# Architecture Design

**Real-time customer support chatbot**

**A diagram of a flowchart

Description automatically generated**

**CI/CD pipelines**

**A diagram of a system

Description automatically generated**

# Design Document

Scalable and Secure real-time customer support chatbot

## 1. Introduction

This document outlines the design of a robust and scalable real-time customer support chatbot service deployed on Azure Kubernetes Service (AKS). The service leverages OpenAI's powerful language models, Langchain for conversational AI orchestration, and a Vector Database FAISS for efficient semantic search. The architecture prioritizes scalability, reliability, security, and performance while adhering to 12-factor app principles and DevOps best practices.

## 2. Architectural Overview

The RAG service follows a microservices-based architecture deployed on Azure AKS, ensuring flexibility, scalability, and maintainability. The core components include:

* **FastAPI:** Provides a high-performance web framework for building the API endpoints, handling user requests and responses.
* **Langchain:** Orchestrates the conversational flow, interacts with OpenAI for language understanding and generation, and manages interactions with the Vector Database.
* **OpenAI API:** Provides embeddings for query understanding and powers the core language model for text generation.
* **Vector Database:** Stores and retrieves contextually relevant information based on semantic similarity search using vector embeddings.
* **Azure Kubernetes Service (AKS):** Manages the deployment, scaling, and orchestration of the microservices within containers.
* **GitHub Action:** Develop CI-CD pipeline and version control
* **Docker Hub:** Stores the Docker images for the application components, enabling efficient deployment and.
* **Azure Monitor:** Provides comprehensive monitoring, logging, and alerting capabilities for the entire system.

## 3. Key Design Considerations

**3.1 Scalability**

* **Horizontal Pod Autoscaling (HPA):** AKS automatically scales the number of pods for FastAPI, Langchain, and the Vector Database based on CPU utilization or other custom metrics, ensuring optimal resource utilization and responsiveness under varying loads.
* **Vector Database Scalability:** The chosen Vector Database FAISS should offer horizontal scalability to handle growing datasets and query volumes.
* **Stateless Design:** Microservices are designed to be stateless, allowing for easy scaling and failover without data loss.

**3.2 Reliability**

* **AKS High Availability:** AKS provides built-in high availability for the control plane and nodes, ensuring service uptime.

**3.3 Security**

* **API Authentication and Authorization:** Implement robust authentication (e.g., API keys, OAuth 2.0) and authorization mechanisms to control access to the RAG service and protect sensitive data.
* **Network Security:** Utilize Azure Virtual Network (VNet) and Network Security Groups (NSGs) to restrict traffic flow between components and isolate the AKS cluster.
* **Secrets Management:** Securely store sensitive information, such as API keys and database credentials, using Azure Key Vault and integrate it with AKS for secure access.

**3.4 Performance**

* **FastAPI Asynchronous Operations:** Leverage FastAPI's asynchronous capabilities to handle concurrent requests efficiently, minimizing latency.
* **Caching:** Implement caching mechanisms using Redis at various levels (e.g., API Gateway, Retrieval ) to reduce response times for frequently accessed data.
* **Optimized Vector Database:** Choose a Vector Database with high-performance indexing and query capabilities to ensure fast retrieval of relevant information.

## 4. 12-Factor App Principles

The RAG service adheres to the 12-factor app methodology, promoting best practices for building scalable and maintainable applications:

* **Codebase:** Version-controlled codebase with a single deployment for consistency.
* **Dependencies:** Explicitly declare and isolate dependencies.
* **Config:** Store configuration in the environment.
* **Backing Services:** Treat backing services as attached resources.
* **Build, Release, Run:** Separate stages for building, releasing, and running the application.
* **Processes:** Execute the app as one or more stateless processes.
* **Port Binding:** Expose services via port binding.
* **Concurrency:** Scale out using the process model.
* **Disposability:** Maximize robustness with fast startup and graceful shutdown.
* **Dev/Prod Parity:** Keep development, staging, and production environments as similar as possible.
* **Logs:** Treat logs as event streams.
* **Admin Processes:** Run administrative/management tasks as one-off processes.

## 5. DevOps Practices

* **CI/CD:** Implement a robust CI/CD pipeline using GitHub Actions or Azure DevOps to automate the build, testing, and deployment process, ensuring rapid and reliable releases.
* **Monitoring and Alerting:** Utilize Azure Monitor to collect metrics, logs, and set up alerts for performance bottlenecks, errors, and security events.
* **Automated Testing:** Implement unit, integration, and end-to-end tests to ensure code quality and catch regressions early in the development cycle using pytest.