

Serverless:
Infrastructure for a lazy dev



Ivan Barlog

AWS Solutions Architect



AWS user since 2014

Using Serverless daily since 2018

Sharing knowledge publicly since 2025

Ivan Barlog

AWS Solutions Architect



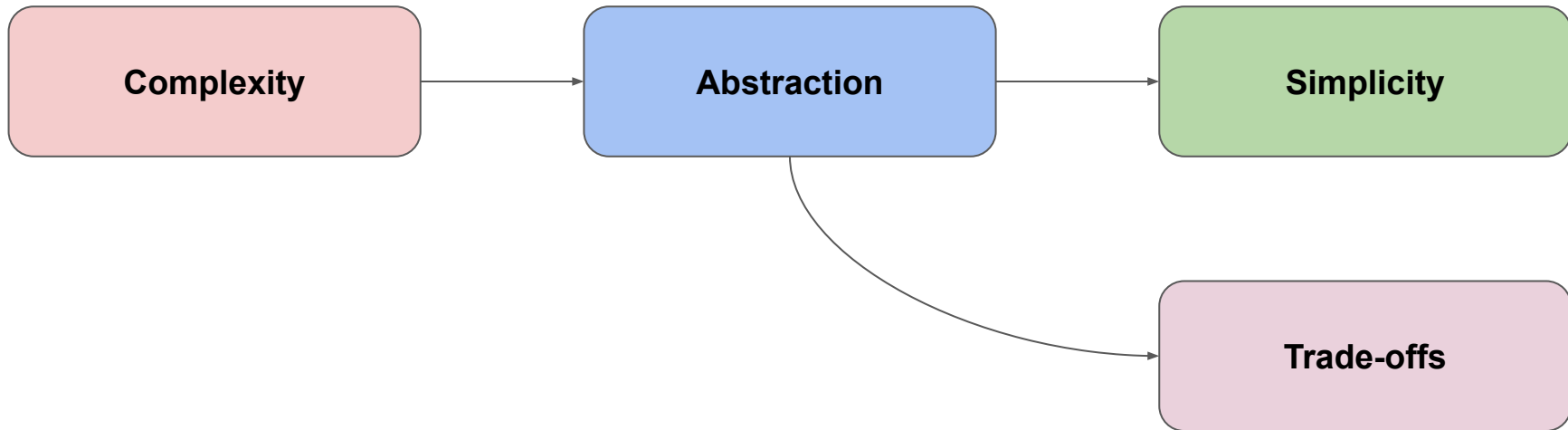
AWS user since 2014

Using Serverless daily since 2018

Sharing knowledge publicly since 2025

Don't really remember how hard Serverless was at first.





Cluster of servers

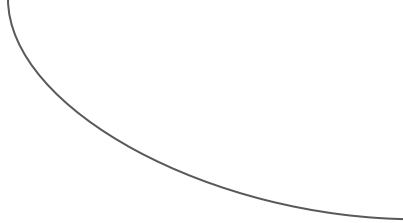
self managed,
runs 24/7 for any
utilization

Complexity

Abstraction

Simplicity

Trade-offs



Cluster of servers

self managed,
runs 24/7 for any
utilization

Serverless

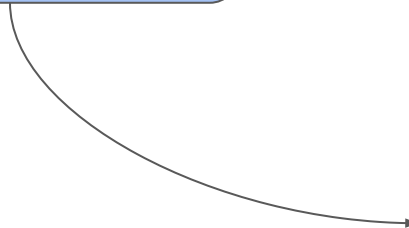
fully managed,
pay per use - stopped
when idle

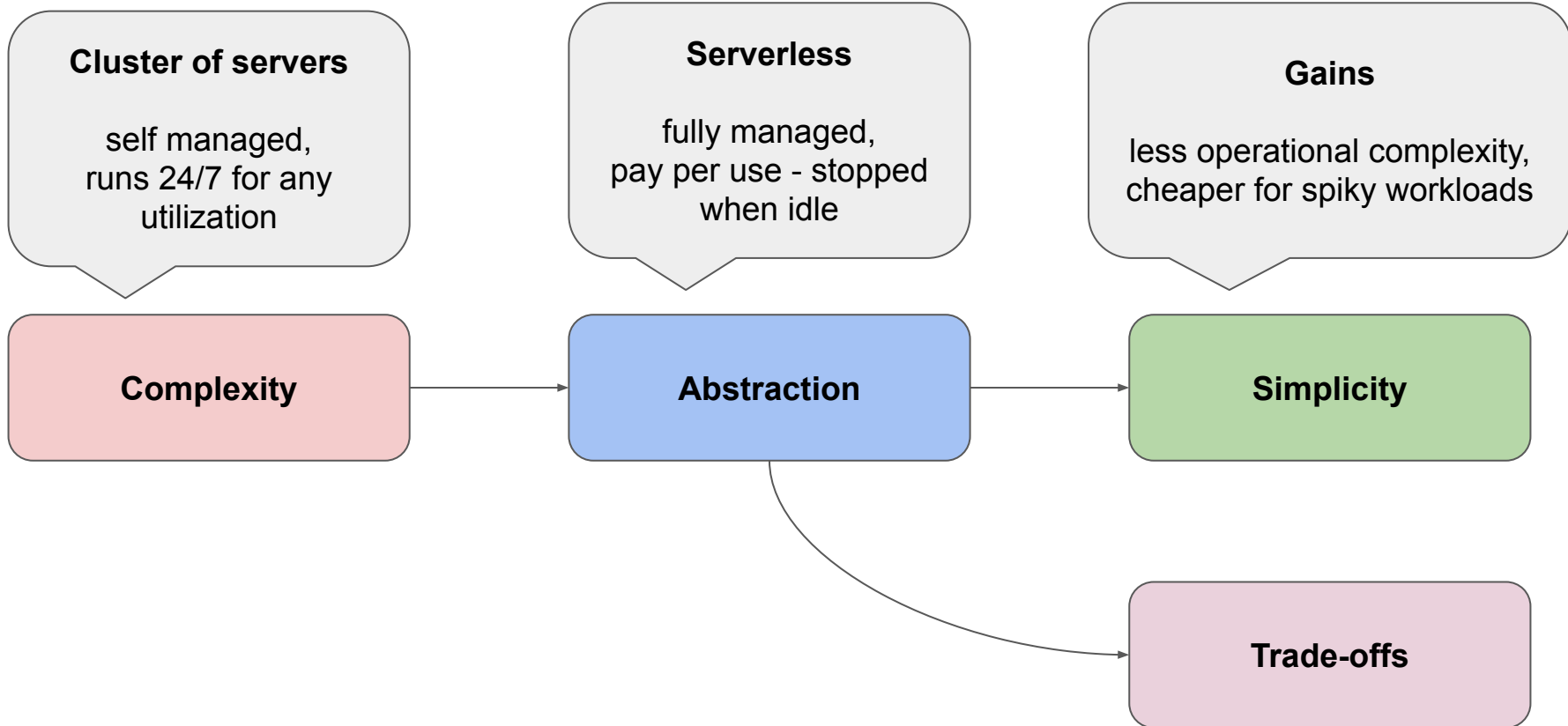
Complexity

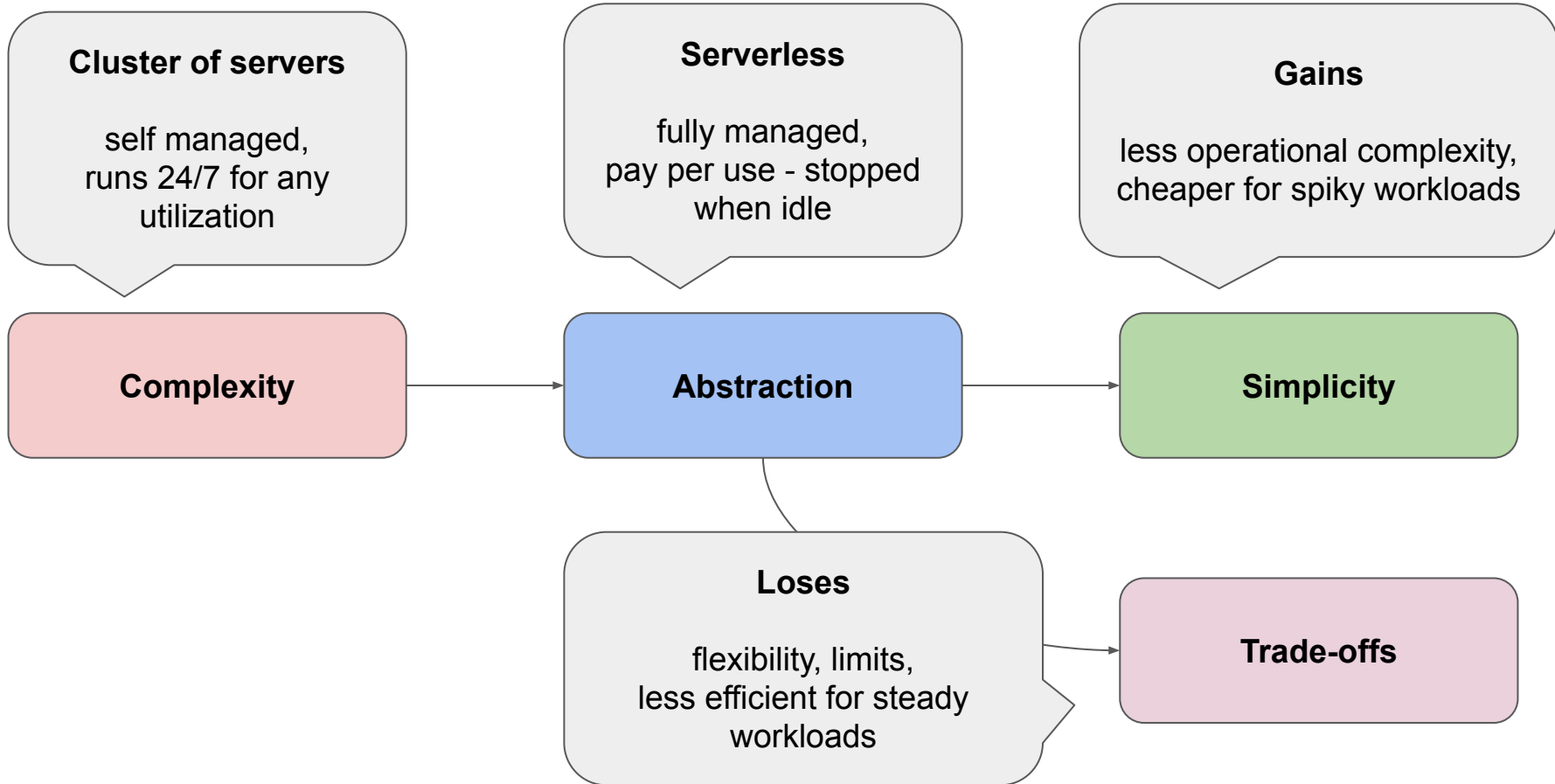
Abstraction

Simplicity

Trade-offs







What is Serverless?

Unicorns and magic



Serverless

Managed by AWS

- management of servers
- management of runtimes
- auto-scaling
- load balancing
- request routing
- patching
- ...

Pricing model

- pay per request
- pay per usage
- pay per duration
- scaling to 0
- on demand pricing

“I don’t want to manage my own cluster - that’s what AWS is for.”



Customer Data

Customer Data

Application

Application

OS

OS

Network + Firewall
Configuration

Network + Firewall
Configuration

Server-Side Encryption

Server-Side Encryption

Compute, Storage,
Networking

Compute, Storage,
Networking

Hardware

Hardware

You're responsible

AWS is responsible



Yan Cui

Serverless compute - not just **Lambda**

Lambda

- standard
- managed runtimes / container
- Lambda Managed Instances
- Step Functions
- Durable Functions
- microVM

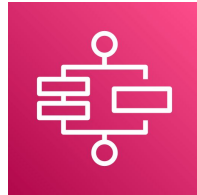
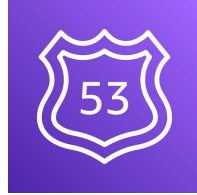
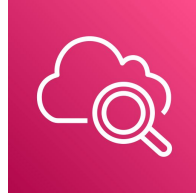
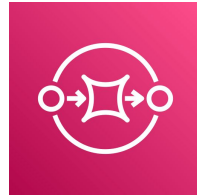
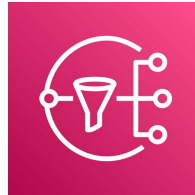
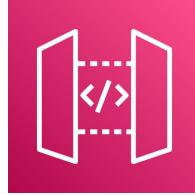
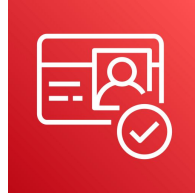


Elastic Compute Service (ECS)

- Fargate
- ECS Managed Instances
- ECS Express Mode



...not just compute



Where to start?

- start small
- use Lambda as glue code
- replace single micro-service

or

- go full-stack
- TypeScript everywhere



I got your back!

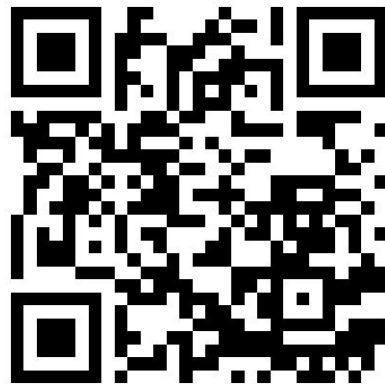
[@beesolve/lambda-fetch-api](#)

- converts AWS events to standard Fetch API
 - still provides access to original event and context
-
- no need to run server inside Lambda
 - almost no overhead
 - use with API Gateway or Function URL
 - does not forces you to use Fetch API everywhere



[@beesolve/knit-on-lambda](#)

- [SvelteKit](#) adapter for AWS Lambda
 - contains CDK construct for easy deployment
 - CloudFront + Function URL + S3 for assets
 - Node.js or Bun runtimes
 - support streaming in Node.js
-
- your full-stack application easily deployable to AWS serverless stack



@beesolve/sqs-handler

- best practice by default
 - CDK construct for Lambda behind SQS
 - runtime for easy async code invocation
 - synchronous invocation for local development
-
- it's simple but you should still understand what's under the hood



```
1 import { SQSClient } from "@aws-sdk/client-sqs";
2 import { createSqsHandlers } from "@beesolve/sqs-handler";
3
4 export const [handler, tasks] = createSqsHandlers({
5   functions: {
6     startLongTask: async (props: { id: string }) => { console.log(`running ${props.id}`); },
7     useExternalSystem: async (props: { whatever: number }) => { console.log(`running with ${props.whatever}`); },
8     startIdempotentTask: async (props: { idempotencyId: string }) => { console.log(`running with ${props.idempotencyId}`); }
9   },
10  sqsClient: new SQSClient(),
11  queueUrlOverride: { startLongTask: "longRunning" },
12  fifo: false,
13  queueUrls: { main: "mainQueueArn", longRunning: "longRunningQueueArn" },
14  localInvocation: true,
15 });
16
17 export type Tasks = typeof tasks;
18
```

```
1  import { asHttpV2Handler } from "@beesolve/lambda-fetch-api";
2
3  import { tasks } from "../tasks";
4
5  const fetchHandler = async (request: Request) => {
6    await tasks.
7      startIdempotentTask (property) startIdempotentTask: AddParameters<
8      return_new R startLongTask (property) startLongTask: AddParameters<(props:
9    };
10     useExternalSystem (property) useExternalSystem: AddParameters<
11
12  export const handler = asHttpV2Handler(fetchHandler);
13
14  export default fetchHandler;
```

```
1  import { asHttpV2Handler } from "@beesolve/lambda-fetch-api";
2
3  import { tasks } from "../tasks";
4
5  const fetchHandler = async (request: Request) => {
6      await tasks.startIdempotentTask({
7          idempotencyId: request.url,
8          notExistingProp: 'typescript shows error'
9      })
10
11     return new Response("created", { status: 201 });
12 };
13
14 export const handler = asHttpV2Handler(fetchHandler);
15
16 export default fetchHandler;
17
```

@beesolve/auth-service

- cookie-based, password-less authentication system
 - supports email codes and passkeys
 - deploy with API Gateway authorizer or without
 - can be easily mocked in local environment
-
- you have full control over the resources
 - there are no passwords to worry about
 - fully open-sourced - you can read the code



@beesolve/email-service

- Simple Email Service (SES) integration
 - asynchronous email sending via SQS
 - support for email attachments
 - support for React email templates
 - visibility with [Email dashboard](#)
-
- you still need to set up SES yourself
(there is no way around this)



Serverless database

DynamoDB

- distributed NoSQL key-value store
- highly available, fully serverless
- schemaless (use [valiobot](#))
- great for OLTP
- not ideal for OLAP (use [DuckDB](#))
- ACID through transactions

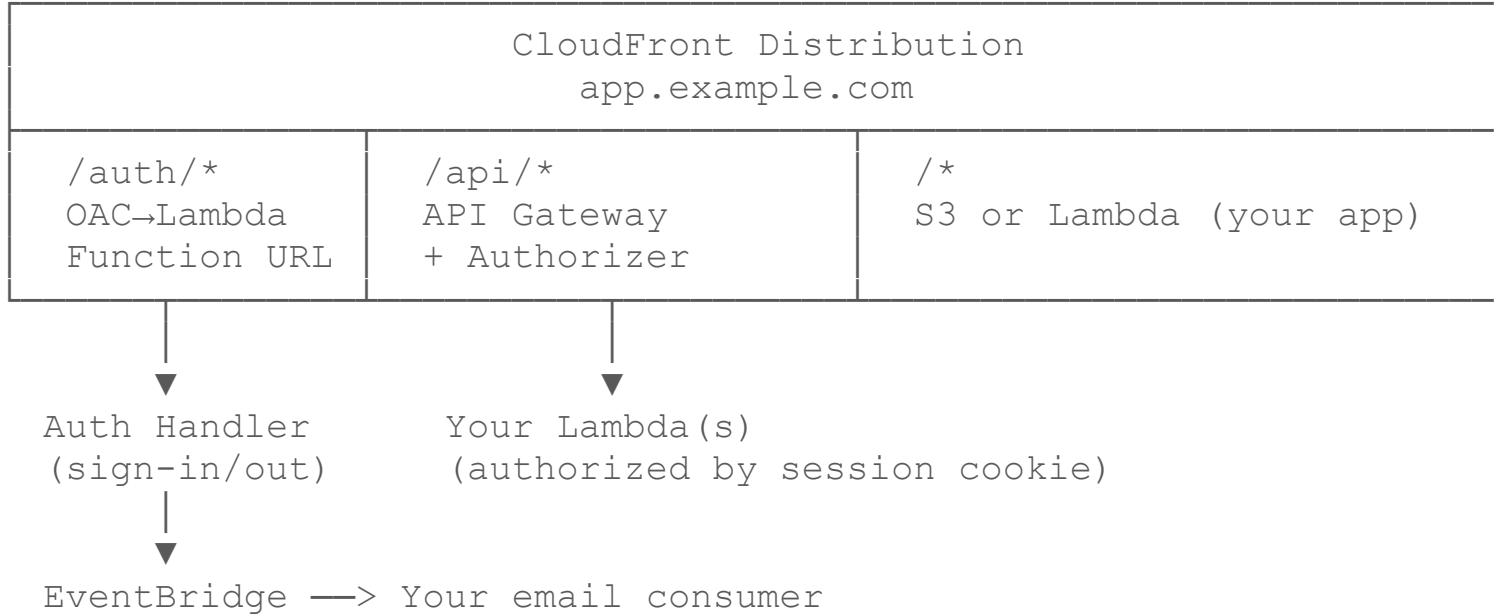


Aurora DSQL

- distributed PostgreSQL-compatible
- highly available, fully serverless
- compute on demand + durable storage
- supports foreign keys *NEW*
- optimistic concurrency control
- **not** a Postgres drop-in replacement



CloudFront, API Gateway and Function URL



FAQ

Q: How do you develop **locally**?

Q: How do you develop **locally**?

A: **Remocal** - execute code **locally** against **remote** resources

– Yan Cui ([Serverless, the HARD parts](#))

- Standard Fetch API Request/Response server
- Remote DynamoDB tables (or local Extend**DB**)
- Turning off “async” in SQS locally for rapid response
- Injecting a fake context for authorization



```
1 import api from "@app/api/api";
2 import app from "@app/web/src/app.html";
3 import auth from "@beesolve/auth-service/api";
4 import { runWithAwsContext } from "@beesolve/lambda-fetch-api";
5 import { serve } from "bun";
6 import { certificateFor } from "devcert";
7
8 const server = serve({
9   routes: {
10     "/*": app, // Serve index.html for all unmatched routes.
11     "/api/*": (request: Request) =>
12       runWithAwsContext(
13         // change user here if you want to see different role in dev
14         toFakeEvent("actual-user-id", request),
15         fakeContext,
16         () => api.fetch(request),
17       ),
18     "/auth/*": auth.fetch,
19   },
20   development: process.env.NODE_ENV !== "production" && {
21     hmr: true, // Enable browser hot reloading in development
22     console: true, // Echo console logs from the browser to the server
23   },
24   tls: await certificateFor("demo.dev"),
25   hostname: "demo.dev",
26 });
27
28 console.log(`🚀 Server running at ${server.url}`);
29
```

Q: How do you test **real** integrations locally?

Q: How do you test **real** integrations locally?

A: I don't.

1. Rapid local development
2. Testing on real infrastructure in staging account
3. Production release

CDK (Cloud Development Kit)

- infrastructure as a code (TypeScript)
- quick reproducible infrastructure in multi-account environment

Q: How do you deploy the code?

Q: How do you deploy the code?

- CDK (Cloud Development Kit) - infrastructure as a code
- highly reproducible
- deploys both infrastructure and code changes
- local deployment vs CI/CD
- secrets in Secrets Manager



Trade-off

- abstractions - when you delete code you delete resources
- not always in sync with API and CloudFormation

Q: What about the cold starts?

Q: What about the cold starts?

- **optimize your code** - size of code matters
- experiment with Lambda size ([aws-lambda-power-tuning](#))
- provision concurrency - too expensive
- SnapStart for Docker lambda

	startType	avg runtime ms	p50 runtime ms	p95 runtime ms	avg init time ms	p50 init time ms	p95 init time ms	invocations
tRPC	cold	433.2228	314.8588	952.958	542.4909	485.8537	703.2572	198
3.4 MB	warm	151.3207	51.3175	583.3668		0	0	1605
SvelteKit	cold	170.3859	75.998	460.242	287.383	295.806	324.48	16
870.9 kB	warm	16.9101	2.8476	70.4479		0	0	238

Q: What about the cost?



Q: What about the cost?

- no platform to manage
 - no DevOps team to hire
 - no idle resources
 - no maintenance of infrastructure
-
- cost of **learning**
 - cost of **mistakes**
 - cost of **starting new**



Key takeaways

Serverless is not a silver bullet.

When something looks simple, look for the trade-offs.

When using abstractions, go one level deeper to understand what's been abstracted away.

Understand why - what problem does it solve?

Think critically, evaluate multiple alternatives.

Build to stay relevant.

Not just because it's cool! 🙄



Ivan Barlog

AWS Solutions Architect



Github [ivanbarlog](https://github.com/ivanbarlog)
[beesolve](https://github.com/beesolve)

Email ivan@barlog.sk

Web barlog.sk
beesolve.com

LinkedIn [ivan-barlog](https://www.linkedin.com/in/ivan-barlog)

Go build something!