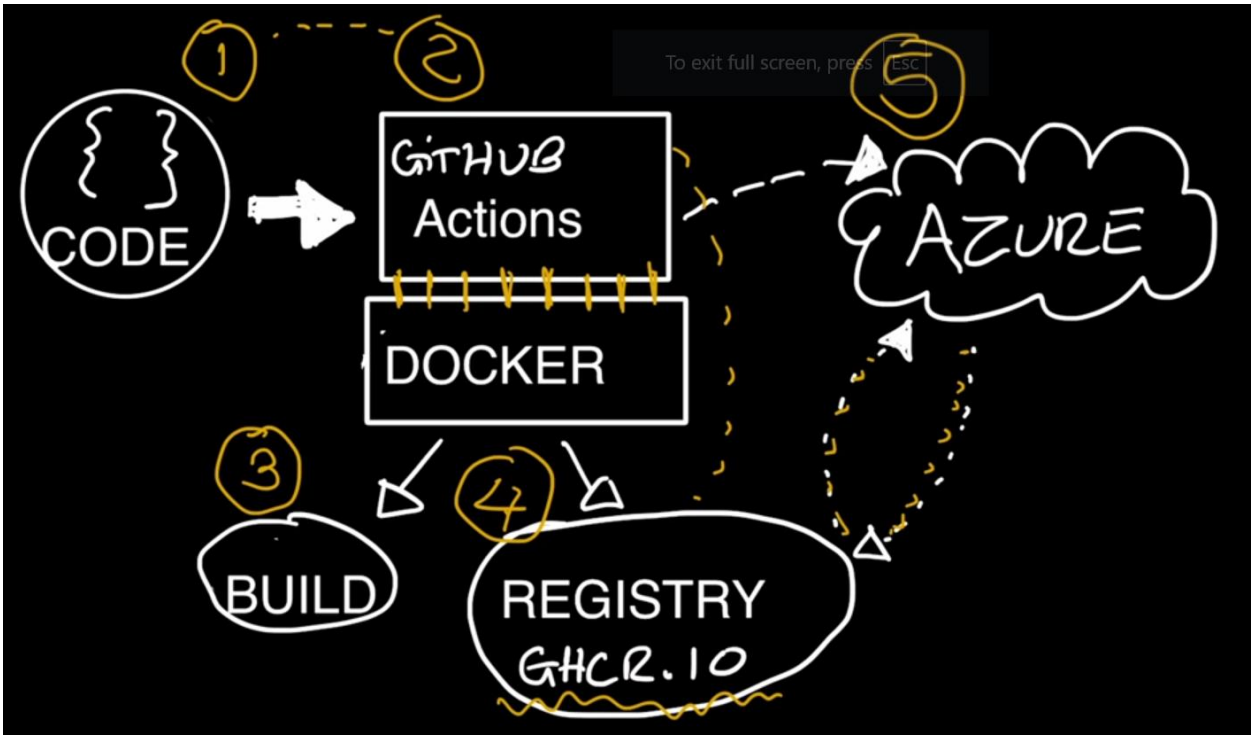
- **API-based application**: An application that interacts with other software components using Application Programming Interfaces (APIs) for data exchange, communication, or functionality enhancement.
- **Embedded model**: A machine learning model integrated within an application, allowing the app to perform specific tasks without relying on external services.
- **Multi-model application**: An AI system that utilizes multiple models tailored to different functions or domains, improving efficiency and performance in various scenarios.
- **HTTP API**: An interface for exchanging data between systems using Hypertext Transfer Protocol (HTTP) requests and responses over the internet.
- **Azure OpenAI**: A cloud-based platform by Microsoft that offers access to large language models through an easy-to-use API, enabling developers to build AI applications with advanced text generation capabilities.
- **Separation of concerns**: A design principle that suggests dividing complex systems into smaller, independent components responsible for specific functions, improving maintainability and scalability.
- **Scale/Scalability**: The ability of an application or system to handle increasing workloads efficiently by adding resources (e.g., computing power, storage) without significantly impacting performance or functionality.

- **Retrieval augmented generation (RAG)**: A technique in AI where a large language model accesses new or recent data outside its training set to provide better answers and improved results.
- **Vector database**: A search engine or database that stores vectorized documents, enabling more accurate information retrieval for AI models.
- **Embeddings**: Representations of text data as vectors in a high-dimensional space, allowing similarity comparisons between different pieces of text.
- **Azure AI Search**: Microsoft's cloud-based search service (formerly Azure Cognitive Services Search) that offers retrieval augmentation capabilities for large language models.
- **Comma separated value (CSV)**: A common data format where values are separated by commas, used in this transcript to demonstrate RAG implementation with a vector database.
- **Pandas library**: A Python library used for data manipulation and analysis, particularly useful when working with CSV files.
- **Qdrant:** Software used for creating an in-memory vector database search, enabling efficient text retrieval and embedding storage.
- **Sentence transformers**: A tool to encode sentences into numerical representations (embeddings) that can be compared using cosine similarity or other distance metrics.
- **Cosine distance**: A measure of similarity between two non-zero vectors in a multi-dimensional space, often used in text analysis and information retrieval.

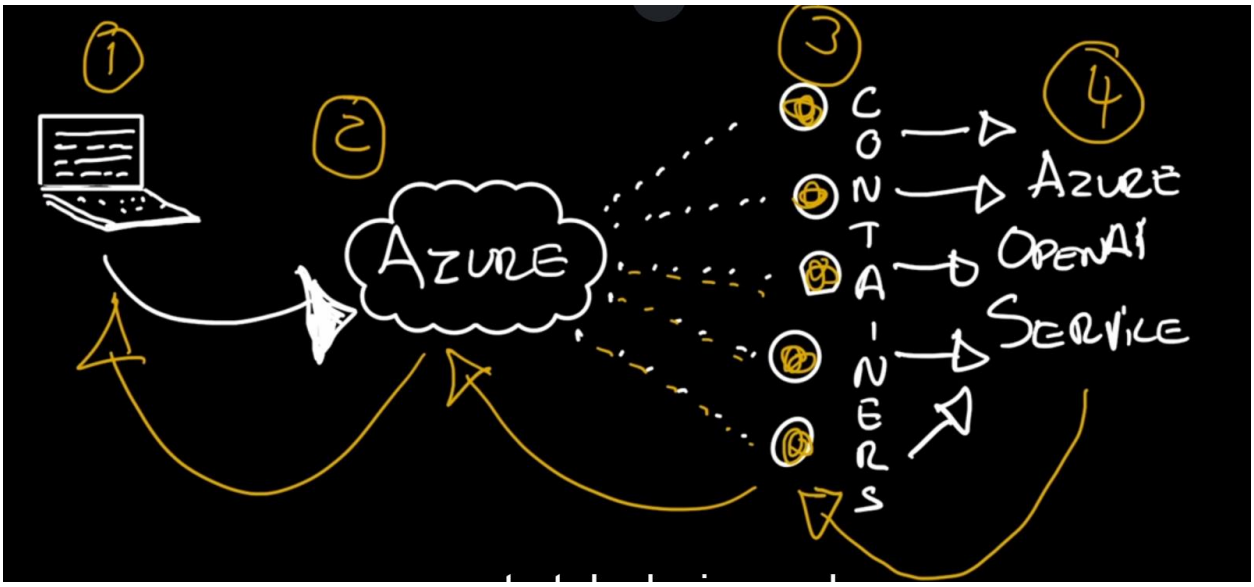
- **Cloud-based generative AI application**: An AI system that utilizes cloud services for deployment, often as an HTTP API, enabling accessibility and scalability.

- **Azure Cloud**: Microsoft's cloud platform that offers various services for implementing AI applications like RAG, using tools such as Azure OpenAI and Azure Cognitive Search.
- **Containerized application**: An application packaged with its dependencies and configurations in a container format (e.g., Docker), allowing easy deployment across different environments.
- **GitHub Actions**: A CI/CD automation tool provided by GitHub, enabling users to create workflows for building, testing, and deploying applications using containers or other methods.
- **Azure Container Registry (ACR)**: A cloud-based registry service in Azure that stores container images, allowing easy deployment of applications on the Azure platform.
- **Horizontal scaling:** An architectural approach to scale resources by adding more instances (e.g., virtual machines or containers) based on demand, ensuring high availability and performance.
- **Ingress:** A configuration setting for routing external traffic into containerized applications in Kubernetes or Azure Container Apps, specifying the target port inside the container.
- **Log stream**: Real-time monitoring of application logs to diagnose issues and understand system behavior, often used during development and deployment phases.



We have our application, it lives on GitHub.
We have an HTTP API.
It's going to receive some requests, it's going to interact with the model.
But how is that going to actually get deployed?
Well, we're going to start with the code.
I'm going to use GitHub actions, which is going to be our second step right here.
And GitHub actions is going to help us with the automation.

It is going to be interacting with docker.
So it's going to set up a docker build system and some commands, and
with that we're going to be able to build our container application.
Building a container application is an important step of what we're trying to do,
which is actually get these to be deployed on the cloud, on Azure.
So that process will build the container and
then we'll move over to our step number 4 right here,
which is to push the resulting image, the resulting container to the registry.
And in this case we're using GitHub container registry,
which does the GHCR.IO.
You can actually use other registries as well.
But in this case we're going to simplify because the username and password,
everything's already set up for us and we'll see those in details later when
we're looking where we're poking around the GitHub actions workflow.
Now, when the registry step is done,
we're going to go back into GitHub actions.
And then GitHub actions is going to say, hey, Azure, by the way,
you are ready to go to pull the resulting image from our registry.
So it's a triggering effect.
We're saying, hey, Azure, in this case want to use Azure container service,
and it's going to pull that container from the registry.
And that's where this interaction is going to happen here,
where things are going to go to the registry, from GitHub container registry,
the GHCR.IO, and going back to Azure, and then it's going to get published.



we are going to have a user.
When I start here, when I call this step 1,
a user is going to interact with our application.
It's going to send some prompts, it's going to send some queries,
and those are going to be received by Azure, the Azure cloud, and
in the cloud is where our application is going to live.
So Azure is going to be able to scale our application.
This is called a horizontal scale because more requests and

more handling of requests will allow us to create these little
blobs here that represent containers.
So the more we need, the more Azure can deploy and can expand.
So that expansion, that groove of containers is called
a horizontal scaling system where we're going to be able to process
all of these requests that are coming in, into our containers.
So once it gets to the container, as you know,
the application is going to be able to talk to.
So this would be a step 3 or containers.
It's going to be able to talk to Azure OpenAI service.
So once we're able to talk to Azure OpenAI service,
that request is going to get processed.
It's going to go to the containers which is going to back through Azure and
finally back to the user with some sort of a response.
So this should give you a new good solid idea on how this is going to
work beyond just looking at the application and the HTTP API and
the code that is actually putting all of these together.

Azure Container ->

- Ingress
- Custom Domains
- Log Stream
- Scale Replicas