RAG Hero

Creating Searchable Vector Databases

Jason Beres

jasonb@infragistics.com

INFRAGISTICS®

The RAG Process: From Query to Response

User Query

Your natural language question or request initiates the process.

Vector Retrieval

The query is embedded and matched against indexed document chunks in Azure AI Search.

Context Augmentation

Relevant retrieved context is merged with the original query, creating an enhanced prompt.

LLM Generation

The enhanced query is sent to a Large Language Model (LLM) to formulate a detailed answer.

Response Delivery

The LLM's tailored and factual response is presented back to the user.

Understanding RAG and Vector Databases

- **Retrieval-Augmented Generation (RAG):**
    - Enhances Large Language Models (LLMs) with an information retrieval component.
    - Retrieves relevant data from an external knowledge base to generate more accurate, contextual, and up-to-date responses.
    - Significantly reduces hallucinations and provides grounded answers.
- **Vector Databases:**
    - Specialized storage for high-dimensional vector embeddings.
    - Store numerical representations of data (text, images, audio) capturing semantic meaning.
    - Efficiently query by comparing vector similarity to find semantically related content.
    - Ideal for semantic search, recommendation systems, and RAG applications.

How Does This Work?

Vectorize Your Question

Your natural language question is transformed into a high-dimensional numerical representation (an embedding vector) using an advanced embedding model. This vector captures the semantic meaning of your query.

Find Closest Vectors

The vector database efficiently compares your query's embedding vector against all stored document/chunk vectors. Similarity is computed using metrics like cosine similarity, indicating how closely aligned two vectors are in space.

Retrieve Top-K Closest

Based on the similarity scores, the database returns the 'top-K' vectors that are most semantically relevant to your original question. These vectors point to the most pertinent pieces of information in the knowledge base.

LLM Delivers Grounded Answer

In a RAG system, these top-K retrieved chunks are then provided as context to a Large Language Model. The LLM uses this specific, relevant information to generate an accurate, contextual, and grounded answer, minimizing hallucinations.

INFRAGISTICS

Ways to Search

Vector Similarity

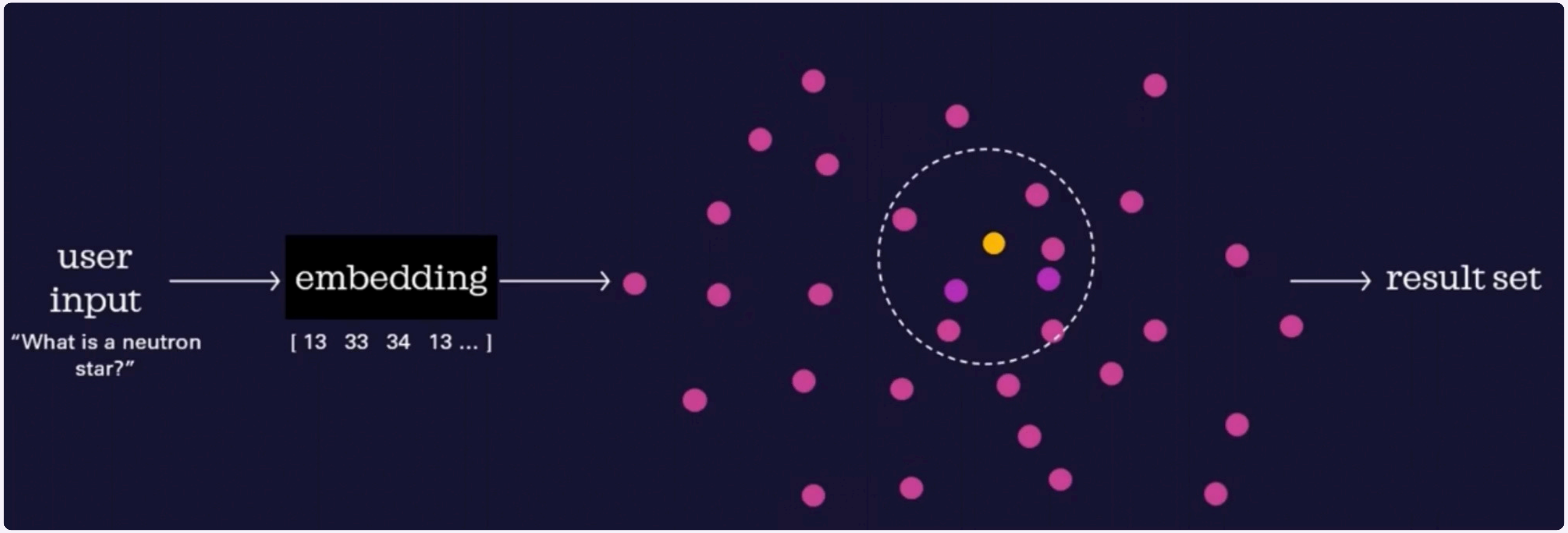
Semantic search using embeddings for conceptual matches beyond keywords

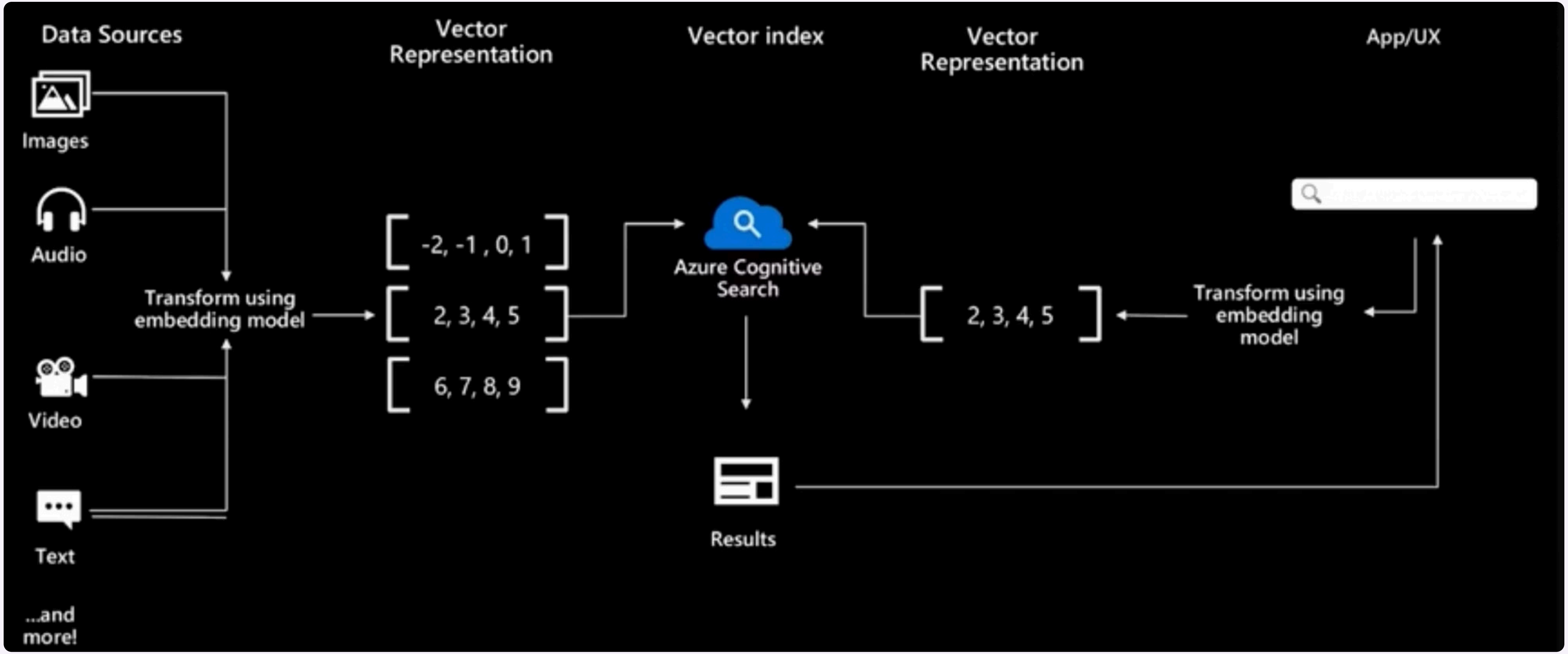
Full-Text Search

Advanced text analysis with tokenization, stemming, and relevance ranking

Filter & Facet

Drill down results by specific criteria and build dynamic navigation (Azure)




Data Sources
Vector Representation
Vector index
Vector Representation
App/UX
Images
Audio
Transform using embedding model
-2, -1 , 0, 1
2, 3, 4, 5
6, 7, 8, 9
Azure Cognitive Search
2, 3, 4, 5
Transform using embedding model
Video
Results
Text
...and more!

Using Azure AI Search Index

1

Azure Portal

Interactive web-based UI for visual index creation and management—ideal for exploration and quick prototyping

2

REST API

Direct HTTP calls to Azure AI Search endpoints for programmatic control and automation

3

Language SDKs

Official libraries for .NET, Java, Python, and JavaScript with strong typing and integrated tooling

4

Infrastructure as Code

ARM templates, Bicep, or Terraform definitions for repeatable deployments across environments

This Project's Approach: Code-first index creation using C# where the index definition is built as a SearchIndex object with fields, vector profiles, and analyzers defined programmatically, then pushed to the service with CreateOrUpdateIndexAsync for version control and consistency.

How to Search

This example of Azure AI Search demonstrates a simple implementation of a Search.

```
using Azure.Search.Documents;
using Azure.Search.Documents.Models;
using Azure;

// Assume 'searchClient' is an initialized SearchClient connected to your Azure AI Search index.
// You would typically get this from a SearchIndexClient, e.g., searchIndexClient.GetSearchClient("your-index-name");

string userQuery = "Tell me about the history of artificial intelligence";

// Define search options with a vector query
var searchOptions = new SearchOptions
{
    VectorSearch = new VectorSearchOptions
    {
        Queries = {
            new VectorizableTextQuery(userQuery)
            {
                KNearestNeighborsCount = 5, // Retrieve the 5 most similar documents
                Fields = { "contentVector" } // Specify the vector field in your index
            }
        }
    },
    Select = { "id", "title", "summary" } // Select fields to return in the results
};

// Execute the search
SearchResults response = await searchClient.SearchAsync(null, searchOptions);

Console.WriteLine("Vector Search Results:");
await foreach (SearchResult result in response.GetResultsAsync())
{
    Console.WriteLine($"Score: {result.Score}");
    Console.WriteLine($"Document ID: {result.Document["id"]}");
    Console.WriteLine($"Title: {result.Document["title"]}");
    Console.WriteLine($"Summary: {result.Document["summary"]}");
    Console.WriteLine("--------------------");
}
```

Key Components Explained

- `SearchClient`: The primary client object used to interact with your Azure AI Search index, allowing you to perform search operations.
- `SearchOptions`: A versatile object that encapsulates various search parameters, including vector search configurations, filters, order-by clauses, and which fields to return.
- `VectorSearchOptions`: Nested within `SearchOptions`, this specifies the details for performing a vector search.
- `VectorizableTextQuery`: Represents a text query that will be vectorized by the search service using an underlying embedding model defined in your index. It's crucial for semantic search.
- `KNearestNeighborsCount`: Determines how many top similar documents (K) to retrieve based on the vector similarity.
- `Fields`: Specifies which vector fields in your index should be used for the vector search. This must match the name of your vector field (e.g., "contentVector").
- `Select`: An important option in `SearchOptions` that dictates which fields from the matching documents should be returned in the search results, optimizing performance and data transfer.

Demo

Please share your event and session feedback!

http://aka.ms/MSIgniteNYCSurvey

http://aka.ms/MSIgniteNYCSessionsSurvey

Thank You to our sponsors

Microsoft

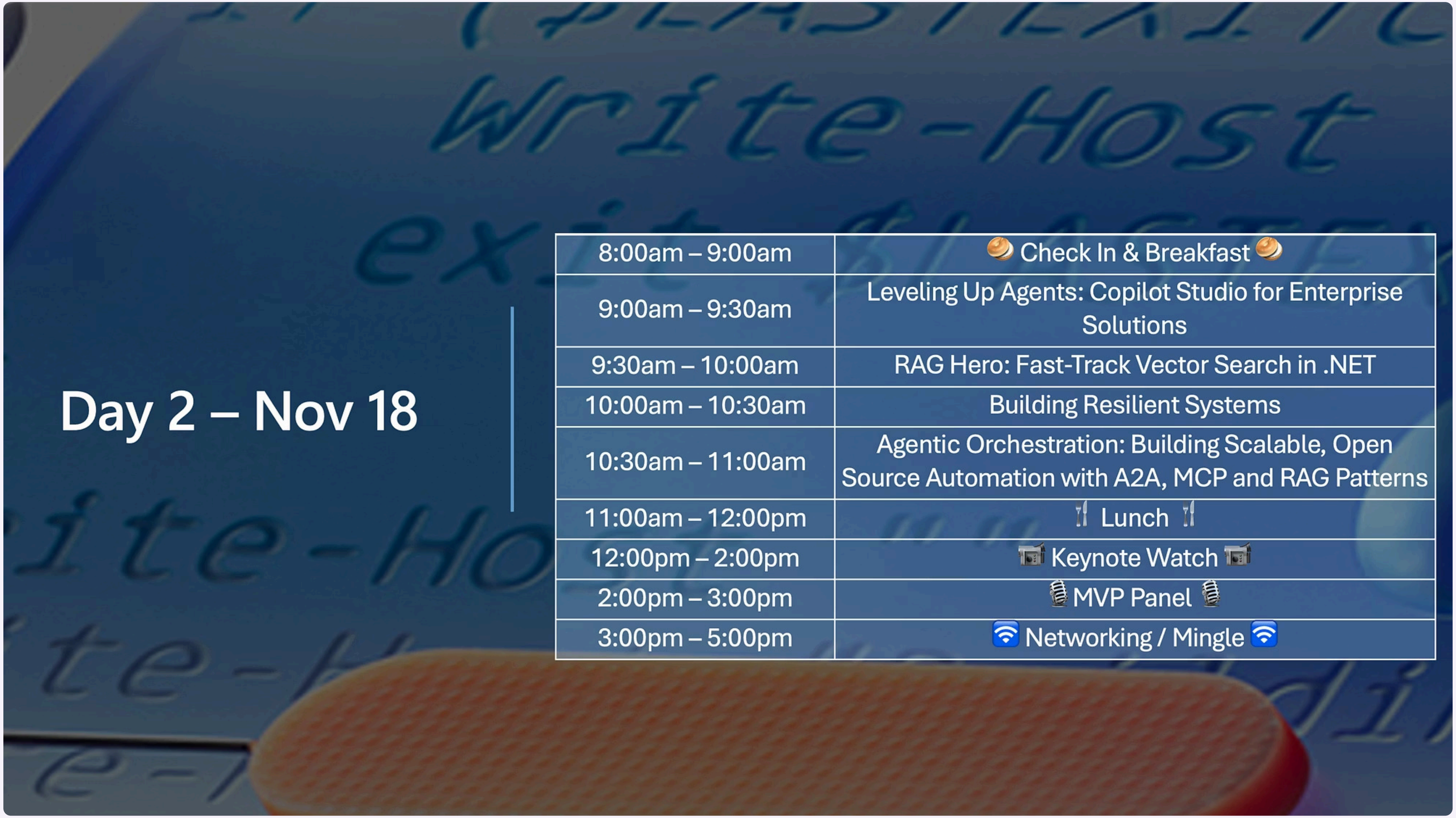
Morgan Stanley

INFRAGISTICS

apidays

uno PLATFORM

And next time -
Your Company!



Day 2 – Nov 18

8:00am – 9:00am	🥯 Check In & Breakfast 🥯
9:00am – 9:30am	Leveling Up Agents: Copilot Studio for Enterprise Solutions
9:30am – 10:00am	RAG Hero: Fast-Track Vector Search in .NET
10:00am – 10:30am	Building Resilient Systems
10:30am – 11:00am	Agentic Orchestration: Building Scalable, Open Source Automation with A2A, MCP and RAG Patterns
11:00am – 12:00pm	🍴 Lunch 🍴
12:00pm – 2:00pm	📹 Keynote Watch 📹
2:00pm – 3:00pm	🎙 MVP Panel 🎙
3:00pm – 5:00pm	🛜 Networking / Mingle 🛜