#ASLI ENGINEERING

Introduction to Serverless Computing

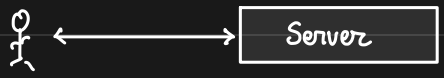
SWIPE

BY

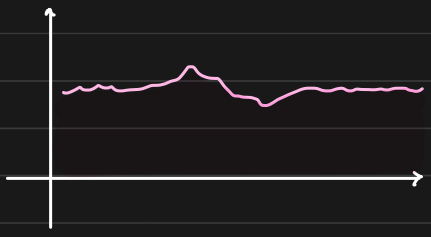
ARPIT BHAYANI

Serverless Computing

Traditional way of computing

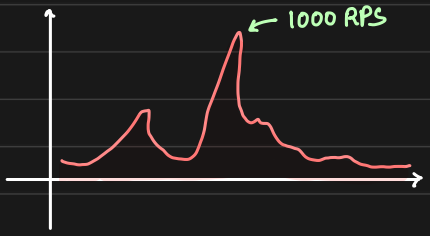


APIs are hosted on a server that is continuously running expecting a continuous traction



eg: a 4GB RAM 2-core machine running to handle 1000 RPS

But what if, your traffic pattern is like this



Instead of continuous your request pattern is bursty.

But you need to be prepared to handle the peak.

Your infra is provisioned to handle the peak of 1000 RPS but is overprovisioned for 99% of the times

Serverless Computing

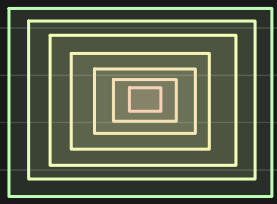
So, what if there is some Infrastructure as a Service, that

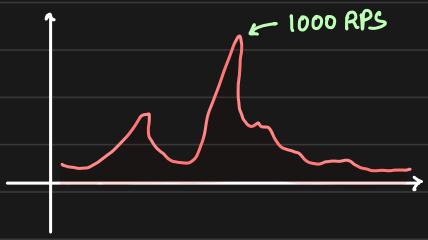
- ↳ scales up and down as per traffic
- ↳ is billed per execution
- ↳ is self-maintained and fault-tolerant

Serverless does NOT mean that server is not required to execute the code

It solely means...

you are not managing and scaling the server





With serverless you are billed per request (execution) and hence you can seamlessly handle the desired scale with zero to no worry.

With serverless computing, engineers can solely focus on their

Core Problem

All major cloud providers have one serverless offering

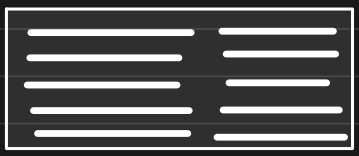
- AWS : Lambda Functions
- GCP : Google cloud functions
- Cloudflare : Cloudflare workers
- Azure : Azure Serverless Functions

Real world applications of Serverless

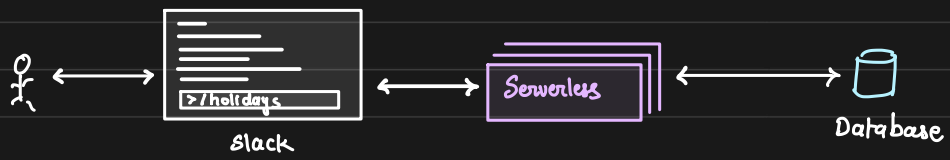
1. Chatbot

Internal chatbot to respond to requests

> /holidays



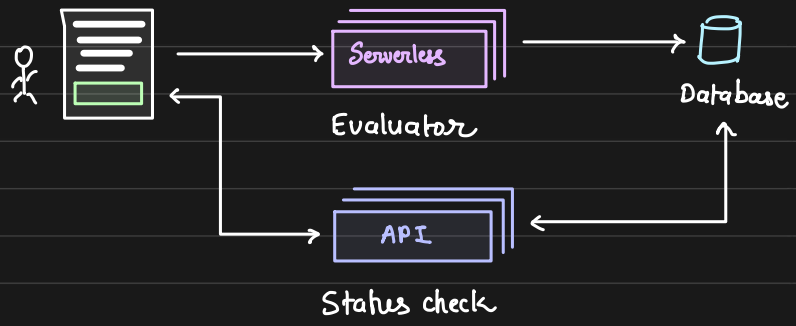
The requests coming to an internal chatbot is very bursty, so a serverless arch is ideal for this usecase



2. Online Judge

Triggering automatic code evaluation upon a user submission is best done serverless

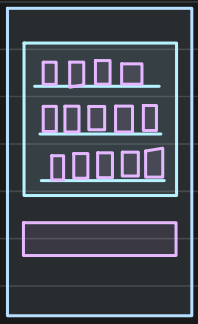
- time bound
- short lived
- bursty traffic



3. Vending Machines

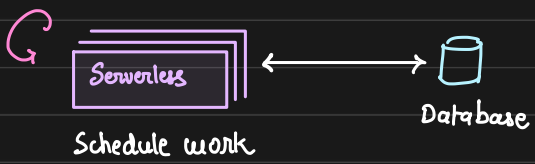
Upon purchase vending machine can invoke a serverless function to update the inventory.

Because traffic is very fluctuating, using serverless here saves a bunch on money.



4. Scheduled DB backup

Trigger a database backup every day at 4pm



Triggering a DB backup is just an API call because DB is managed

5. Batch / Stream processing

Trigger a serverless function as soon as a msg comes in your broker.

* Making event consumption reactive

No need to run consumers continuously

Advantages of Serverless Computing

1. No server management and maintenance

   Servers are managed by the vendors (AWS, GCP, etc) and engineers need not worry. So no DevOps, access control, security patches, etc

2. Pay-as-you-go Pricing

   Charged only for what you use. So ideal for bursty traffic where you do not need to constantly keep your server running & pay full price

   1 ¢ per 100ms of execution

3. No Capacity Planning

   Provisioning is on-demand, precise and realtime, so, engineers do not need to plan capacity for the incoming traffic

   Upon the burst, the function will automatically scale horizontally.

   Traditional arch overwhelms when it sees a surge of requests

Disadvantages of Serverless Computing

No Silver Bullet

1. Cold Start Problem

Because the function is not constantly running, the container may need to 'boot' up upon 1st invocation and hence time to serve the 1st request might be unnecessarily very high

2. Not built for long-running processes

Serverless execution has a time limit (say 15 min) which means you cannot deploy a logic that requires > 15 mins for its execution

eg: Huge map-reduce job is not for Serverless a traditional server is a better choice

3. Testing and debugging is tough

logging is not straightforward. no visibility to 'server' to check what's happening

4. Vendor lock-in

Hard to move from one vendor to another

So, when should you NOT use serverless?

1. load is almost constant and predictable
2. long running process and execution
3. You need multi-tenancy
4. You want to use serverless, just because it is cool.

So, when should you use serverless?

1. Quick build, prototype, test and deploy
2. use case is small and light weight
3. traffic is bursty ... Peak to 0 alternates