Using Generative AI to Tackle API Sprawl in Enterprises

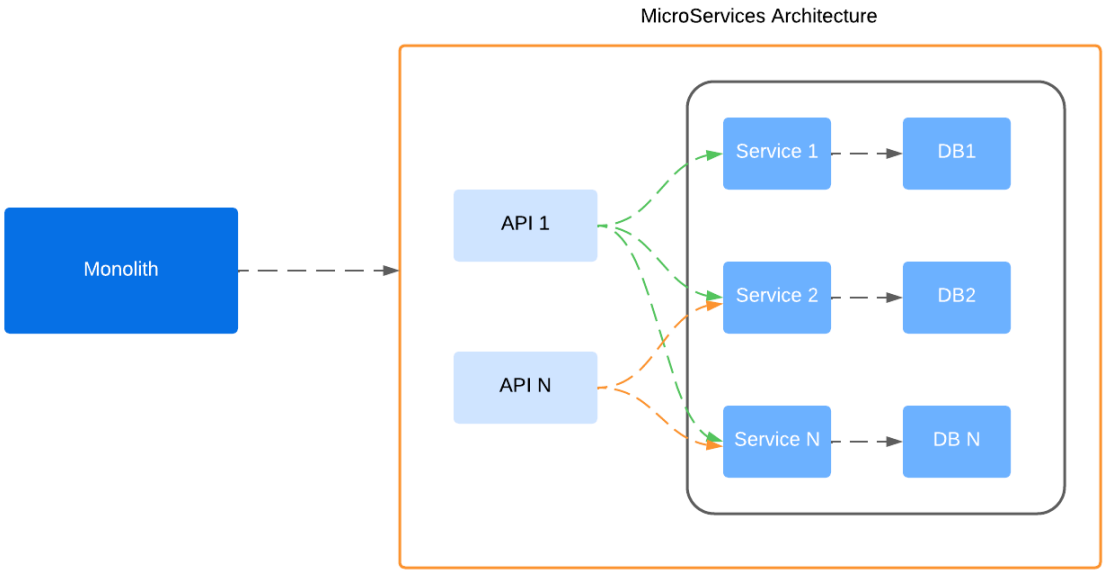
Intelligent Governance and Automation for Efficient API Management

Anirudha Karandikar

AGENDA

- History and Context - Monolith vs Microservices
- Problem Statement – API Sprawl in Enterprises
- Proposed Construct – Centralized API Governance + Automation
- Governance Model
- API Discovery Tool
- API Documentation
- Strategies to control existing API sprawl
- Conclusion

History and Context: Monolith Vs MicroServices



Microservices architecture is a software design approach where an application is built as a collection of small, independent services that communicate through APIs. Each service focuses on a specific business function, unlocking following potential benefits:

- Scalability
- Faster Development
- Resilience
- Easier Maintenance
- Agility
- Technology Diversity
- Cost Efficiency

KEY MILESTONES

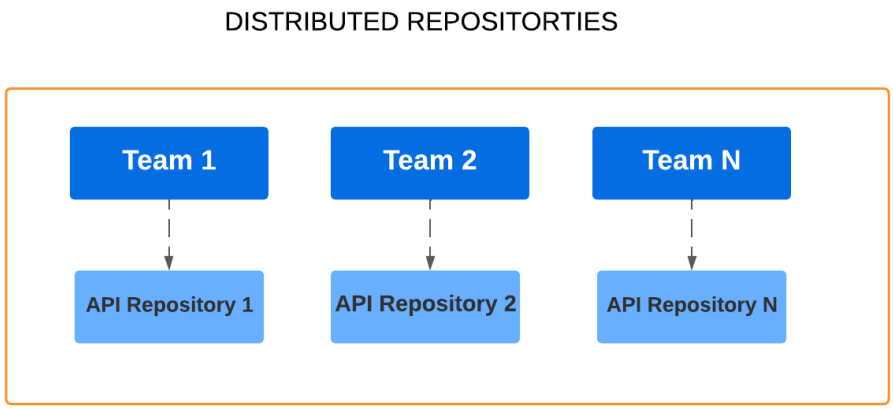
2011-2013: Companies like Netflix and Amazon pioneered the adoption of microservices.

2014-2015: The term "microservices" became more widely recognized in the software architecture community

2016-Present: With the rise of cloud-native technologies, containerization (Docker), and orchestration tools (Kubernetes), microservices adoption accelerated across various industries

Problem Statement : API Sprawl

- Widespread adoption of REST APIs and microservices has led to an explosion in the number of APIs.
- Many enterprises adopted distributed development as orgs within enterprises moved at different pace and this setup provided team autonomy and reduced conflicts.

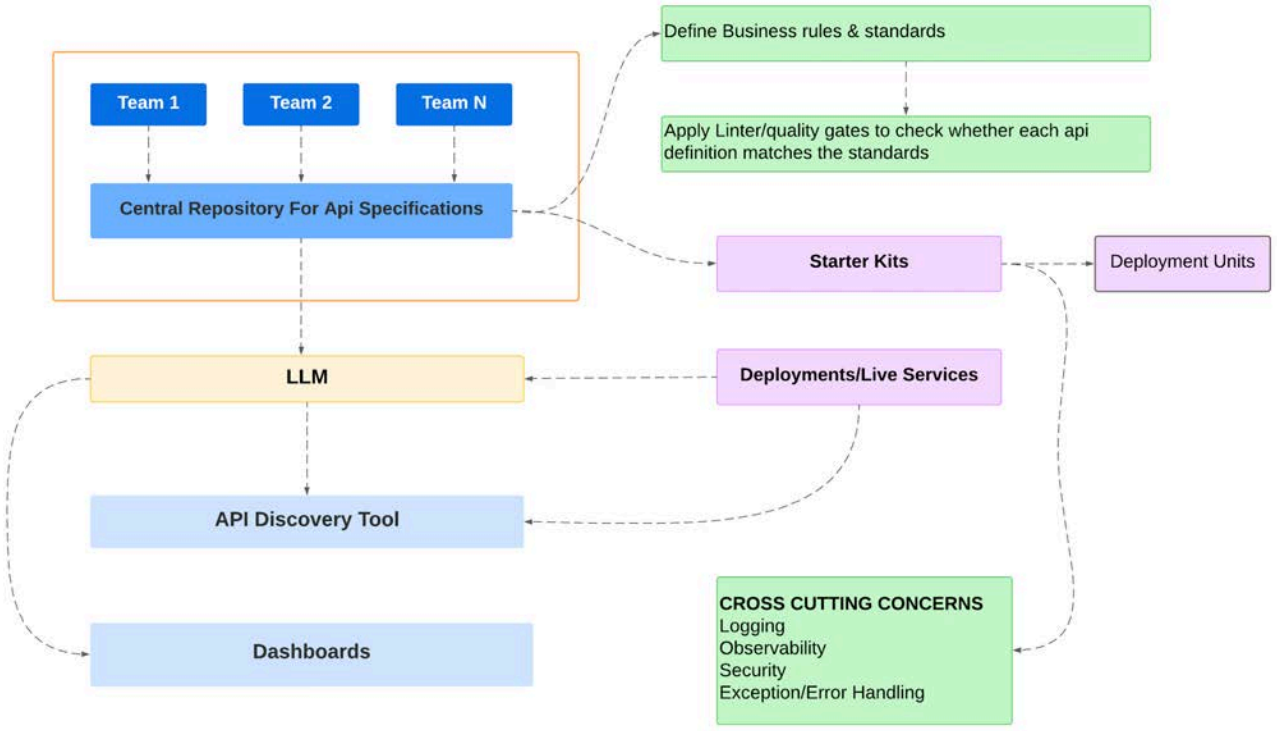


- As enterprises scale, managing APIs becomes difficult due to:
- **Duplication**: Different teams creating APIs that serve the same function.
- **Harder to exercise security controls and measures:** Non standard APIs expose potential attack vectors. Also, a large number of apis expose greater attack surface.
- **Operational Inefficiencies**: It's hard to enforce standards, leading to mismatch in operational and maintenance requirements leading to performance and maintenance issues.
- **Inconsistent Documentation**: Incomplete or outdated API documentation makes it hard to find and reuse existing APIs.

Proposed Construct:Centralized Governance + Automation

Why Centralization Matters:

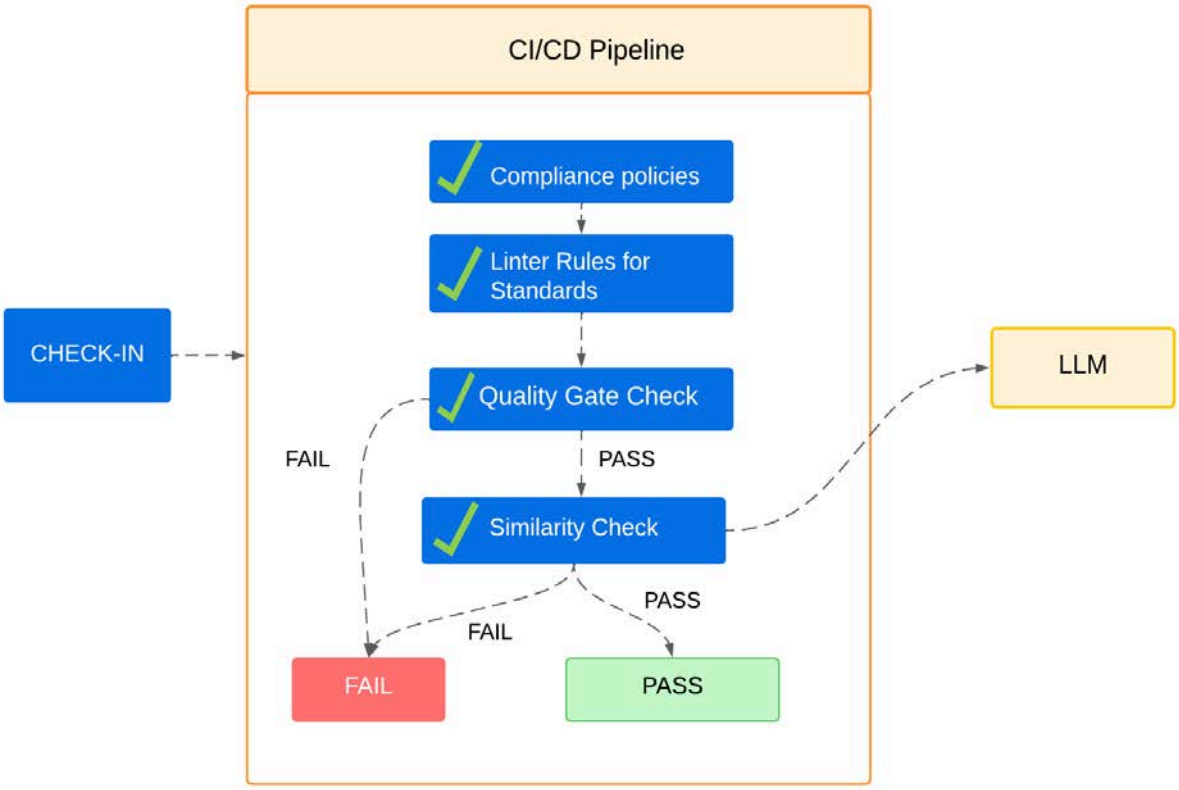
A decentralized approach leads to scattered APIs, inconsistent standards, and difficulty in tracking changes or enforcing governance. By moving to a centralized monorepo model, organizations can have a single repository that offers **transparency** and improves **visibility** as teams can access a unified source for API definitions which increase collaboration and accelerates development cycles.



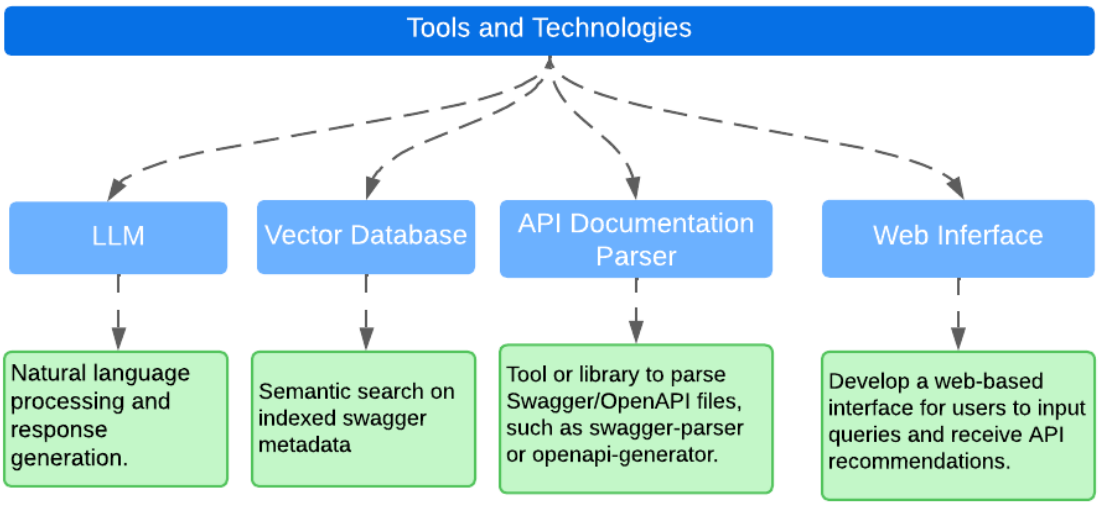
This approach provides a robust governance model around API contracts, fostering standardization, compliance, and efficient access in the development process.

Governance Model

There are many ways in which governance can be applied. Below represents a sample based on CI/CD paradigm



API Discovery Tool



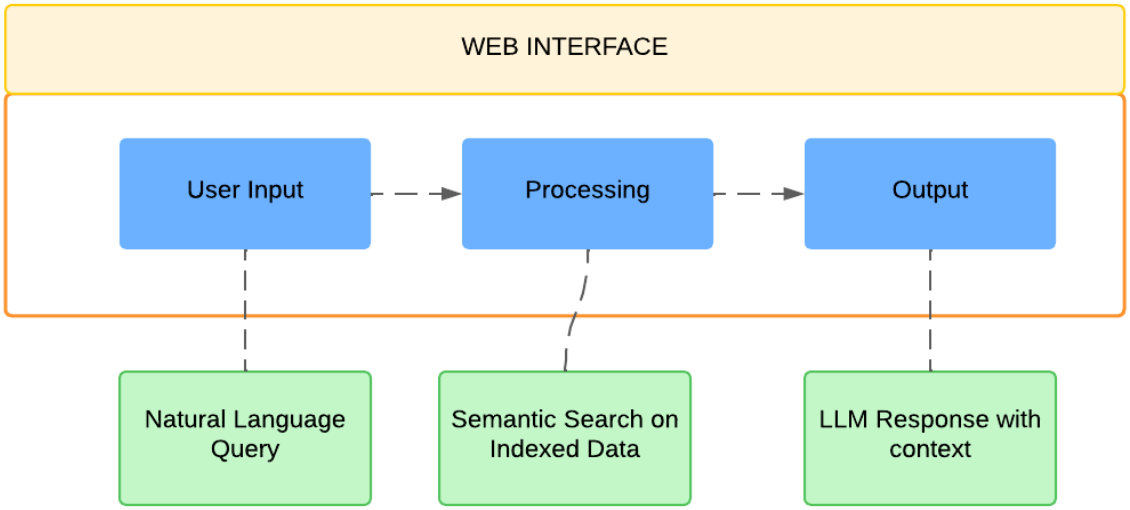
Steps involved

1. **Data Preparation :** Data extraction (including endpoint URLs, request methods (GET, POST, etc.), parameters, responses, and descriptions which will be used to construct prompts for the LLM and organize it in a structured format which is easy to feed.
2. **Indexing and Context Building**: Index the metadata prepared in step 1 and store it as embeddings in the vector database. Also, build a mapping system that links API functionalities to natural language queries. For example, mapping "How do I get user details?" to the relevant API endpoints in the repository.
3. **Building the Query Interface:** Create an interface for natural language query parsing where users can input natural language queries **.** Also**,** craft prompts for the LLM to query the indexed Swagger metadata effectively(prompt engineering).Use

the vector database to perform a semantic search based on the user query. Find the closest matches among the API descriptions and return those to the user.

4. **Response Generation:** Once relevant information is identified, use the LLM to generate a detailed response.Enhance the LLM's responses by including parts of the original Swagger documentation that are relevant to the query(Contextual Augmentation)

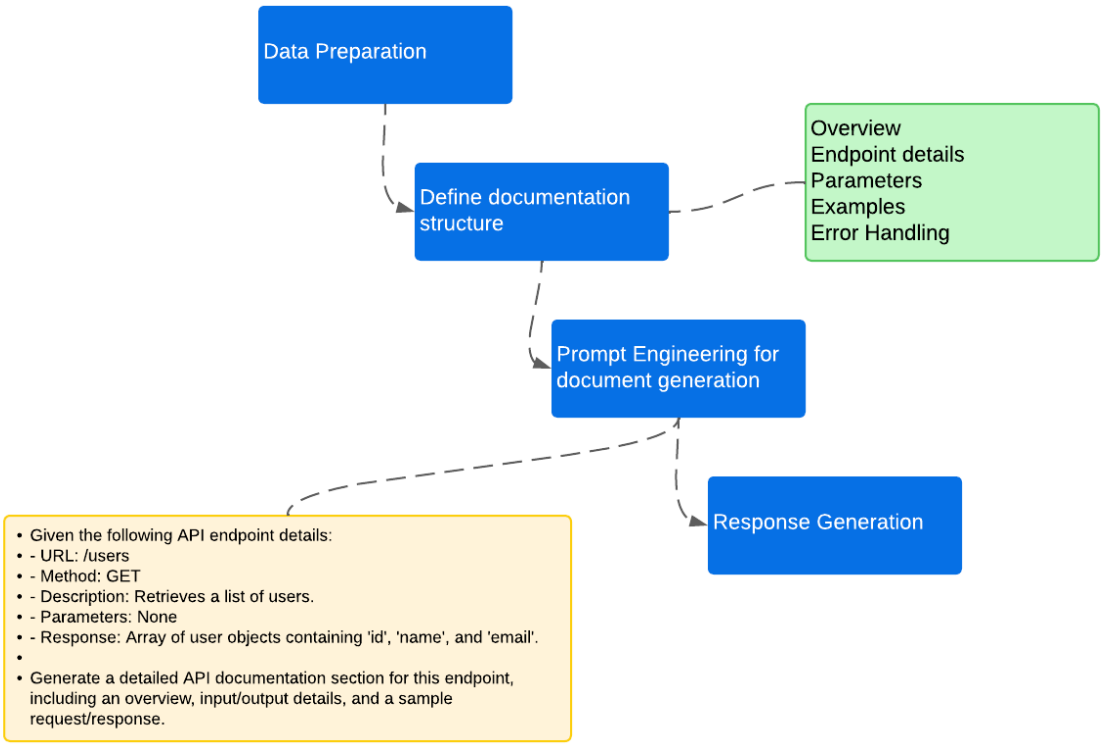
Flow



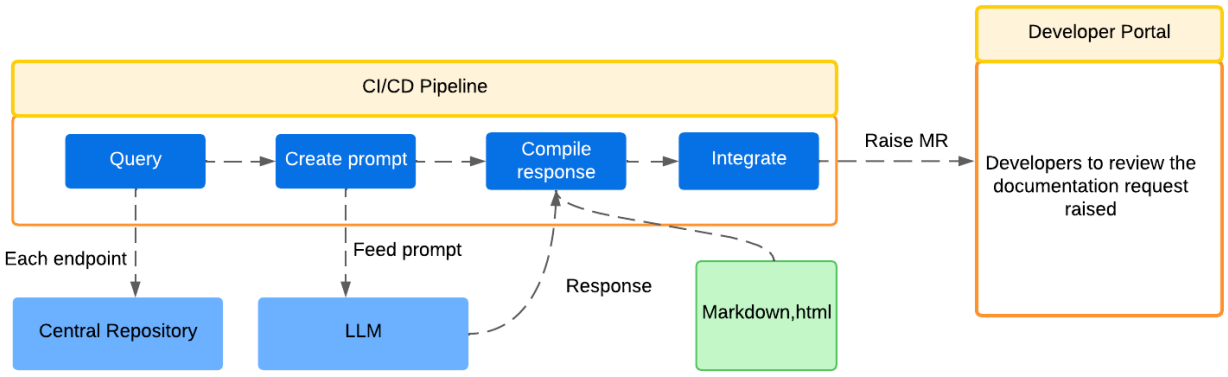
- ☑ User Input(Natural language query)
- ☑ Processing (semantic search on indexed data)
- ☑ Output (LLM response with context)

API Documentation

Steps involved:



Automation


Developer Portal
CI/CD Pipeline
Query
Create prompt
Compile response
Integrate
Raise MR
Developers to review the documentation request raised
Each endpoint
Feed prompt
Response
Central Repository
LLM
Markdown,html

Strategies to address and control existing API sprawl

1. **Centralize API Definitions and Establish Common Standards**: Begin by gradually moving all API definitions to a centralized repository. This unified approach fosters better visibility and standardization, ensuring that every team adheres to a common set of guidelines and best practices.
2. **Identify Unused or Redundant APIs**: Utilize AI-driven insights to analyze API usage patterns. Identify APIs that are outdated, underutilized, or have multiple versions serving the same function. This will help streamline the API portfolio by deprecating or phasing out unnecessary APIs.
3. **Cluster Similar APIs for Consolidation**: Leverage AI to detect cohorts of similar APIs by examining attributes like request/response structures, functionality, and categorization. Grouping these similar APIs allows for strategic consolidation, creating more robust and versatile APIs while reducing duplication.
4. **Implement Custom Tagging for Better Management**: Use custom tags to categorize APIs based on their purpose (e.g., public-facing vs. internal) and deployment platforms. This makes it easier to phase out, modernize, or update targeted groups of APIs, facilitating a more organized and controlled evolution of the API ecosystem.

These strategies, guided by AI insights, help enterprises move toward a more structured and efficient API environment, mitigating the risks and inefficiencies associated with API sprawl.

Conclusion

Generative AI is transforming the API landscape by enabling intelligent API discovery, similarity detection, and automated documentation, thereby streamlining the entire API lifecycle.

By leveraging AI, developers can prevent the creation of duplicate APIs, effortlessly find existing solutions, and receive real-time suggestions that enhance productivity and accelerate time-to-market.

Moreover, integrating AI with API gateways facilitates continuous monitoring of API usage patterns, empowering enterprises to optimize, consolidate, or retire APIs based on actionable insights. This approach not only simplifies governance but also drives resource optimization, setting the foundation for a more agile and efficient API ecosystem.

In essence, right use of AI becomes the catalyst that empowers enterprises to navigate API sprawl, turning complexity into a managed, strategic asset.