
Serverless Computing

Introduction

- Serverless Computing (or simply serverless) is emerging as a new and compelling model for the deployment of cloud applications

Server less architecture relies on managed services where you only pay for what you use, without worrying about infrastructure

Introduction

There are many immediate benefits to not managing your own servers:

- You don't have to worry about them randomly rebooting or going down.
- You don't end up with snowflake servers, where you don't know quite what's installed on them but they are mission-critical to your organisation.
- You're not responsible for installing software on them. Even if you use configuration management tools such as Chef or Ansible to automate this, that's still extra code you have to maintain over time.

What is Serverless Computing?

- Serverless Computing is a cloud computing execution model in which the cloud provider dynamically manages the allocation of machine resources, and bills based on the actual amount of resources consumed by an application, rather than billing based on pre-purchased units of capacity..

- The version of serverless that explicitly uses functions as the deployment unit is also called Function-as-a-Service (FaaS).

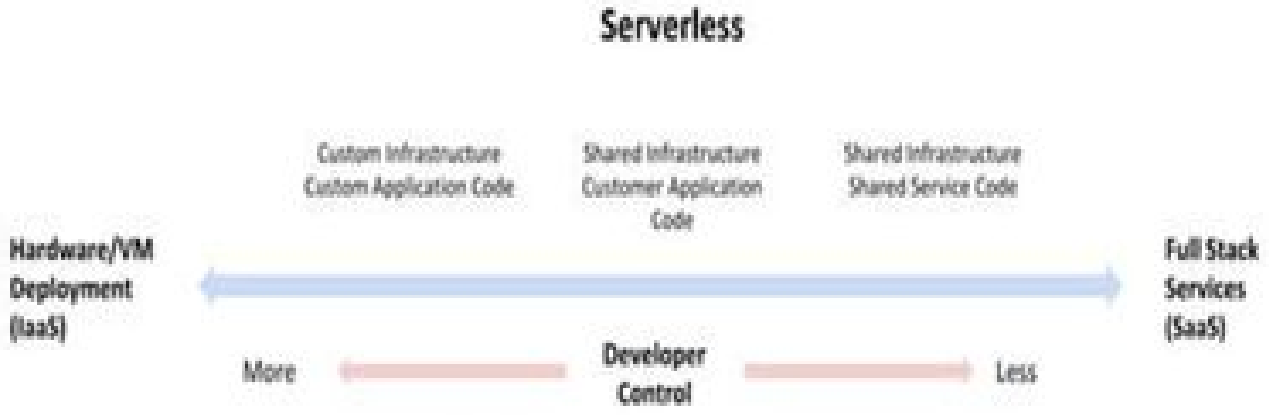
Feature	Server-Based	Serverless
Base Cost	$20–$50/month per server	Free for first 1M requests/month
Scaling Cost	Adds more servers, increases cost	Auto-scaling, pay-as-you-go
Maintenance	Manual (OS updates, patches)	Managed by the platform
Setup Time	Hours to days	Minutes
Flexibility	High customization	Limited customization
Application	high-customization, long-running applications	event-driven, cost-sensitive, and scalable applications

What is Serverless Computing?

- The Infrastructure-as-a-Service (IaaS) model is where the developer has the most control over both the application code and operating infrastructure in the cloud
- The developer is responsible for provisioning the hardware or virtual machines.
- Can customize every aspect of how an application gets deployed and executed.
- On the opposite extreme are the PaaS and SaaS models, where the developer is unaware of any infrastructure.
- The developer has access to prepackaged components or full applications. The developer is allowed to host code here, though that code may be tightly coupled to the platform.

Serverless?

- Serverless can be explained by varying level of developer control over the cloud infrastructure.



Serverless : Characteristics

- Independent, server-side, logical functions : small, separate, units of logic that take input arguments, process them in some manner, then return the result.
- Cost : Typically its Pay As You Go
- Simple Deployment : Thanks to the small size of deployment artifacts, in general, deployments are simple and quick. Deployment artifacts are typically idiomatic of the chosen runtime e.g. NuGet packages, npm packages, JAR files
- Ephemeral : designed to spin up quickly, do their work and then shut down again.
- Programming languages : Serverless services support a wide variety of programming languages - Node, Python.
- Stateless : FaaS are stateless, not storing states ,as containers running code will automatically destroy and created by platform.Horizontal Scaling becomes easy...

Serverless : Characteristics

- Scalable by Default
- Event Triggered :Although functions can be invoked directly, they are typically triggered by events from other cloud services, such as incoming HTTP requests,
- Simple Deployment Model.
- Small Deployable Units and More focus on Business Value.
- Managed by third party .
- No more "Works on my Machine"

Commercial platforms

- Amazon's AWS Lambda
- Google's Cloud Functions
- Microsoft Azure Functions
- IBM Cloud Functions
- OpenLambda

Commercial platforms

- Amazon's AWS Lambda
- Google's Cloud Functions
- Microsoft Azure Functions
- IBM Cloud Functions
- OpenLambda

Amazon's AWS Lambda

- Amazon's AWS Lambda was the first serverless platform ,it is a compute service that lets you run code without provisioning or managing servers."
- AWS Lambda executes code only when needed and scales automatically, from a few requests per day to thousands per second.
- Pay only for the compute time.
- Can run code for virtually any type of application or backend service

Amazon's AWS Lambda

- Currently AWS Lambda supports Node.js, Java, C# , Go and Python and PowerShell
- AWS Lambda automatically scales application by running code in response to each trigger.
- With AWS Lambda,we are charged for every *100ms*

Google's Cloud Functions

- Google Cloud Functions provides basic FaaS functionality to run serverless functions written in Node js , Go, Python and Java.
- Automatically scales, highly available and fault tolerant.
- No servers to provision, manage, or upgrade
- Pay only while your code runs.

AIMDek Technologies

Microsoft Azure Functions

- Microsoft Azure Functions provides HTTP webhooks and integration with Azure services to run user provided functions.
- The platform supports C#, F#, Node.js, Python, java and PowerShell.
- Pay only for the time spent running your code with Consumption plan.
- The runtime code is open-source and available on GitHub under an MIT License.

Microsoft Azure

AWS Lambda, Azure Functions, and Google Cloud Functions are the primary Function-as-a-Service (FaaS) offerings from the top three cloud providers. While they share core serverless principles (automatic scaling, pay-per-use, no server management), they differ significantly in ecosystem integration, developer experience, and specific technical features.

Feature	AWS Lambda	Azure Functions	Google Cloud Functions
Primary Strength	Deepest ecosystem integration, mature tooling, performance optimization	Strong enterprise, hybrid cloud, and .NET integration, stateful workflows via Durable Functions	Simplicity, rapid deployment, designed for real-time data processing and mobile backends
Max Execution Time	15 minutes	10 minutes (Consumption Plan), up to unlimited (Premium Plan)	9 minutes (Gen 1), up to 60 minutes (Gen 2 via Cloud Run)
HTTP Endpoints	Requires a separate service (API Gateway or ELB)	Native HTTP triggers built-in	Automatic HTTPS endpoints built-in
Ideal User	Teams already heavily invested in the AWS ecosystem needing maximum control and broad integrations	Microsoft-centric organizations needing strong identity management and hybrid capabilities	Teams prioritizing simplicity, quick deployments, and seamless integration with GCP or Firebase

OpenLambda

- OpenLambda is an open-source serverless computing platform. The source-code is available in GitHub under an Apache License.
- The Lambda model allows developers to specify functions that run in response to various events.
- OpenLambda will consist of a number of subsystems that will coordinate to run Lambda handlers:

Benefits

- Compared to IaaS platforms, serverless architectures offer different tradeoffs in terms of control, cost, and flexibility.
- The serverless paradigm has advantages for both consumers and providers.
- From the consumer perspective, a cloud developer no longer needs to provision and manage servers, VMs, or containers as the basic computational building block for offering distributed services.
- The stateless programming model gives the provider more control over the software stack, allowing them to, among other things, more transparently deliver security patches and optimize the platform.

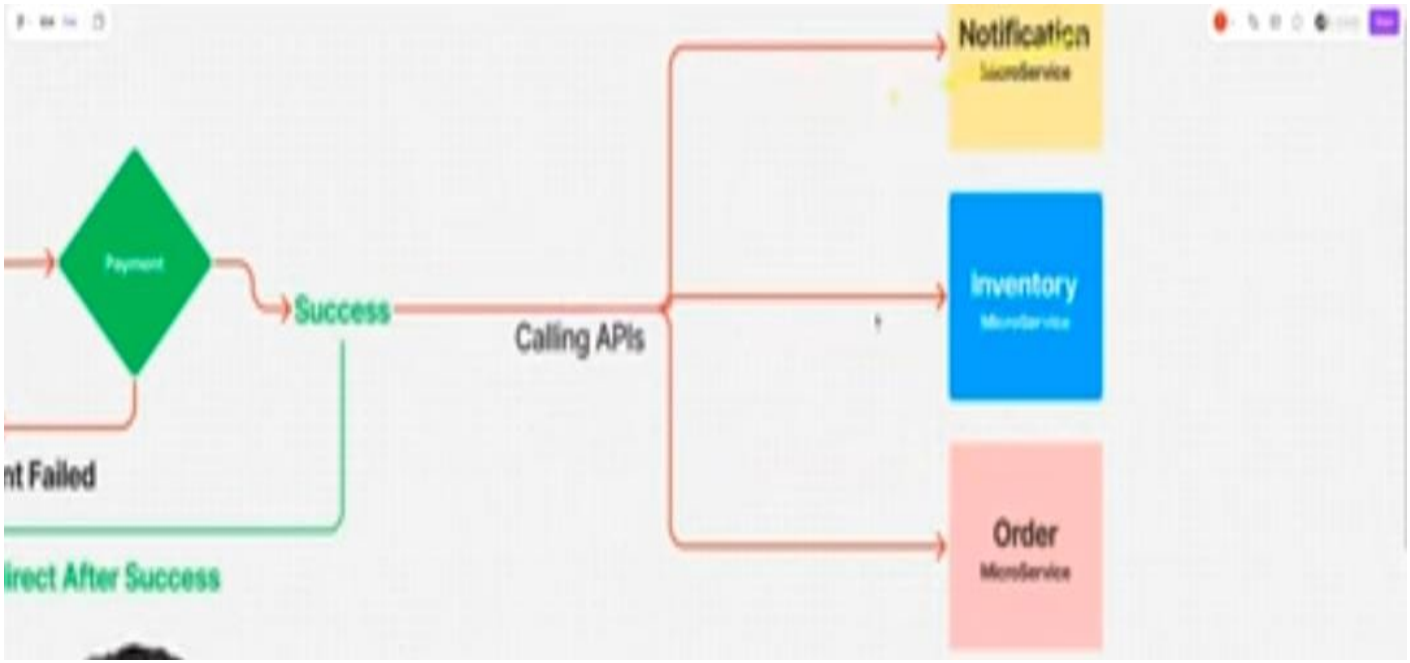
AIMDek Technologies

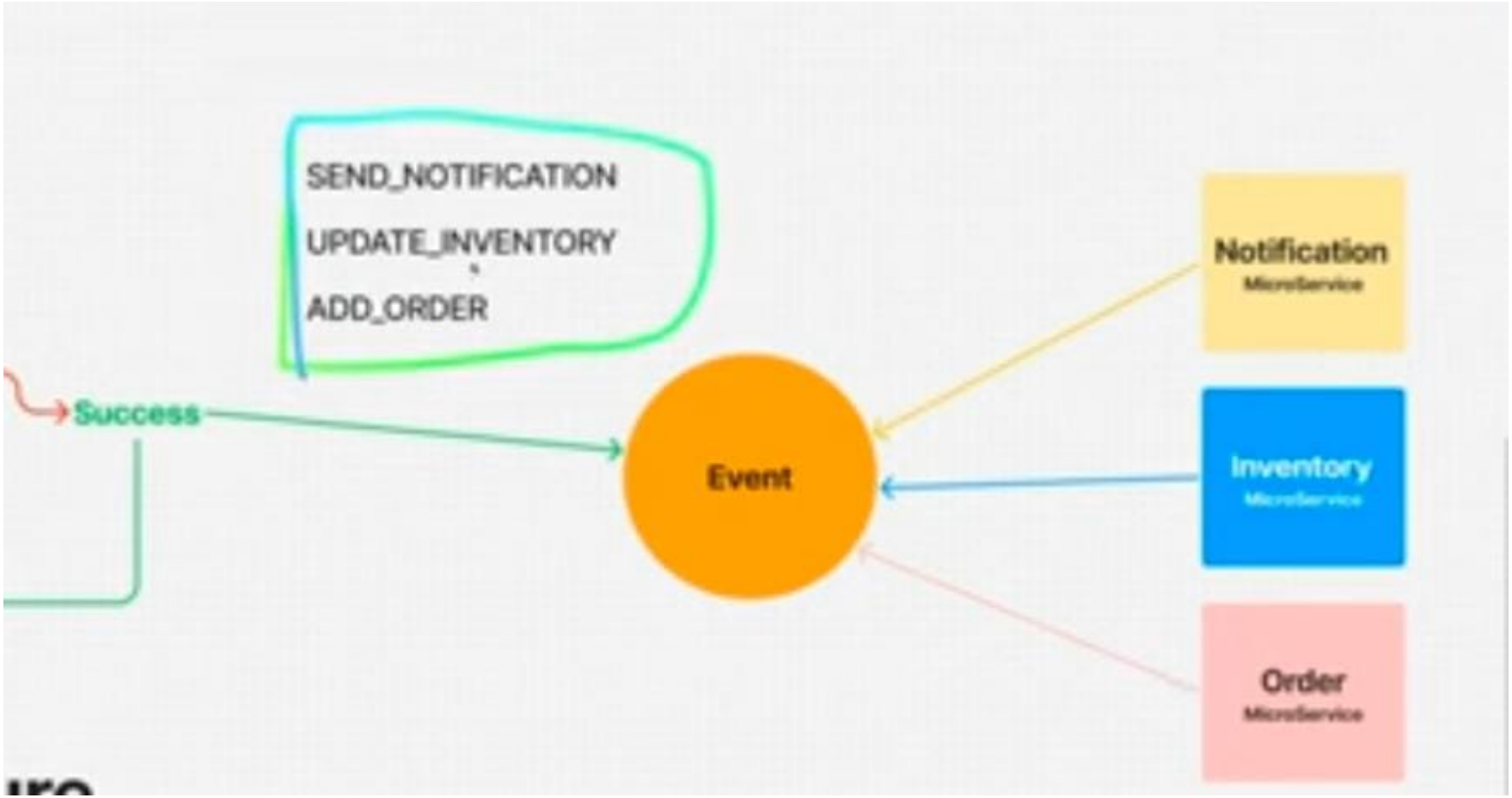
Current state of serverless platforms

- There are many similarities between serverless platforms.
- They share similar pricing, deployment,and programming models.
- Current serverless platforms only make it easy to use the services in their own ecosystem.
- Open source solutions may work well across multiple cloud platforms.

Event-driven computing

Event-driven computing is a software paradigm where system actions are triggered by events—significant changes in state like user clicks, sensor inputs, or system updates—rather than sequential procedural code. It enables real-time, asynchronous processing, improving responsiveness and scalability, particularly in microservices, GUI applications, and IoT, allowing components to communicate without direct, rigid coupling.


Notification
Inventory
Order
Payment
Success
Calling APIs
nt Failed
irect After Success


SEND_NOTIFICATION
UPDATE_INVENTORY
ADD_ORDER
Success
Event
Notification
MicroService
Inventory
MicroService
Order
MicroService

Tightly Coupled (High Dependency)

- **Definition:** Components are deeply dependent on one another; a change in one often requires changes in others.
- **Benefits:** Can offer higher performance, tighter security, and easier, faster communication between modules.
- **Drawbacks:** Difficult to maintain, modify, or scale; changes can cause ripple effects.
- **Example:** A monolith application where the database schema is directly embedded within the user interface code.

Key Differences Summary

- **Interdependence:** High in tight, low in loose.
- **Maintainability:** Easier in loose, harder in tight.
- **Flexibility:** High in loose, low in tight.
- **Scalability:** Better in loose.

Loosely Coupled (Low Dependency)

- **Definition:** Components operate independently, with minimal knowledge of each other's internal workings.
- **Benefits:** High flexibility, easy to test, swap, and reuse components.
- **Use Cases:** Modern microservices, distributed systems, and scalable, maintainable architectures.
- **Example:** A plug-in architecture, where modules can be added or removed without changing the core application.