**⚙️ Serverless in Kubernetes + Key Differences from Traditional Servers**

**🚀 What is Serverless in Kubernetes?**

**Serverless in Kubernetes** means running code without managing the underlying infrastructure — **you focus only on writing functions or services**, and the platform takes care of **scaling, scheduling, and resource provisioning**. In Kubernetes, this is achieved using frameworks like:

* **Knative** (most popular)
* **OpenFaaS**
* **Kubeless**

These tools allow developers to deploy serverless workloads on Kubernetes by abstracting away pods, services, and scaling logic. Your code runs **only when needed**, and Kubernetes dynamically allocates resources, scales the app, and cleans up when idle.

**📊 Serverless vs Traditional Servers — Summary from the Image**

| **Aspect** | **Traditional Servers** | **Serverless** |
| --- | --- | --- |
| **Server Management** | Requires manual provisioning, config, maintenance. | No need to manage servers; the cloud provider or K8s manages it. |
| **Scalability** | Manual scaling, may cause downtime or delays. | Auto-scaling on demand, often seamless. |
| **Cost Model** | Pay for servers whether used or not. | Pay only for compute while code runs. |
| **Responsibility** | You handle security, patching, upkeep. | Provider shares responsibility; focus is on app logic. |
| **Complexity** | Requires infra/server expertise. | Simplifies development via abstraction. |
| **Development Flexibility** | Full control over stack. | Focus on app code; infrastructure is abstracted. |
| **Deployment Time** | Time-consuming setup/config. | Quick deployment; faster time-to-market. |
| **Usage-Based Billing** | No, pay regardless of usage. | Yes, based on actual resource use. |
| **Use Cases** | Best for stable, predictable workloads. | Ideal for event-driven, real-time, or sporadic tasks. |