accompany Planner tasks, such as custom metadata or logs from external systems. In contrast, MySQL's JSON support, while functional, lacks the advanced indexing capabilities of PostgreSQL's JSONB, making complex queries on this data less performant.

From an architectural standpoint, PostgreSQL's object hierarchy, which includes schemas within databases, offers greater organizational flexibility than MySQL's flatter structure. This is beneficial for a system that might need to logically separate data for different functions or tenants in the future. Furthermore, PostgreSQL provides more robust support for advanced SQL features like Common Table Expressions (CTEs) and window functions, enabling more sophisticated data analysis directly within the database. Its query optimizer is generally considered more advanced, leading to better performance on complex, multi-join queries that are likely to arise when analyzing Planner data.

While MySQL has historically been favored for its simplicity and high performance in read-heavy web applications, PostgreSQL has largely closed the performance gap and often excels in mixed read-write workloads and scenarios involving large datasets. PostgreSQL's implementation of Multi-Version Concurrency Control (MVCC) is highly effective at managing concurrent operations without excessive locking. For the specific needs of a RAG-enabled system, PostgreSQL's extensibility is a decisive advantage. Extensions like **pgvector** allow it to function as a capable vector database, co-locating relational data and vector embeddings within the same system, which simplifies the architecture and reduces operational overhead. While MySQL is a reliable and performant database, PostgreSQL's superior feature set, extensibility, and strong trajectory in AI-related applications make it the more strategic and powerful choice for this project.

Vector Database: Qdrant vs. pgvector

An essential component of the RAG system is the vector database, which stores numerical representations (embeddings) of documents and enables efficient similarity searches. The choice lies between a dedicated, high-performance vector database like Qdrant and an integrated solution like the pgvector extension for PostgreSQL. For a containerized, single-user system, the simplicity and efficiency of an integrated approach make **pgvector** the recommended solution.

Qdrant is a powerful, open-source vector database written in Rust, engineered for high performance, scalability, and advanced features. It offers sophisticated capabilities such as advanced filtering with payloads, hybrid search combining dense and sparse vectors, and vector quantization to reduce memory usage and accelerate search speeds. Benchmarks often show Qdrant outperforming competitors in requests per second (RPS) and achieving low latency at high recall rates. Its architecture is designed for distributed deployments, with support for sharding and replication, making it an excellent choice for large-scale, enterprise-grade AI applications. However, for a single-user system, deploying and managing a separate, dedicated database service introduces significant architectural and operational complexity. It requires separate data pipelines, synchronization logic between the relational and vector stores, and additional container management.

In contrast, **pgvector** is an open-source extension for PostgreSQL that adds vector similarity search capabilities directly to the relational database. It allows developers to create a `vector` data type column within standard SQL tables, store embeddings alongside their corresponding metadata, and perform nearest-neighbor searches using SQL queries. pgvector supports exact and approximate nearest neighbor (ANN) searches with popular indexing methods like IVFFlat and HNSW, and it handles various distance metrics such as cosine distance and L2 distance. The primary advantage of this approach is its profound simplicity. By integrating vector storage and search into the primary database, the system avoids the need for a separate database service. This simplifies the Docker Compose configuration, reduces memory overhead, and eliminates the complexities of data synchronization. Developers can perform powerful hybrid queries that combine traditional SQL filtering (e.g., on task due dates or assignees) with semantic vector search in a single, atomic transaction. While pgvector may not match the raw throughput of a dedicated database like Qdrant in highly concurrent, large-scale scenarios, its performance is more than sufficient for a single-user application and its architectural benefits are overwhelming. The ability to leverage PostgreSQL's mature ecosystem for backups, security, and administration further solidifies its position as the most practical and efficient choice for this project.

Graph Database: Neo4j and Its Alternatives

While a relational database can model many aspects of Planner data, a graph database excels at managing and querying highly interconnected data with complex relationships. For an intelligent Planner management system, a graph database like **Neo4j** could provide advanced capabilities, such as visualizing task dependencies, analyzing team collaboration patterns, or mapping project workflows. While not a mandatory component for a baseline system, its inclusion should be considered for future enhancements.

Neo4j is a leading native graph database that uses a property graph model, where data is stored as nodes, relationships, and key-value properties. Its core strength is its Cypher query language, a declarative, SQL-like language designed for intuitive and efficient graph traversal. Neo4j is ACID-compliant and offers robust features for graph analytics and machine learning through its Graph Data Science (GDS) library. For this project, Neo4j could model Planner tasks as nodes, with relationships representing dependencies (`DEPENDS_ON`), assignments (`ASSIGNED_TO`), and containment (`PART_OF`). This would enable powerful queries that are difficult and inefficient to execute in a relational database, such as finding the critical path in a project or identifying individuals who are bottlenecks across multiple plans. Deploying Neo4j is straightforward with Docker, using official images that support volume mounting for data persistence and environment variables for configuration, making it a good fit for the proposed containerized architecture.

However, Neo4j is not without alternatives. **Memgraph** is a strong competitor, offering an in-memory, C++ based architecture that delivers significantly higher performance in both read- and write-heavy workloads. It is compatible with the Cypher query language, making it a potential drop-in replacement for performance-critical applications. **ArangoDB** is a multi-model database that supports graph, document, and key-value models, offering flexibility for applications with diverse data needs. Its unified query language (AQL) can perform complex joins across different data models. For a system that needs to scale horizontally to handle massive datasets, distributed graph databases like **Dgraph** or **NebulaGraph** offer superior performance and scalability compared to Neo4j's standard architecture.

For the initial build of a single-user Planner management system, a dedicated graph database may be an unnecessary complexity. The combination of PostgreSQL and pgvector can handle the primary requirements. However, if the system's roadmap includes advanced analytical features focused on relationships and dependencies, **Neo4j** is the recommended choice due to its maturity, extensive documentation, strong community support, and ease of deployment with Docker. Its capabilities would provide a powerful extension to the system's analytical core without requiring a complete architectural overhaul.

RAG Framework and Orchestration

The core intelligence of the system will be driven by a Retrieval-Augmented Generation (RAG) pipeline, which requires a framework to orchestrate data ingestion, embedding, retrieval, and interaction with a Large Language Model (LLM). This section evaluates leading frameworks to determine the best fit for building a flexible and powerful conversational agent for Planner.

RAG Frameworks: LangChain vs. LlamaIndex vs. Haystack

The choice of a RAG framework is pivotal, as it provides the scaffolding for the entire AI workflow. The three most prominent open-source frameworks are LangChain, LlamaIndex, and Haystack, each with a distinct architectural philosophy and set of strengths. For building a versatile and potentially complex Planner management agent, **LangChain** is the recommended framework due to its unparalleled flexibility and focus on orchestration.

LlamaIndex is purpose-built for optimizing the "retrieval" component of RAG. It excels at data ingestion, indexing, and querying from a wide variety of data sources. Its architecture is highly optimized for low-latency, high-throughput retrieval, and benchmarks often show it to be the fastest framework for pure search and Q&A tasks. It provides advanced indexing strategies and a high-level API that simplifies the process of building a robust RAG pipeline. However, its primary focus is on the data layer, and it is less comprehensive when it comes to building complex, multi-step agentic workflows that involve chaining multiple tools and LLM calls.

Haystack is an enterprise-focused framework designed for building production-ready, scalable search systems. It features a modular, pipeline-centric architecture composed of components like retrievers, readers, and rankers. This design makes it highly scalable and suitable for end-to-end NLP applications, particularly in enterprise environments where accuracy and robustness are paramount. However, its comprehensive nature can lead to a steeper learning curve and higher resource requirements, which may be overkill for a single-user system.

LangChain, in contrast, is best described as a general-purpose orchestration framework for LLM applications. Its core strength lies in its concept of “chains” and “agents,” which allow developers to flexibly compose sequences of operations involving LLMs, external tools, and data sources. While it also provides robust RAG capabilities, its true power is in its modularity and extensive ecosystem of integrations. For a Planner management system, this is a significant advantage. A user’s request might require multiple steps: retrieving existing tasks (a RAG operation), calling the Graph API to create a new task (a tool-use operation), and then generating a summary report (an LLM chain operation). LangChain’s architecture is explicitly designed to handle such complex, multi-step workflows. While it may have slightly higher latency in simple retrieval tasks compared to LlamaIndex, its ability to build sophisticated, tool-using agents makes it the most suitable choice for a system that goes beyond simple Q&A to actively manage and manipulate Planner data.

Advanced RAG with Knowledge Graphs: The Role of Graphiti

While traditional RAG relies on semantic search over text chunks, an emerging and more powerful approach involves structuring knowledge into a graph. **GraphRAG** enhances retrieval by leveraging the explicit relationships between entities, enabling more complex, multi-hop reasoning. **Graphiti** is a notable open-source framework designed for building real-time, temporal knowledge graphs specifically to enhance RAG systems.

Graphiti addresses a key limitation of standard RAG: handling dynamic and time-sensitive data. It constructs a knowledge graph from incoming data streams and supports incremental updates without requiring a full re-computation of the graph. This is highly relevant for a Planner system, where task statuses, due dates, and assignments change frequently. Graphiti features bi-temporal modeling, tracking both when an event occurred and when it was ingested, allowing for precise historical queries. Its hybrid retrieval mechanism combines semantic search, keyword search, and graph traversal to provide context-aware results without relying on an LLM during the retrieval step. While implementing a full GraphRAG system with Graphiti represents a significant increase in complexity compared to a standard vector-based RAG pipeline, it offers a clear path for future enhancement. For an advanced version of the Planner management system, Graphiti could be used to build a dynamic knowledge graph of all Planner activity, enabling the AI to answer complex temporal and relational questions like “What tasks were moved from ‘In Progress’ to ‘Completed’ last week?” or “Show me the chain of dependencies for the ‘Q4 Launch’ task.” For the initial implementation, a standard RAG approach with LangChain is sufficient, but Graphiti should be considered a key technology for future roadmap development.

Document Ingestion and Processing

To enable RAG, the system must effectively process and ingest documents from various sources to create a searchable knowledge base. This requires a robust document processing toolkit that can parse different file formats, extract text and metadata, and structure the content in a way that is optimal for embedding and retrieval.

Document Processing Tools: Unstructured.io vs. Apache Tika vs. Docling

The selection of a document processing tool is crucial for the quality of the RAG system’s knowledge base. The primary contenders in the open-source space are Unstructured.io, the long-standing Apache

Tika, and the more recent IBM-developed Docling. For building an AI-native system, **Unstructured.io** is the recommended choice due to its specific focus on preparing documents for LLM workflows.

Apache Tika is a mature and highly versatile content analysis toolkit from the Apache Software Foundation. Its greatest strength is its ability to detect and extract text and metadata from over a thousand different file formats. It provides a single, unified interface for parsing, making it an excellent general-purpose tool for data extraction, particularly for tasks like search engine indexing. However, Tika's primary function is to extract raw text. It does not inherently attempt to understand or preserve the document's high-level logical structure, such as distinguishing between titles, paragraphs, tables, and lists. This can result in a less coherent input for the embedding model, potentially reducing retrieval accuracy.

Docling is an open-source toolkit from IBM Research, designed for efficient, AI-driven document conversion. It supports a wide range of formats and uses advanced models like DocLayNet for layout analysis and TableFormer for table recognition. A key feature is its ability to output to DocTags, a custom markup format that preserves rich structural information. Docling is powerful and deeply integrated with IBM's AI ecosystem, including its Granite models. However, its ecosystem is more closely tied to IBM's platforms, and it may represent a heavier dependency for a project aiming for a more vendor-neutral stack.

Unstructured.io is an open-source ETL (Extract, Transform, Load) solution designed from the ground up for the age of LLMs. Its mission is to transform complex, unstructured documents (like PDFs, HTML files, and Word documents) into structured, AI-ready formats. Unlike Tika, Unstructured.io does not just extract text; it partitions documents into logical elements like `Title`, `NarrativeText`, `ListItem`, and `Table`. It can even convert tables within a PDF into a clean HTML representation. This focus on preserving and modeling the document's structure is its key advantage. By providing clean, semantically meaningful chunks of content, it enables the creation of higher-quality embeddings, which directly translates to more accurate and contextually relevant retrieval in a RAG pipeline. While it may have more dependencies than Tika (e.g., Poppler for PDFs, Tesseract for OCR), its superior output quality for AI workflows makes it the optimal choice for this project.

Web Crawling and Data Acquisition

To augment the knowledge base with external information—such as project documentation, technical articles, or competitor analysis relevant to Planner tasks—the system requires a web crawling component. The choice of tool depends on the nature of the target websites, particularly whether they are static or rely heavily on JavaScript for rendering content.

Web Crawling Tools: Scrapy vs. BeautifulSoup vs. Playwright vs. Firecrawl

The landscape of Python-based web scraping tools offers a spectrum of capabilities, from simple HTML parsing to full browser automation. For a system that needs to reliably extract data from the modern web, a tool capable of handling dynamic content is essential. Therefore, **Playwright** is the recommended primary tool, with **Firecrawl** as a compelling AI-driven alternative.

BeautifulSoup is a lightweight Python library designed for parsing HTML and XML documents. It is exceptionally easy to use and is ideal for quick, small-scale scraping tasks on websites with simple, static HTML. However, it has a major limitation: it cannot execute JavaScript. As many modern websites use JavaScript to load content dynamically, BeautifulSoup alone is often insufficient for comprehensive data acquisition.

Scrapy is a powerful and highly scalable web crawling framework. It is designed for large-scale, asynchronous scraping projects and provides a complete ecosystem with features like request scheduling, proxy management, and data processing pipelines. While extremely capable, Scrapy has a steeper learning curve and its architecture is often overkill for the more targeted data acquisition needs of a single-user system. Like BeautifulSoup, it does not handle JavaScript rendering out of the box and requires integration with browser automation tools to do so.

Playwright, developed by Microsoft, is a modern browser automation framework that provides a high-level API to control headless versions of Chromium, Firefox, and WebKit. Its primary advantage is its ability to fully render any webpage, exactly as a user would see it in their browser. This means it can effortlessly handle dynamic content, single-page applications (SPAs), and complex user interactions. It includes features like auto-waiting, which simplifies scripting by automatically waiting for elements to be ready before interacting with them. For a system that needs to reliably extract content from any website, Playwright's robustness and ability to handle the modern, JavaScript-heavy web make it the superior choice.

Firecrawl represents a new generation of AI-powered scraping tools. It is an API-driven service that simplifies the entire scraping process. Instead of writing complex selectors and handling anti-bot measures, a developer can simply provide a URL and Firecrawl will return clean, structured Markdown or JSON data. It automatically handles JavaScript rendering, proxy rotation, and other complexities. It can even use an LLM to extract specific data based on a natural language schema. While it is a commercial service with a pricing model based on usage, its ease of use and low maintenance overhead can significantly reduce development time. For developers looking to prioritize speed of implementation over fine-grained control, Firecrawl is an excellent alternative to building and maintaining custom Playwright scripts.

Document Generation and Reporting

A key feature of an intelligent Planner management system is the ability to generate reports, summaries, and other documents based on Planner data. This requires a flexible document generation library capable of producing professional-quality files in common formats like PDF, Word, and PowerPoint.

Document Generation Libraries: PDF and Office Formats

Python's ecosystem offers several excellent libraries for document generation, each specializing in different output formats. The recommended approach is to use a combination of libraries, selecting the best tool for each required format.

For generating PDF documents, the two leading contenders are **ReportLab** and **WeasyPrint**. ReportLab is a mature and powerful library that provides a low-level API for drawing text, graphics, and charts directly onto a PDF canvas. It offers precise control over every element of the document, making it ideal for creating complex, data-intensive reports with custom layouts, such as financial statements or detailed project dashboards. However, this level of control comes with a steeper learning curve. **WeasyPrint**, on the other hand, takes a different approach by converting HTML and CSS into high-quality PDFs. This is an incredibly powerful and intuitive workflow for developers already familiar with web technologies. They can use standard HTML for structure and CSS for styling, including advanced features like `@media print` rules for print-specific layouts. For generating visually rich reports from Planner data, WeasyPrint is the recommended choice. A developer can easily create an HTML template using a templating engine like Jinja2, populate it with Planner data, and render a professional-looking PDF with just a few lines of Python code.

For generating native Microsoft Office documents, dedicated libraries provide the most reliable solution. **python-docx** is the standard library for creating and manipulating Microsoft Word (.docx) files. It allows for the programmatic creation of paragraphs, headings, tables, images, and text with various formatting options. Similarly, **python-pptx** is the go-to library for generating Microsoft PowerPoint (.pptx) presentations. It enables the creation of slides, the addition of text, shapes, images, and tables, and the use of master layouts from a template. For a Planner management system, these libraries would allow the AI to fulfill requests like “Generate a Word document summarizing all overdue tasks for the ‘Q4 Marketing’ plan” or “Create a PowerPoint slide for each team member showing their assigned tasks this week.” By using these specialized libraries, the system can produce native, editable Office documents that integrate seamlessly into existing business workflows.

System Architecture and Integration Patterns

To bring these selected components together into a cohesive and functional application, a well-defined architecture is required. A microservices-inspired approach, orchestrated with Docker Compose, provides the ideal balance of modularity, scalability, and ease of development for a containerized, single-user system.

Microservices Architecture for a Single-User System

While the term “microservices” often implies large, distributed systems, its core principles—decoupling, single responsibility, and independent deployment—are highly beneficial even for a single-user application. By structuring the system as a collection of distinct, containerized services, we gain significant advantages. Each component (e.g., the RAG API, the database, the document processing worker) can be developed, tested, and updated independently. This modularity simplifies the codebase and allows for the use of the best technology for each specific job, aligning perfectly with the polyglot nature of the recommended stack (e.g., Python for the AI backend, PostgreSQL for the database).

The proposed architecture consists of several key services:
1. **Frontend Service:** A web-based user interface. For this project, this role is fulfilled by the user’s Microsoft Teams client.
2. **Orchestration Service:** The central hub that manages user interaction and invokes other services. This will be the LangChain-based RAG application, exposing an API for the frontend.
3. **Database Service:** The PostgreSQL container, running with the pgvector extension, providing both relational and vector storage.
4. **Graph Database Service (Optional):** The Neo4j container, for advanced relational analytics.
5. **Document Processing Service:** A dedicated service running Unstructured.io, which can process uploaded files asynchronously.
6. **Web Crawling Service:** A service that can execute Playwright scripts on demand to fetch external web content.

These services communicate with each other over a private container network using lightweight APIs (e.g., REST or gRPC). An API Gateway can be used as a single entry point to the system, handling concerns like authentication and routing, though for a simple single-user system, the orchestration service can often fill this role. This decoupled architecture ensures that the system is maintainable, extensible, and easy to reason about.

Docker Compose for Multi-Container Deployment

Docker Compose is the ideal tool for defining and running this multi-container application. It uses a simple YAML file (`docker-compose.yml`) to configure all the application’s services, networks, and volumes. This declarative approach provides a single source of truth for the entire application stack, making it incredibly easy to spin up a complete, isolated development environment with a single command (`docker compose up`).

A typical `docker-compose.yml` for this project would define each of the services mentioned above. For the **PostgreSQL service**, it would specify the official Postgres image, map a local volume to the container's data directory to ensure data persistence, and set environment variables for the database user and password. For the **RAG API service**, it would build a custom Docker image from a Dockerfile in the project's source code, expose the API port to the host machine, and define dependencies on the database service using the `depends_on` directive to ensure a proper startup order.

Docker Compose also manages networking. By default, it creates a single private network for all services defined in the file, allowing them to discover and communicate with each other using their service names as hostnames (e.g., the RAG API can connect to the database at `postgres:5432`). This removes the need for hardcoding IP addresses and simplifies inter-service communication. Volumes are used to persist data beyond a container's lifecycle, which is essential for the database, and can also be used for sharing files between services or for mounting source code into a container for live-reloading during development. This containerized, Compose-driven approach provides a portable, reproducible, and efficient environment for both developing and deploying the intelligent Planner management system.

Final Recommendations and Roadmap

This comprehensive analysis of the technology stack provides a clear and actionable path for developing an intelligent Microsoft Teams Planner management system. The recommendations are tailored to create a robust, flexible, and maintainable system within a containerized, single-user context.

The foundational recommendation is to adopt a **hybrid data architecture** centered on **PostgreSQL** with the **pgvector** extension. This provides a powerful, integrated solution for both structured relational data and high-dimensional vector embeddings, simplifying the overall system architecture. For future enhancements involving complex relationship analysis, **Neo4j** should be integrated as a specialized graph database.

For the core AI functionality, **LangChain** is the recommended orchestration framework. Its modularity and agentic capabilities are essential for building a system that can perform complex, multi-step actions beyond simple question-answering. The RAG pipeline should be powered by documents processed with **Unstructured.io**, which excels at creating clean, structured data optimized for LLM consumption.

Data acquisition from external web sources should be handled by **Playwright** to ensure reliable scraping of modern, dynamic websites. For document generation, a combination of **WeasyPrint** for PDFs and the **python-docx/python-pptx** libraries for native Office formats offers maximum flexibility.

The entire system should be architected as a set of decoupled services and deployed using **Docker Compose**. This approach promotes modularity, simplifies development, and ensures portability. By following this technology roadmap, the development team can efficiently build a powerful and intuitive conversational AI that seamlessly integrates with and enhances the Microsoft Teams Planner experience.

References

PostgreSQL vs MySQL comparison 2024 2025 - bytebase.com (https://www.bytebase.com/blog/postgres-vs-mysql/)

anyone_know_what_the_long_term_trend_between - reddit.com (https://www.reddit.com/r/PostgreSQL/comments/1f8ta5p/anyone_know_what_the_long_term_trend_between/)
MySQL vs Postgres in 2024 - dbconvert.com (https://dbconvert.com/blog/mysql-vs-postgres-in-2024/)
PostgreSQL vs. MySQL: Which One Is Better For Your Use Case? - integrate.io (https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/)
Postgres vs MySQL - cyberpanel.net (https://cyberpanel.net/blog/postgres-vs-mysql)
PostgreSQL vs. MySQL - flatirons.com (https://flatirons.com/blog/postgresql-vs-mysql/)
MySQL vs. PostgreSQL: What's the Difference? - aws.amazon.com (https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/)
PostgreSQL vs. MySQL: A Comprehensive Comparison - portable.io (https://portable.io/learn/psql-2-mysql)
PostgreSQL vs. MySQL: A 360-Degree Comparison of Syntax, Performance, Scalability, and Features - enterprisedb.com (https://www.enterprisedb.com/blog/postgresql-vs-mysql-360-degree-comparison-syntax-performance-scalability-and-features?lang=en)
Using PostgreSQL over MySQL in 2024 - reddit.com (https://www.reddit.com/r/PostgreSQL/comments/1e4wq0p/using_postgresql_over_mysql_in_2024/)
Neo4j Deployment Center - neo4j.com (https://neo4j.com/deployment-center/)
Simple Graph Database Setup with Neo4j and Docker Compose - medium.com (https://medium.com/@matthewghannoum/simple-graph-database-setup-with-neo4j-and-docker-compose-061253593b5a)
Docker - neo4j.com (https://neo4j.com/docs/operations-manual/current/docker/)
docker-neo4j - github.com (https://github.com/neo4j/docker-neo4j)
Deploy Neo4j with Docker - render.com (https://render.com/deploy-docker/neo4j)
Introduction - neo4j.com (https://neo4j.com/docs/operations-manual/current/docker/introduction/)
Neo4j Documentation - neo4j.com (https://neo4j.com/docs/)
neo4j - hub.docker.com (https://hub.docker.com/_/neo4j)
Deployment of Neo4j Docker container with APOC and Graph-Algorithms plugins - medium.com (https://medium.com/swlh/deployment-of-neo4j-docker-container-with-apoc-and-graph-algorithms-plugins-bf48226928f4)
Run Neo4j in a Docker container - development.neo4j.dev (https://development.neo4j.dev/developer/docker-run-neo4j/)
qdrant - github.com (https://github.com/qdrant/qdrant)
Qdrant Vector Database - qdrant.tech (https://qdrant.tech/qdrant-vector-database/)
Qdrant - Vector Database for AI - qdrant.tech (https://qdrant.tech/)
A Developer's Friendly Guide to Qdrant Vector Database - cohorte.co (https://www.cohorte.co/blog/a-developers-friendly-guide-to-qdrant-vector-database)
What is a vector database? — Qdrant - medium.com (https://medium.com/@qdrant/what-is-a-vector-database-qdrant-fb8cd9b3b524)
Pgvector vs Qdrant: Which is the Best Vector Database? - tigerdata.com (https://www.tigerdata.com/blog/pgvector-vs-qdrant)
Documentation - qdrant.tech (https://qdrant.tech/documentation/)
Overview - qdrant.tech (https://qdrant.tech/documentation/overview/)
Benchmarks - qdrant.tech (https://qdrant.tech/benchmarks/)
Qdrant - aws.amazon.com (https://aws.amazon.com/marketplace/pp/prodview-rtphb42tydtzg)
pgvector - github.com (https://github.com/pgvector/pgvector)
PostgreSQL Extensions: pgvector - tigerdata.com (https://www.tigerdata.com/learn/postgresql-extensions-pgvector)
pgvector: Embeddings and vector similarity - supabase.com (https://supabase.com/docs/guides/database/extensions/pgvector)
What is pgvector? - enterprisedb.com (https://www.enterprisedb.com/blog/what-is-pgvector)
pgvector Tutorial: Storing and Querying Vector Embeddings in PostgreSQL - datacamp.com (https://

www.datacamp.com/tutorial/pgvector-tutorial)
pgvector 0.7.0 Released - postgresql.org (https://www.postgresql.org/about/news/pgvector-070-released-2852/)
pgvector 0.5.0 Released - postgresql.org (https://www.postgresql.org/about/news/pgvector-050-released-2700/)
Use the pgvector extension in Azure Database for PostgreSQL - Flexible Server - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/how-to-use-pgvector)
PostgreSQL Vector Search Guide with pgvector - northflank.com (https://northflank.com/blog/postgresql-vector-search-guide-with-pgvector)
PostgreSQL as a Vector Database: Using pgvector - tigerdata.com (https://www.tigerdata.com/blog/postgresql-as-a-vector-database-using-pgvector)
Neo4j Alternative: What Are My Open-Source DB Options? - memgraph.com (https://memgraph.com/blog/neo4j-alternative-what-are-my-open-source-db-options)
Neo4j vs. Memgraph - memgraph.com (https://memgraph.com/blog/neo4j-vs-memgraph)
Free/cheaper alternatives to Neo4j for a personal project? - reddit.com (https://www.reddit.com/r/Database/comments/jpmnxp/freecheaper_alternatives_to_neo4j_for_a/)
Neo4j Alternatives & Competitors - gartner.com (https://www.gartner.com/reviews/market/cloud-database-management-systems/vendor/neo4j/product/neo4j-graphdatabase/alternatives)
Neo4j Alternatives - puppygraph.com (https://www.puppygraph.com/blog/neo4j-alternatives)
Neo4j Graph Database Competitors and Alternatives - g2.com (https://www.g2.com/products/neo4j-graph-database/competitors/alternatives)
Neo4j Alternative: Open-Source, Distributed, and Lightning-Fast - nebula-graph.io (https://www.nebula-graph.io/posts/Neo4j_Alternative_Open-Source_Distributed_and_Lightning_Fast)
DataWalk as a Neo4j Alternative - datawalk.com (https://datawalk.com/neo4jalternative/)
Neo4j Graph Database Alternatives - softwarereviews.com (https://www.softwarereviews.com/categories/119/products/4481/alternatives)
Top 10 Open-Source Graph Databases - index.dev (https://www.index.dev/blog/top-10-open-source-graph-databases)
Awesome-GraphRAG - github.com (https://github.com/DEEP-PolyU/Awesome-GraphRAG)
Graphiti: Building Temporal Knowledge Graphs for Agentic Memory - neo4j.com (https://neo4j.com/blog/developer/graphiti-knowledge-graph-memory/)
Graph RAG - huggingface.co (https://huggingface.co/papers?q=Graph+RAG)
Building Temporal Knowledge Graphs with Graphiti - falkordb.com (https://www.falkordb.com/blog/building-temporal-knowledge-graphs-graphiti/)
graphiti - github.com (https://github.com/getzep/graphiti)
GraphRAG Explained: Enhancing RAG with Knowledge Graphs - medium.com (https://medium.com/@zilliz_learn/graphrag-explained-enhancing-rag-with-knowledge-graphs-3312065f99e1)
From RAG to GraphRAG: A Technical Review - arxiv.org (https://arxiv.org/abs/2506.05690)
Top 10 Open-Source RAG Evaluation Frameworks You Must Try - medium.com (https://walseisarel.medium.com/top-10-open-source-rag-evaluation-frameworks-you-must-try-15c40d0ba2c0)
RAG vs. GraphRAG: A Comprehensive Comparison for Question Answering - arxiv.org (https://arxiv.org/html/2502.11371v1)
RAG vs. GraphRAG: A Comprehensive Comparison for Question Answering - arxiv.org (https://arxiv.org/abs/2502.11371)
LlamaIndex vs LangChain vs Haystack - medium.com (https://medium.com/@heyamit10/llamaindex-vs-langchain-vs-haystack-4fa8b15138fd)
LangChain, LlamaIndex, or Haystack: Which Framework Suits Your LLM Needs? - medium.com (https://dkaarthick.medium.com/langchain-llamaindex-or-haystack-which-framework-suits-your-llm-needs-7408fee8ab1e)
LlamaIndex vs LangChain vs Haystack: Choosing the right one for your LLM application - linkedin.com

(https://www.linkedin.com/pulse/llamaindex-vs-langchain-haystack-choosing-right-one-subramaniam-yvere)
How does Haystack differ from other search frameworks like LangChain and LlamaIndex? - milvus.io (https://milvus.io/ai-quick-reference/how-does-haystack-differ-from-other-search-frameworks-like-langchain-and-llamaindex)
A Comparative Analysis of LLM Application Frameworks for Enterprise AI - ijgisonline.com (https://ij-gis.pubpub.org/pub/6yecqicl)
What are the differences between LangChain and other LLM frameworks like LlamaIndex or Haystack? - milvus.io (https://milvus.io/ai-quick-reference/what-are-the-differences-between-langchain-and-other-llm-frameworks-like-llamaindex-or-haystack)
Case Study: GenAI Tools - tuhinsharma.com (https://tuhinsharma.com/blogs/case-study-genai-tools/)
LlamaIndex vs LangChain: Which is better? - blog.n8n.io (https://blog.n8n.io/llamaindex-vs-langchain/)
IBM Granite-Docling: End-to-end document conversion for AI - ibm.com (https://www.ibm.com/new/announcements/granite-docling-end-to-end-document-conversion)
Docling: An open-source toolkit for AI-driven document conversion - research.ibm.com (https://research.ibm.com/blog/docling-generative-AI)
Docling - docling-project.github.io (https://docling-project.github.io/docling/)
IBM Automation Document Processing - ibm.com (https://www.ibm.com/products/document-processing)
ibm-granite/granite-docling-258M - huggingface.co (https://huggingface.co/ibm-granite/granite-docling-258M)
Docling: An Efficient Open-Source Toolkit for AI-Driven Document Conversion - research.ibm.com (https://research.ibm.com/publications/docling-an-efficient-open-source-toolkit-for-ai-driven-document-conversion)
docling - github.com (https://github.com/docling-project/docling)
Docling - docling.ai (https://www.docling.ai/)
Docling’s rise: How IBM is making data LLM-ready - ibm.com (https://www.ibm.com/think/news/doclings-rise-llm-ready-data)
Build a document question-answering system with Docling and Granite - ibm.com (https://www.ibm.com/think/tutorials/build-document-question-answering-system-with-docling-and-granite)
Apache Tika, an underrated alternative to Unstructured.io for RAG and fine-tuning - reddit.com (https://www.reddit.com/r/LocalLLaMA/comments/1aq1qkd/apache_tika_an_underrated_alternative_to/)
Unstructured Answer: Structured vs. Unstructured Data in AI - restack.io (https://www.restack.io/p/unstructured-answer-structured-vs-unstructured-cat-ai)
unstructured - github.com (https://github.com/Unstructured-IO/unstructured)
Top Unstructured Data Tools - wajidkhan.info (https://wajidkhan.info/top-unstructured-data-tools/)
Unstructured Data, Apache Tika, and Beer - dankeeley.wordpress.com (https://dankeeley.wordpress.com/2016/05/18/unstructured-data-apache-tika-and-beer/)
Apache Tika – a content analysis toolkit - tika.apache.org (https://tika.apache.org/)
Resources for Advanced Document Processing - cwiki.apache.org (https://cwiki.apache.org/confluence/display/TIKA/Resources+for+Advanced+Document+Processing)
Unstructured is an ETL solution for LLMs - news.ycombinator.com (https://news.ycombinator.com/item?id=36616799)
MinIO and Tika for Text Extraction - blog.min.io (https://blog.min.io/minio-tika-text-extraction/)
Active ‘apache-tika’ Questions - stackoverflow.com (https://stackoverflow.com/questions/tagged/apache-tika?tab=Active)
Mastering RAG: How to Select an Embedding Model - galileo.ai (https://galileo.ai/blog/mastering-rag-how-to-select-an-embedding-model)
Generate embeddings with Azure OpenAI - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-generate-embeddings)
Choose the Best Embedding Model for Your Retrieval-Augmented Generation (RAG) System - enter-

prisebot.ai (https://www.enterprisebot.ai/blog/choose-the-best-embedding-model-for-your-retrieval-augmented-generation-rag-system)
RAG and embeddings: is it better to embed text with labels or not? - community.openai.com (https://community.openai.com/t/rag-and-embeddings-is-it-better-to-embed-text-with-labels-or-not/604100)
Retrieval Augmented Generation (RAG) - docs.mistral.ai (https://docs.mistral.ai/guides/rag/)
Retrieval-Augmented Generation (RAG) - promptingguide.ai (https://www.promptingguide.ai/research/rag)
Best Open Source Web Scraping Libraries - firecrawl.dev (https://www.firecrawl.dev/blog/best-open-source-web-scraping-libraries)
BeautifulSoup4 vs Scrapy Comparison - firecrawl.dev (https://www.firecrawl.dev/blog/beautifulsoup4-vs-scrapy-comparison)
Best AI Web Scraping Tools - scrapeops.io (https://scrapeops.io/web-scraping-playbook/best-ai-web-scraping-tools/)
What is the best scraper tool right now? Firecrawl, Scrapy, etc. - reddit.com (https://www.reddit.com/r/LocalLLaMA/comments/1jw4yqv/what_is_the_best_scraper_tool_right_now_firecrawl/)
AI-Powered Web Scraping Solutions 2025 - firecrawl.dev (https://www.firecrawl.dev/blog/ai-powered-web-scraping-solutions-2025)
Scrapy vs. Playwright - brightdata.com (https://brightdata.com/blog/web-data/scrapy-vs-playwright)
Firecrawl vs. BeautifulSoup - blog.apify.com (https://blog.apify.com/firecrawl-vs-beautifulsoup/)
Top 10 Tools for Web Scraping - firecrawl.dev (https://www.firecrawl.dev/blog/top_10_tools_for_web_scraping)
Top 5 Open-Source Web Scraping Tools for Developers - firecrawl.dev (https://www.firecrawl.dev/blog/top-5-open-source-web-scraping-tools-for-developers)
Python Web Scraping Projects - firecrawl.dev (https://www.firecrawl.dev/blog/python-web-scraping-projects)
Generate PDFs in Python with Libraries - templated.io (https://templated.io/blog/generate-pdfs-in-python-with-libraries/)
Generating PDF in Python - glukhov.org (https://www.glukhov.org/post/2025/05/generating-pdf-in-python/)
Top 10 Ways to Generate PDFs in Python - nutrient.io (https://www.nutrient.io/blog/top-10-ways-to-generate-pdfs-in-python/)
Create selectable PDF files with Lambda, Python, and ReportLab - dev.to (https://dev.to/aws-builders/create-selectable-pdf-files-with-lambda-python-and-reportlab-5gp0)
Flask PDF Generation: ReportLab, WeasyPrint, and PDFKit Compared - codingeasypeasy.com (https://www.codingeasypeasy.com/blog/flask-pdf-generation-reportlab-weasyprint-and-pdfkit-compared)
Creating Various PDF Files via Python - medium.com (https://akpolatcem.medium.com/creating-various-pdf-files-via-python-eba91a40df9d)
Python PDF Generation from HTML with WeasyPrint - dev.to (https://dev.to/bowmanjd/python-pdf-generation-from-html-with-weasyprint-538h)
Create PDF Documents in Python with ReportLab - pythonassets.com (https://pythonassets.com/posts/create-pdf-documents-in-python-with-reportlab/)
Best python library for generating PDFs? - reddit.com (https://www.reddit.com/r/Python/comments/82z6cw/best_python_library_for_generating_pdfs/)
Tips for using WeasyPrint for PDF generation - reddit.com (https://www.reddit.com/r/Python/comments/jmfbwk/tips_for_using_weasyprint_for_pdf_generation/)
python-pptx - pypi.org (https://pypi.org/project/python-pptx/)
python-pptx - python-pptx.readthedocs.io (https://python-pptx.readthedocs.io/en/latest/index.html)
How can I automate the generation of a PowerPoint with Python? - stackoverflow.com (https://stackoverflow.com/questions/71359430/how-can-i-automate-the-generation-of-a-powerpoint-with-python)
python-pptx - github.com (https://github.com/scanny/python-pptx)

Convert DOC to PPT in Python - blog.aspose.com (https://blog.aspose.com/total/convert-doc-to-ppt-python/)
Convert DOC to PPT in Python - gist.github.com (https://gist.github.com/aspose-com-gists/3d5bf343526692ab0cf44ad9c0487ece)
Docxtemplater - docxtemplater.com (https://docxtemplater.com/)
Creating and Updating PowerPoint Presentations in Python using python-pptx - geeksforgeeks.org (https://www.geeksforgeeks.org/python/creating-and-updating-powerpoint-presentations-in-python-using-python-pptx/)
python-pptx - products.documentprocessing.com (https://products.documentprocessing.com/editor/python/python-pptx/)
wokelo-docs - pypi.org (https://pypi.org/project/wokelo-docs/)
Multi-container applications with Docker Compose - learn.microsoft.com (https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/multi-container-applications-docker-compose)
Docker Compose overview - docs.docker.com (https://docs.docker.com/compose/)
Working with Docker Compose - code.visualstudio.com (https://code.visualstudio.com/docs/containers/docker-compose)
Using Awesome-Compose to Build and Deploy Your Multi-Container Application - docker.com (https://www.docker.com/blog/using-awesome-compose-to-build-and-deploy-your-multi-container-application/)
Running multi-container applications - docs.docker.com (https://docs.docker.com/get-started/docker-concepts/running-containers/multi-container-applications/)
compose - github.com (https://github.com/docker/compose)
How to Run Multiple Containers with Docker Compose - freecodecamp.org (https://www.freecodecamp.org/news/run-multiple-containers-with-docker-compose/)
An Empirical Study of Docker Compose Configuration Patterns - arxiv.org (https://arxiv.org/html/2305.11293v2)
Use Docker Compose to deploy multiple containers - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/ai-services/containers/docker-compose-recipe)
Tutorial: Deploy a multi-container group using Docker Compose - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/container-instances/tutorial-docker-compose)
Technology Stack for Microservices - aalpha.net (https://www.aalpha.net/blog/technology-stack-for-microservices/)
Pattern: Microservice Architecture - microservices.io (https://microservices.io/patterns/microservices.html)
Choosing the Right Technology Stack for Your Microservices - medium.com (https://medium.com/@jpromanonet/choosing-the-right-technology-stack-for-your-microservices-a863e34bf755)
Microservices architecture style - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices)
Microservices Technology Stack Delivers Content as a Service - contentstack.com (https://www.contentstack.com/cms-guides/microservices-technology-stack-delivers-content-as-a-service)
What is a Microservice Tech Stack? - hygraph.com (https://hygraph.com/blog/what-is-microservice-tech-stack)
Pattern: Multiple technology stacks per service - microservices.io (https://microservices.io/articles/dark-energy-dark-matter/dark-energy/multiple-technology-stacks.html)
Microservice Architecture Tech Stack Essentials - springfuse.com (https://www.springfuse.com/microservice-architecture-tech-stack-essentials/)
5 Best Technologies to Build Microservices Architecture - clariontech.com (https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture)
What is a microservice architecture? - contentful.com (https://www.contentful.com/resources/microservice-architecture/)