**Server vs Serverless**

To deploy any application, we need servers that provide compute power (CPU/RAM), storage, and networking. For instance, even a web browser like Google Chrome relies on backend servers with sufficient computing resources to handle user requests.

If we utilize 100% of a server’s compute capacity (CPU/RAM), the server can become overwhelmed and start to choke. A common example is when Osmania University releases exam results—the sudden spike in traffic causes the website to crash due to excessive load on the server. Similarly, during major online sales, e-commerce platforms often experience downtimes or degraded performance because of the unexpected surge in user requests. These scenarios happen because the existing servers can't scale up instantly to handle the increased demand.

When managing physical or virtual servers manually, it's the responsibility of the administrator to constantly monitor the load and adjust. This can be a tedious and time-consuming task.

Thankfully, platforms like **Microsoft Azure** provide a **Monitoring Dashboard**—known as **Azure Dashboard**—that helps track server performance in real-time. Just like Task Manager in Windows shows CPU, memory, disk, and GPU usage, the Azure Dashboard gives a comprehensive view of your cloud resources. This makes it easier to identify performance bottlenecks and act before a system failure occurs.

Cloud provider saw this gap and they came up with serverless concept:

**Azure VM as a Server:**

When you use an **Azure Virtual Machine (VM)** to host a web application, you’re responsible for configuring the infrastructure—such as the amount of RAM and CPU. For example, you might deploy your app on a VM with **8 GB RAM and 64-core CPU**.  
However, you must **manually adjust** these resources based on the application's usage. If the traffic suddenly spikes, and your current configuration can't handle it, the server may slow down or crash unless you scale it yourself.

**Azure Functions (Serverless):**

Azure Function is a serverless service that runs your code in response to events, like an order placed, and automatically scales based on demand — you only pay for what you use."

Azure Function is a PaaS **(Platform as a service)** because it provides a runtime environment and automatic scaling.

Azure Functions allow you to run your web application code without worrying about managing the infrastructure. You simply provide the code or a Docker file, and Azure takes care of the rest.

* By default, Azure Functions run on a 2-node base configuration:
  + Each node has 8 GB RAM
  + Combined, the base gives you 16 GB RAM and access to 32 CPU cores
* This setup can handle around 1 million (1M) requests comfortably.
* Now imagine it's Black Friday, and you suddenly get 1 billion (1B) requests.  
  Azure Function will automatically scale out by adding more instances (nodes) to handle the load. This is called horizontal scaling—a commonly asked interview question.
* Even though Azure Function scales horizontally, it acts like one VM to you, running your code consistently while Azure manages the scaling in the background.

**📦 Use Case: Order Confirmation in E-Commerce**

Imagine you're running an **online store** (like Amazon or Flipkart). When a user places an order, several small tasks need to happen:

1. Send an **order confirmation email**
2. Update the **inventory**
3. Generate an **invoice**
4. Notify the **warehouse** for shipping

Instead of making your main app do everything (which can make it slow), you can give each small task to Azure Functions, so your app runs faster and smoother.

* When an order is placed, it **triggers** these Azure Functions.
* Each function does its job **independently** and **automatically scales** based on how many orders you’re getting.
* You only **pay for the time** the function runs, making it **cost-effective**.

**🔁 Quick Flow Example:**

1. **User places an order** →
2. **Trigger**: Azure Function is triggered
3. **Function 1**: Sends confirmation email
4. **Function 2**: Updates inventory
5. **Function 3**: Notifies shipping department
6. All done — and you **only pay for the actual execution time**

**📱 Use Case: Processing Uploaded Images in a Photo-Sharing App**

Imagine a photo-sharing app like Instagram or Snapchat where users upload images.

**What needs to happen when a user uploads a photo?**

* Resize the image to different sizes (thumbnail, medium, large)
* Apply filters or watermarks
* Generate metadata (like date, location)
* Store processed images in cloud storage

**How Azure Functions help here:**

* When a user uploads an image, **Azure Function is triggered automatically**.
* It processes the image (resizing, filtering) **without slowing down the main app**.
* If many users upload photos at the same time, Azure Functions **scale out automatically** to handle the load.
* You only pay for the actual time spent processing images.

**What are Triggers in Serverless Functions?**

Think of a trigger like a “starter button” that tells your function, **“Hey, do your job now!”**

**Blob Trigger**

**What it does:**  
Runs your function when a new file (image, document, etc.) is uploaded or changed in Azure Blob Storage.

**Example (Instagram-style):**  
You upload a photo → a function is triggered to:

* Create smaller image sizes (thumbnails)
* Add filters or watermarks
* Scan for inappropriate content

**Queue Trigger**

**What it does:**  
Runs your function when a message is added to a queue (like a to-do list).

**Example:**  
When someone places an order, a message goes into a queue → function is triggered to:

* Update stock
* Send confirmation email
* Prepare invoice

**Use case:**

* Background task processing
* Order handling
* Email notifications

**Timer Trigger**

**What it does:**  
Runs your function on a schedule (like an alarm or reminder).

**Example:**  
Every day at midnight → function runs to:

* Clean up old images
* Send daily report emails
* Refresh data in a database

**Use case:**

* Scheduled backups
* Data cleanup
* Automated reporting

Azure functions = AWS Lambda = Goggle Cloud function

Types of Scaling

* **Horizontal** **Scaling** (scaling out): Is a method of increasing system capacity by adding more machines or servers to a resource pool. Like adding more lanes to a highway to handle increased traffic.
* **Vertical** **Scaling** (scaling up): Is a method of increasing the capacity of a single server by adding more resources like CPU, memory, or storage.

|  |  |  |
| --- | --- | --- |
| Aspect | Horizontal Scaling | Vertical Scaling |
| Failure impact | Low (other servers can work) | High (if one fails, all fails) |
| Cost efficiency | May waste resources when traffic is low | Efficient but limited |
| Scalability | Nearly unlimited | Limited to machine capacity |
| Setup complexity | More complex | Easier |

**Microservice vs Monolith application**

Let’s take Amazon as an example. It has different modules like:

* Login Page
* Product Service
* Amazon Pay
* Order Management

In a **monolithic application**, all these features are part of **one large codebase** and are **hosted on a single virtual machine**. This means if that one VM crashes, the entire application could go down — affecting all features.

But this is not ideal, especially for large-scale applications.

Now, imagine each feature is built and deployed as a **separate microservice** — for example:

* Login service on one VM
* Product catalogue on another
* Payment system on a third
* Order tracking on a fourth

Each microservice runs **independently**, and even if one service (say, Amazon Pay) goes down, the rest (like browsing products or logging in) can still function.

Even though cloud platforms handle failure more gracefully today, this example helps illustrate **why companies are moving toward microservices**.

In a microservices architecture, services often need to **communicate with each other**. One common method is through **message queues**. This is especially useful for decoupling services and handling communication **asynchronously**.

**🔁 How it works:**

* One microservice acts as a **Publisher (Producer)**:  
  It sends messages (events or data) to a **message queue**.
* Other microservices act as **Subscribers (Consumers)**:  
  They **listen to the queue** and process messages whenever they arrive.
* There can be **multiple subscribers** to the same queue.
* You can also create **multiple queues** for different types of messages or services.

**Azure Function (Queue Trigger)**:  
One person is given a task (e.g., send an email) when a sticky note is dropped in a box.

**Microservices with Queues**:  
A group of people — inventory, shipping, billing — all read their own sticky notes from the same box and act on them.