ST10092354
Lerato Kekana
ICE Task 4

An Azure Function is a server less compute service that lets you run event-driven code without having to manage the infrastructure. You write your code and specify the triggers (events) that will execute your function, such as an HTTP request, a message on a queue, a timer, or changes in a database. Azure functions automatically scale based on demand, making it cost-effective and ideal for processing small units of work or micro services.

Why Use Azure Functions?

Serverless Architecture: No need to manage servers, reducing maintenance overhead.
Event-Driven: Perfect for executing code in response to events such as HTTP requests, timers, or database changes.
Scalability: Functions automatically scale to handle spikes in demand, allowing you to respond to workloads seamlessly.
Cost Efficiency: You only pay for the execution time, making it a great choice for low-demand scenarios.

Examples of Usage:

Sending automated emails after a user places an order.
Processing payment transactions.
Running scheduled tasks like daily reports or data cleanups.

In-Process Model vs. Isolated Worker Model

Azure Functions offer two hosting models for .NET-based functions: the In-Process Model and the Isolated Worker Model. Here's a breakdown of the differences:

1. In-Process Model

Execution Context: Runs within the same process as the Azure Functions runtime.
Tight Integration: The function is tightly integrated with the runtime, offering better performance and faster execution times.
Dependency Management: Shares dependencies with the Azure Functions runtime, allowing easy access to built-in bindings (like HTTP, Queue, Blob, etc.).
Use Case: Ideal for most standard scenarios where tight integration and quick responses are necessary. Supports .NET Core 3.1 and .NET 6.

2. Isolated Worker Model

Execution Context: Runs in a separate process from the Azure Functions runtime.
Loose Integration: Provides more control over dependencies and versioning, making it suitable for scenarios where isolation and custom dependency management are essential.
Customisation: Allows you to use middleware, custom logging, and additional configuration without affecting the runtime.
Use Case: Preferred when you need complete control over dependency isolation, middleware, and cross-cutting concerns. Supports .NET 5 and later versions.

Summary of differences:

In-Process: Tightly coupled with the runtime for faster execution and simpler development but limited flexibility over dependencies.
Isolated Worker: Runs separately from the runtime, offering greater control and flexibility at the cost of slightly higher complexity and possibly slower cold starts.