Winning with JavaScript in Serverless Apps

Priyatham (Pri)

Outline

- What and What not
- The Execution Model Spectrum: Isolates vs. VMs vs. Containers.
- JavaScript and runtimes
- The "Cold Start" Reality
- Warmup ways
- Optimize: Code & Cache
- Future

Unix Philosphy

Serverless

Do one thing and do it well !

Serverless can learn a thing or two from it, in trying to break down big functions into small functions that do one thing and do it really really well, multiple times

./images/serverless-cat-meme.jpg

When NOT to use serverless ?

The mindset

- Does your code run for more than 15 minutes ?
- Any socket or voice chat ?
- Too much database stuff ?
- Huge dependencies ?
- does it have a binary or docker ?

Execution model

Execution Models

V8 isolates

A V8 isolate is not a process and not a thread. It is a self-contained instance of the V8 JavaScript engine with its own heap, garbage collector, and compilation pipeline. Example: Cloudfare Workers, Deno deploy

Execution Models

Micro-VM

A kernel-based Virtual Machine, lighter but still has the same advantages of isolation, security and VM like OS support. Example: AWS Lambda

Execution Models

Container

Full container with a Node.js environment, entirely proxied. Startup and management are the issues

Edgy vs Trad

Edgy

Edge functions

Edge functions are run at the edge, close to your user whichever server it is. Less support of libraries, needs to be simpler and pretty fast

Trad Functions

Trad(itional) serverless functions

Run at the server where your cloud provider allocates the serverless functions. More support for libraries, could be complex, a little slower

JavaScript

Can I do it without JavaScript ?

Yes, but that's not fun

Use what you like, you are the expert

Event-Driven Architecture

Event-driven

Don't send API, only clicks and boops

Instead of APIs, PUT, GET or POST to serverless

Send every click to a queue and let serverless do them parallel

Event-driven

Javascript is made for Event-Driven

Call Stack, Task queue & Event loop

Events are stacked and queued, executed to the event loop

Serverless requirements

Requirements for serverless

- Have a config file
- The function file needs to export a function named **handler**
- The function needs to return an object with a **statusCode** matching a valid HTTP response code
- The response object also needs to include a **body** value, which is plain text by default

The config

Serverless.yml

The configuration file for the function

This also gives you subdomains that you can use for many other things.

The better way

Serverless.yml

```
service: my-service

provider:
  name: aws
  runtime: nodejs14.x
  stage: dev
  region: us-east-1

functions:
  hello:
    handler: handler.hello
```

The function

The Node.js file

The Node.js file with the function

How it works an example

Function

Node.js function

```
        // Simple function hello in handler.js
exports.hello = (event) => {
  const response = {
    statusCode: 200,
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ message: 'Hello from serverless!',
    input: event })
  };
};
```

JavaScript Runtimes

Different kinds

Runtimes

You can have the classic Node.js, Workers, Deno

The open source workerd

Comparison matrix

Runtime Matrix

Every runtime has its own things with varying compatability to Node.js API

This matrix shows what's supported

Cold Start

Serverless

More time to start the function

Delay experienced when a function is invoked for the first time after a period of inactivity, or when the cloud provider needs to provision a new execution environment to handle increased demand.

Cold start

How to measure it?

Using monitoring tools like CloudWatch, Insights based on the provider

What happens in a cold start

- Container Initialization: execution env or container
- Runtime Loading: Node.js/bun runtime loaded.
- Code Fetching: Fetching your precious code
- Initialization Logic: Any code defined outside the main function handler is executed.

Warmer

Cold start

Warmer Function/Ping the function

After measuring when cold starts, occurs have a helper serverless function timed to keep it warm, or a cron job like automation.

Optimize Optimize Optimize

Better warmup

Warmup improvements

- Provisioned Concurrency: pre-configure how many functions you want
- Minimum Instances GCP: Define minimum number of instances to be alive at any time
- Code Fetching: Fetching your precious code
- Get premium Azure: gives always ready instances

Code optimization

- Minimizing Package Size: you know this, but you can hear it twice
- Lazy Loading Dependencies: You also know this, and you also can hear it twice
- Init loading: Code outside the main handler function runs during the cold start

Cache

You could cache it

In-memory or use external cache like redis to reduce fetching from a DB everytime

Init different ways

Bad way

```
export async function handler(req) {

const db = new Database();

return db.query();
}
```

Init different ways

Good but suspicious

```
const db = new Database();

export async function handler(req) {

return db.query();

}
```

Init different ways

Better way

```
async function setup() { ... }

let initialized = false;
export default { async fetch(req) {

if (!initialized) { await setup();

initialized = true; } ...

}
}
```

Future

Hopeful Future

- Improved Container Reuse and Runtime Optimizations
- Containerized Serverless: Lambda container images
- WebAssembly (Wasm) runtimes: Spin framework
- Edge functions
- Standarized serverless interfaces: Hopefully open-source ones
-

Conclusion

- What is and why serverless
- Execution Models
- Edge functions
- Cold-Start
- Javascript & Node.js
- Runtime compatibilities
- Future

References

- AWS blogs
- Azure blogs
- Cold vs Hot Mikhail Shilkov
- Spin blogs
- Open-source FaaS: breakp

Thank you & Questions !

feedback

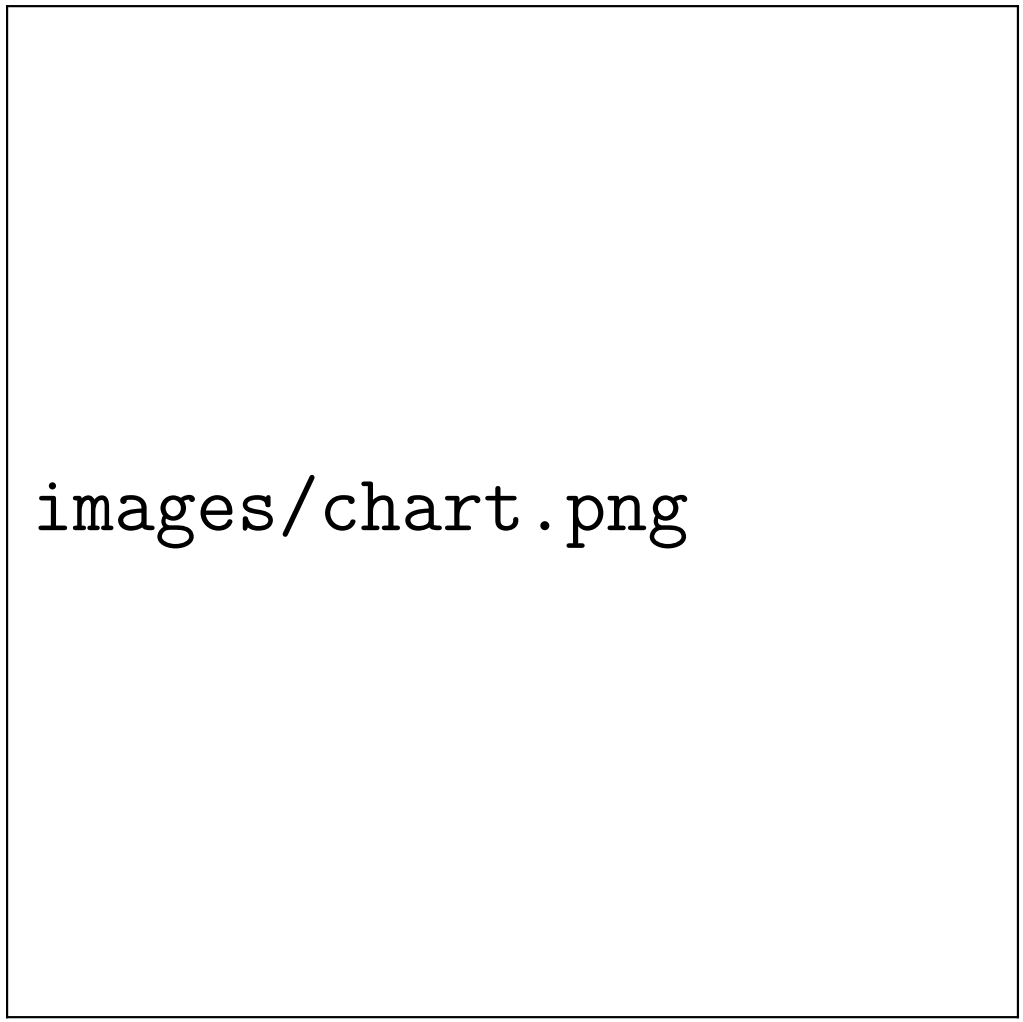

Figure: Comments on joind.in