Why your next project should be serverless



vs.



Ivan Barlog

AWS Solutions Architect







AWS user since 2014

Using Serverless daily since 2018

Sharing knowledge publicly since 2025

Ivan Barlog

AWS Solutions Architect





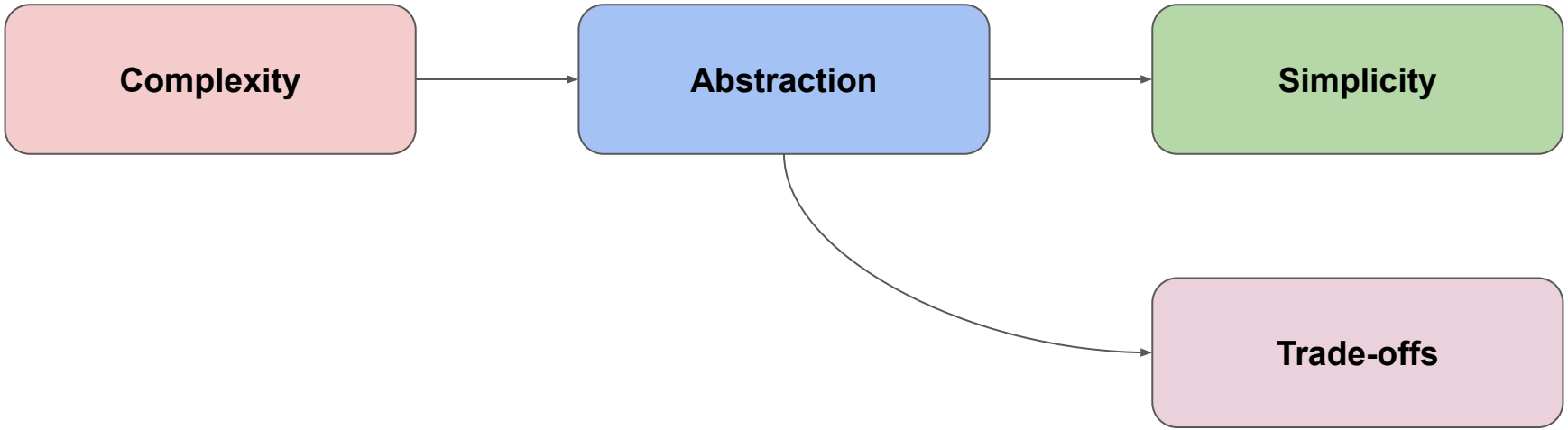


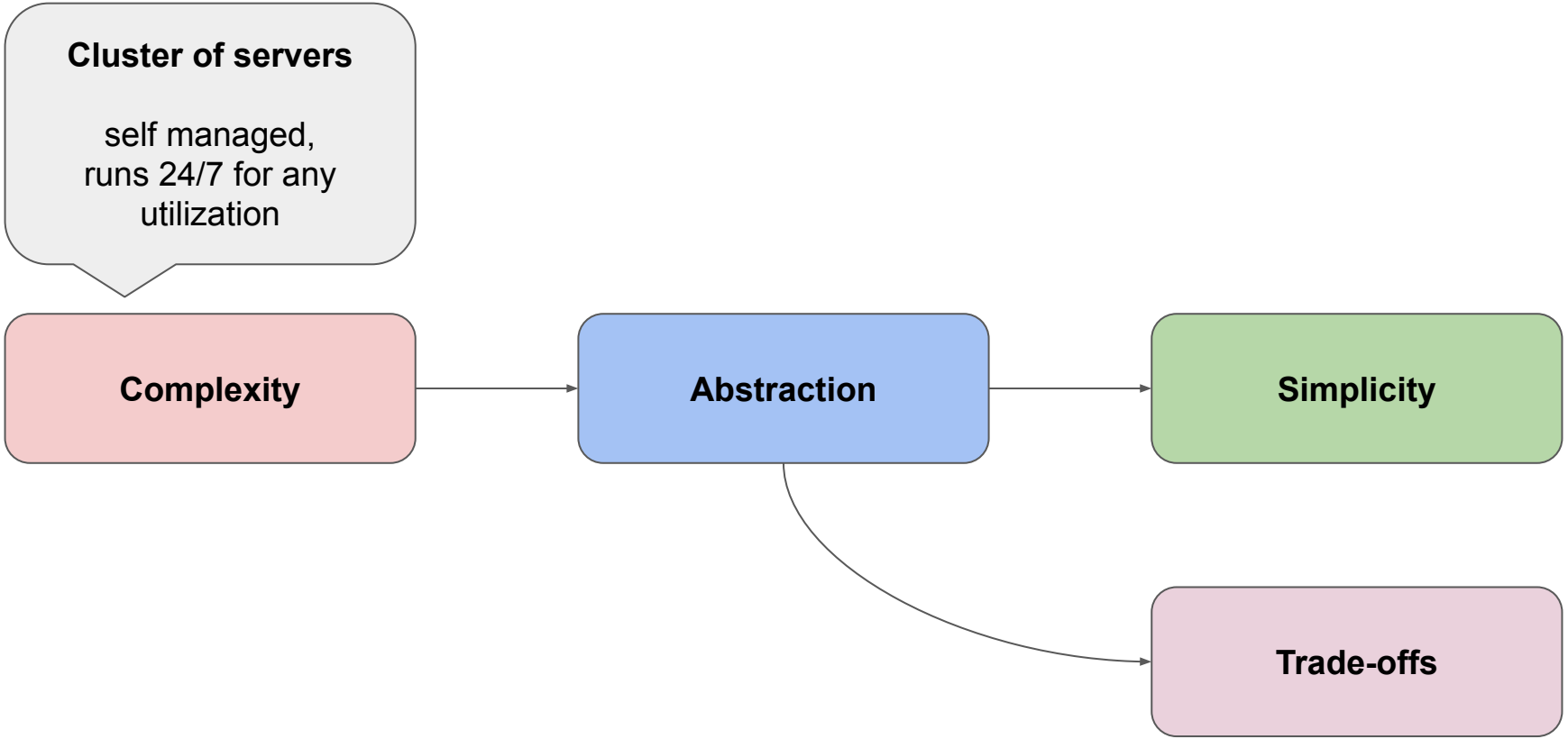
AWS user since 2014
Using Serverless daily since 2018
Sharing knowledge publicly since 2025

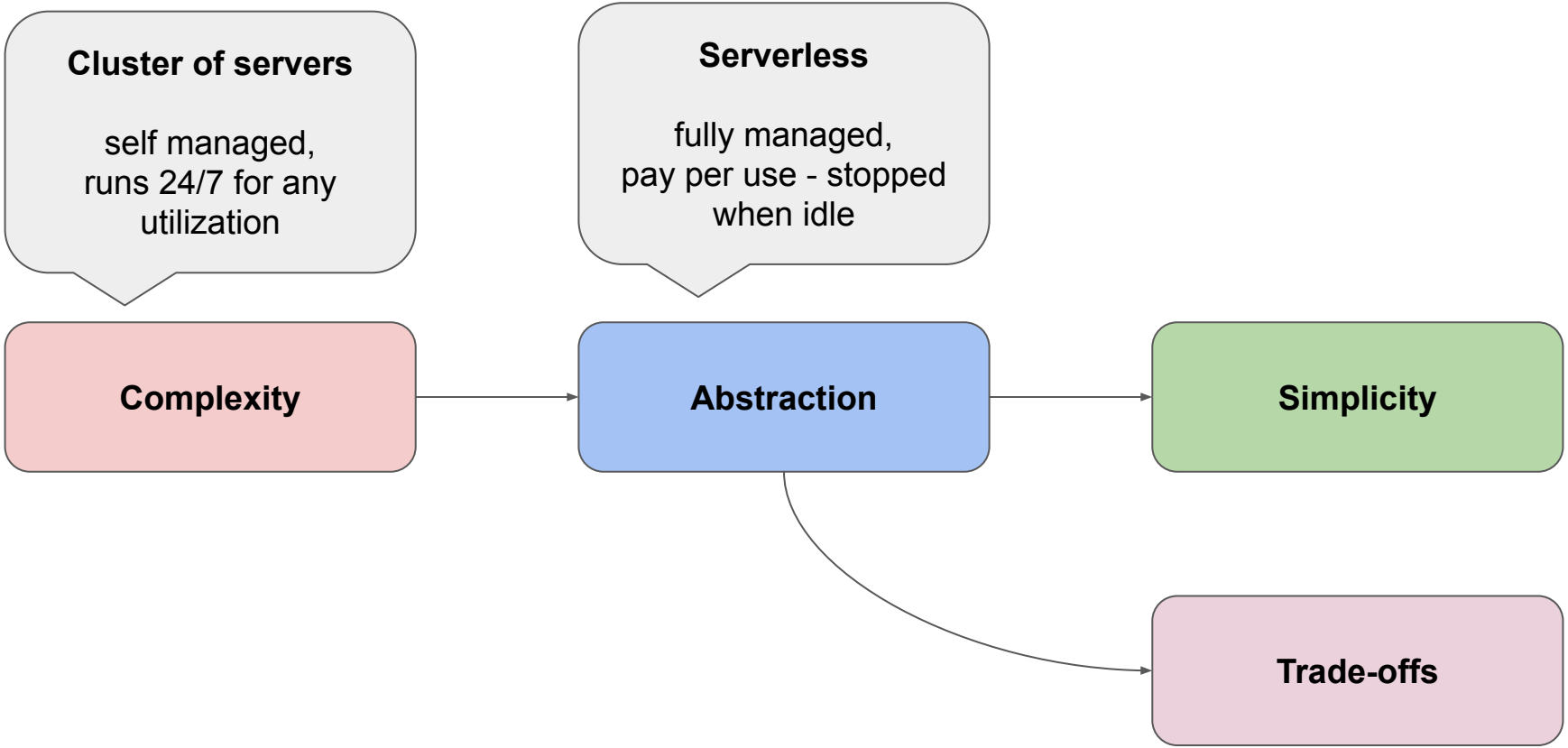
I use monoliths, micro-services and I think both are great!

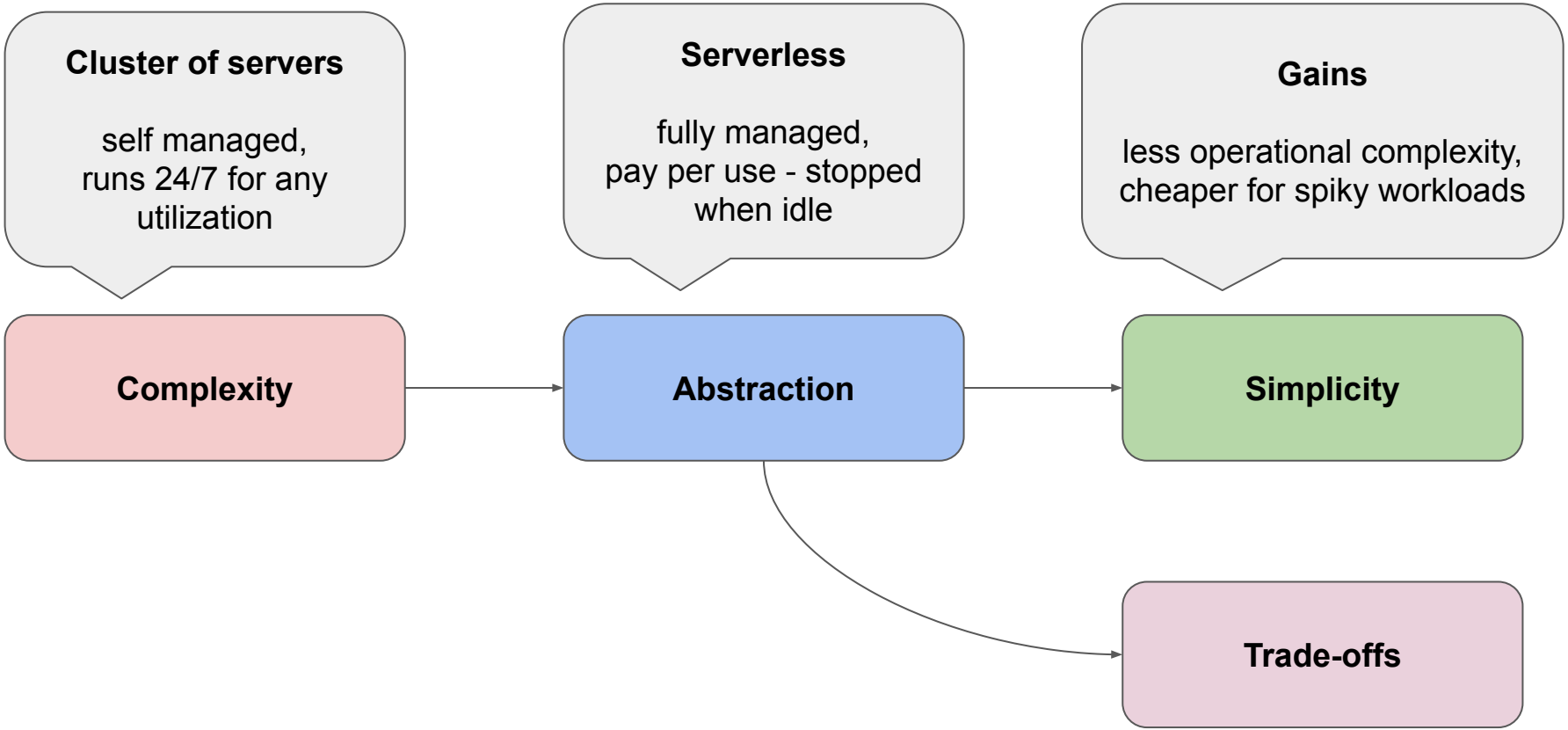
Serverless is simple.

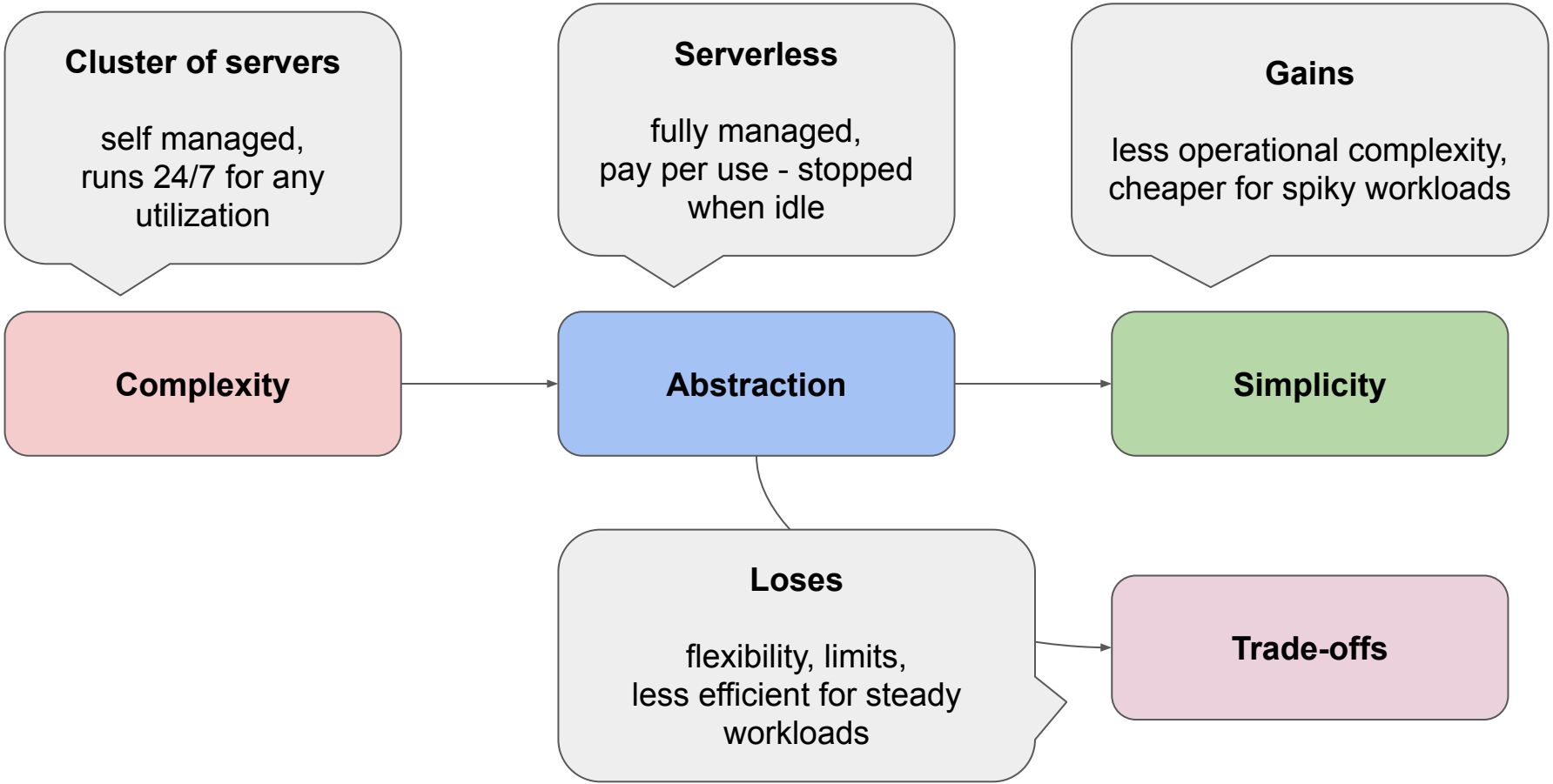

Complexity
Abstraction
Simplicity


Complexity
Abstraction
Simplicity
Trade-offs


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


Cluster of servers
self managed, runs 24/7 for any utilization
Serverless
fully managed, pay per use - stopped when idle
Gains
less operational complexity, cheaper for spiky workloads
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
Gains
less operational complexity, cheaper for spiky workloads
Complexity
Abstraction
Simplicity
Loses
flexibility, limits, less efficient for steady workloads
Trade-offs

What is Serverless?

Unicorns and magic





Serverless

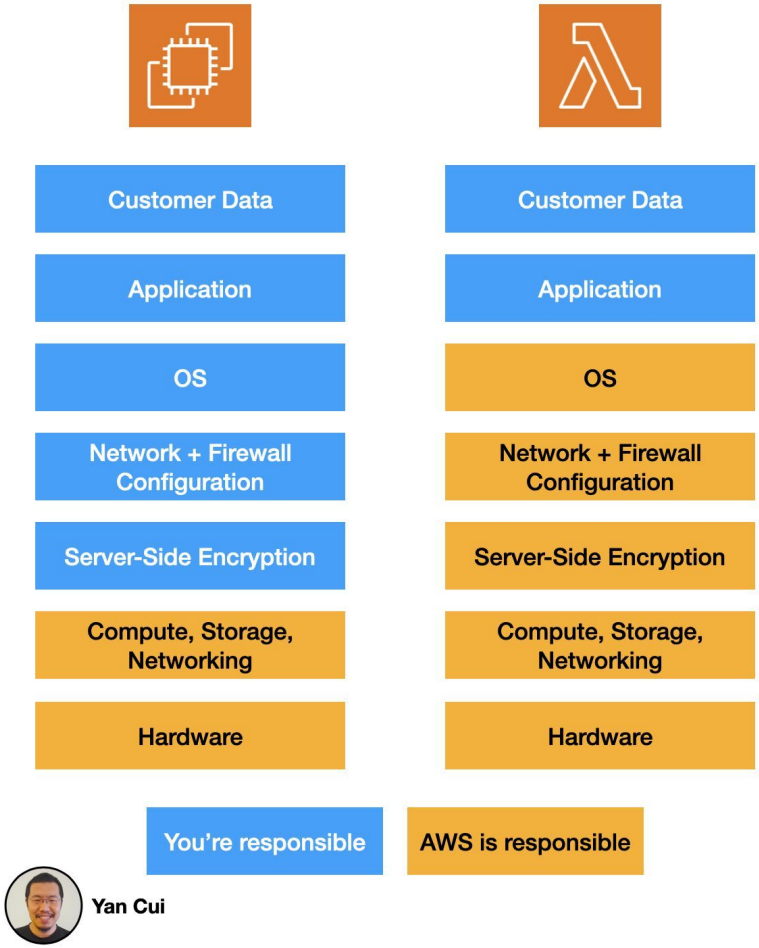
Managed by AWS

- 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.”



Serverless compute - not just **Lambda**

Lambda

- Event Functions (standard)
- Web Functions *NEW*
- managed runtimes / container
- Lambda Managed Instances
- Step Functions
- Durable Functions
- microVM

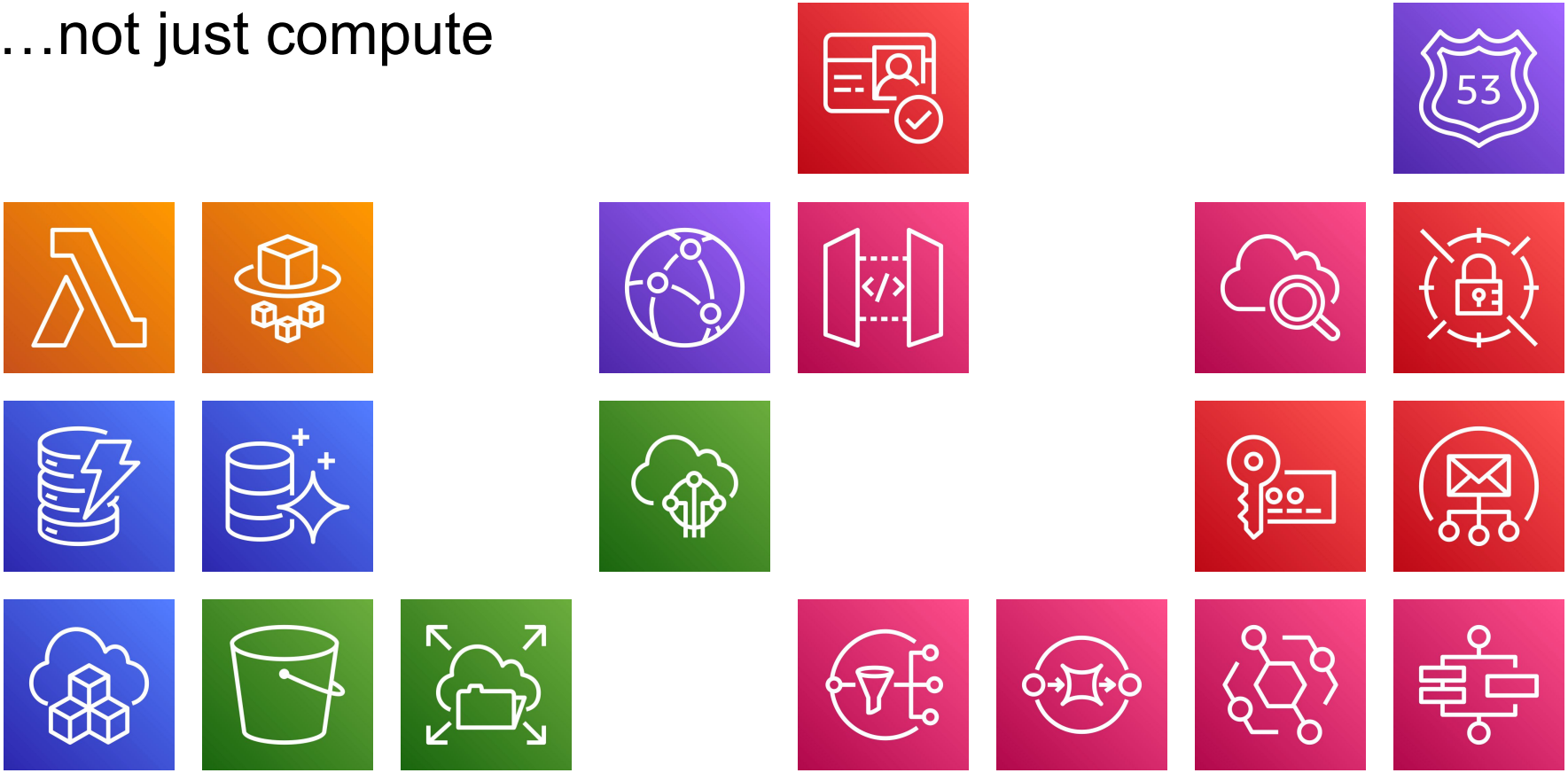


Elastic Container Service (ECS)

- Fargate
- ECS Managed Instances
- ECS Express Mode



…not just compute



Why use serverless then?

Why use serverless then?

> Sometimes you **win**, sometimes you **learn**.
>
> — anonymous

Where to start?

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

or

- go full-stack
- TypeScript everywhere



I got your back!

BeeSolve packages

- 20+ packages
- open source (MIT)
- TypeScript
- AWS focused
- CDK support
- AI assisted
- maintained

@beesolve/kit-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/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 (eg. in other events like SQS)



What about Lambda Web Functions?

- ability to run applications listening on port
- successor to aws-lambda-web-adapter
- first class citizen



- only `nodejs24` runtime
- 2 GB RAM per execution environment
- no request isolation by default
- not paying for idle CPU
- RAM is paid for whole execution lifetime (not just what is used)

@beesolve/lambda-function-url-protection

- adds secret to header sent from CloudFront to Lambda behind Function URL
- contains helpers for verifying the secret in runtime

- CloudFront OAC can only sign GET requests
- you need to sign your POST/PUT requests yourself
- CloudFront function has limits for reading the body
- Lambda@Edge might be an overkill

`@beesolve/aws-env-loader`

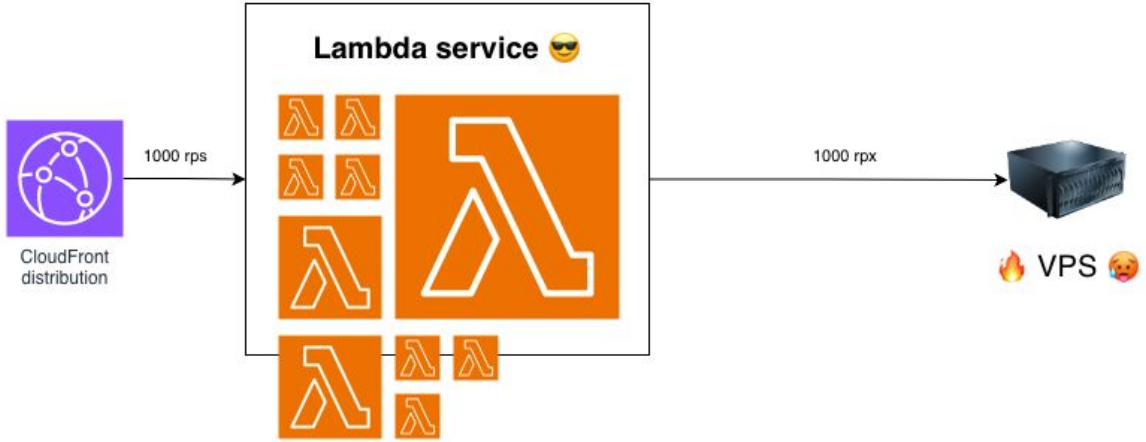
- infers environment variables from your CDK application
- caches values on your disk
- stores secrets in macOS keychain

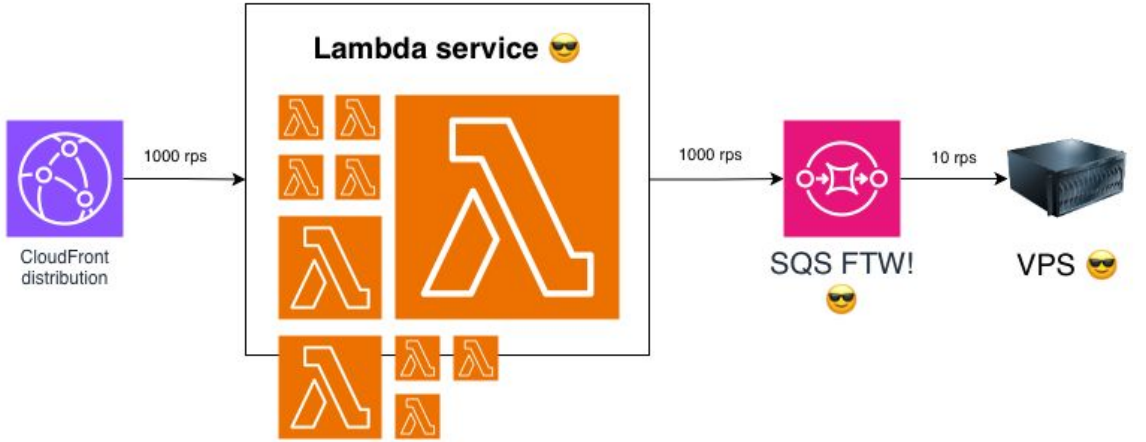
- no need for `.env.*` files
- secrets are encrypted
- loading on demand eg. not able to push to git



Simple Queue Service

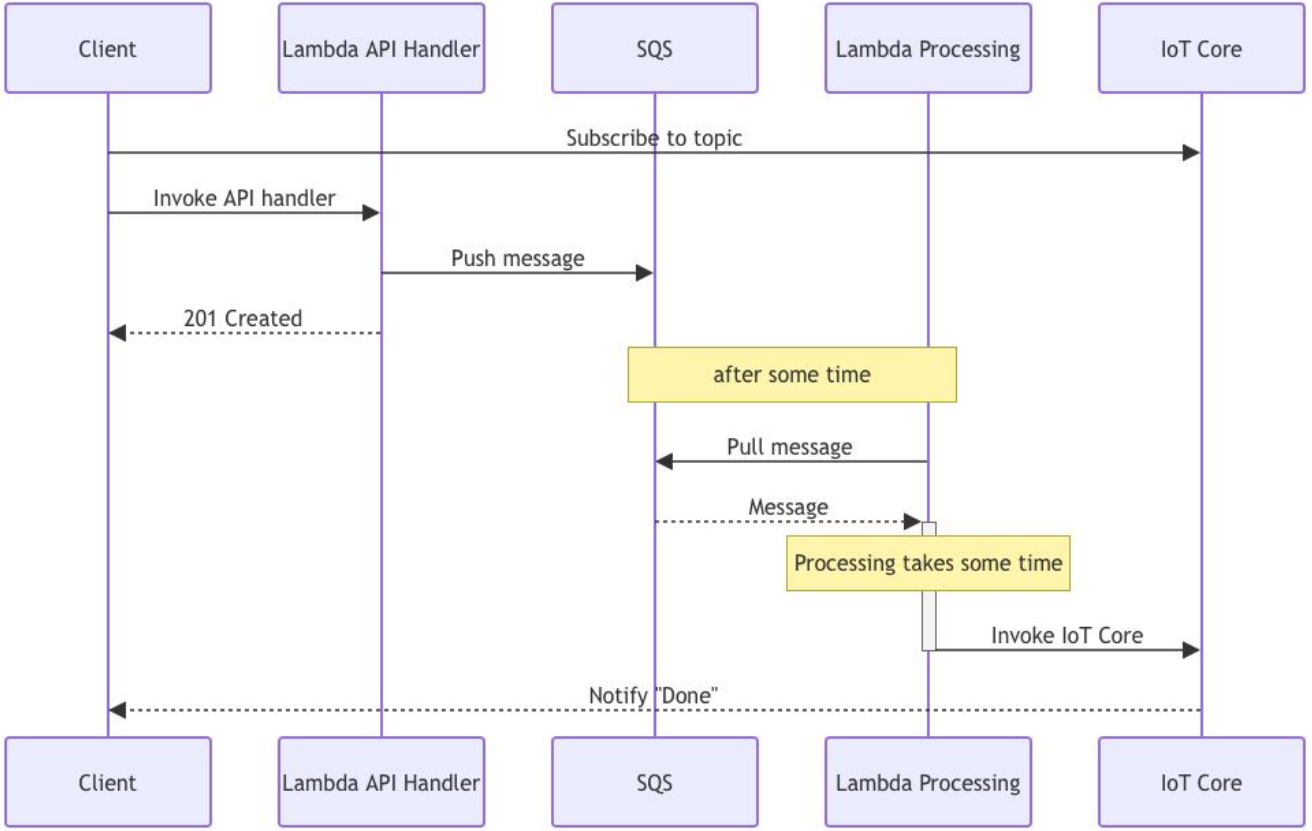
Buffer to protect downstream





Simple Queue Service

Asynchronous callback



@beesolve/sqs-handler

- best practice by default
- CDK construct for Lambda behind SQS
- runtime for easy async code invocation
- synchronous invocation for local development
- fully typed end-to-end

- it’s simple but you should still understand what’s under the hood



```ts
import { SQSClient } from "@aws-sdk/client-sqs";
import { createSqsHandlers } from "@beesolve/sqs-handler";

const sqsClient = new SQSClient();

export const [handler, tasks] = createSqsHandlers({
  fifo: false,
  sqsClient,
  queueUrls: {
    main: process.env.BEESOLVE_TASKS_MAIN_QUEUE_URL ?? "",
  },
  functions: {
    sendWelcomeEmail: async (userId: string) => {
      console.log(`Sending welcome email to ${userId}`);
    },
    processUpload: async (fileKey: string, metadata: { size: number }) => {
      console.log(`Processing ${fileKey} (${metadata.size} bytes)`);
    },
  },
});

```

```ts
import { asHttpV2Handler } from "@beesolve/lambda-fetch-api";

import { tasks } from "./tasks.ts";

const fetchHandler = async (request: Request): Promise<Response> => {
  await tasks.sendWelcomeEmail("user-123");

  await tasks.processUpload("uploads/file.png", {
    size: 4096,
    nonExistingParameter: "type error",
  });

  return new Response("created", { status: 201 });
};

export const handler = asHttpV2Handler(fetchHandler);

export default fetchHandler;

```

processUpload (property) processUpload: (fileKey: string, meta
sendWelcomeEmail (property) sendWelcomeEmail: (userId: string,

Code sample.

@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 `valibot`)
- 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



@beesolve/dsql-helpers

- group-expense-with-ease.com - migrated from DynamoDB to DSQL
- contains helpers and CDK construct
- Drizzle ORM
- AWS official DSQL adapter for Drizzle

- an official AWS Drizzle adapter is 0.x version
- still young - needs feedback



FAQ

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

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

A: **Remocal** - execute code lo**cal**ly against **remo**te resources

– Yan Cui (Serverless, the HARD parts)

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



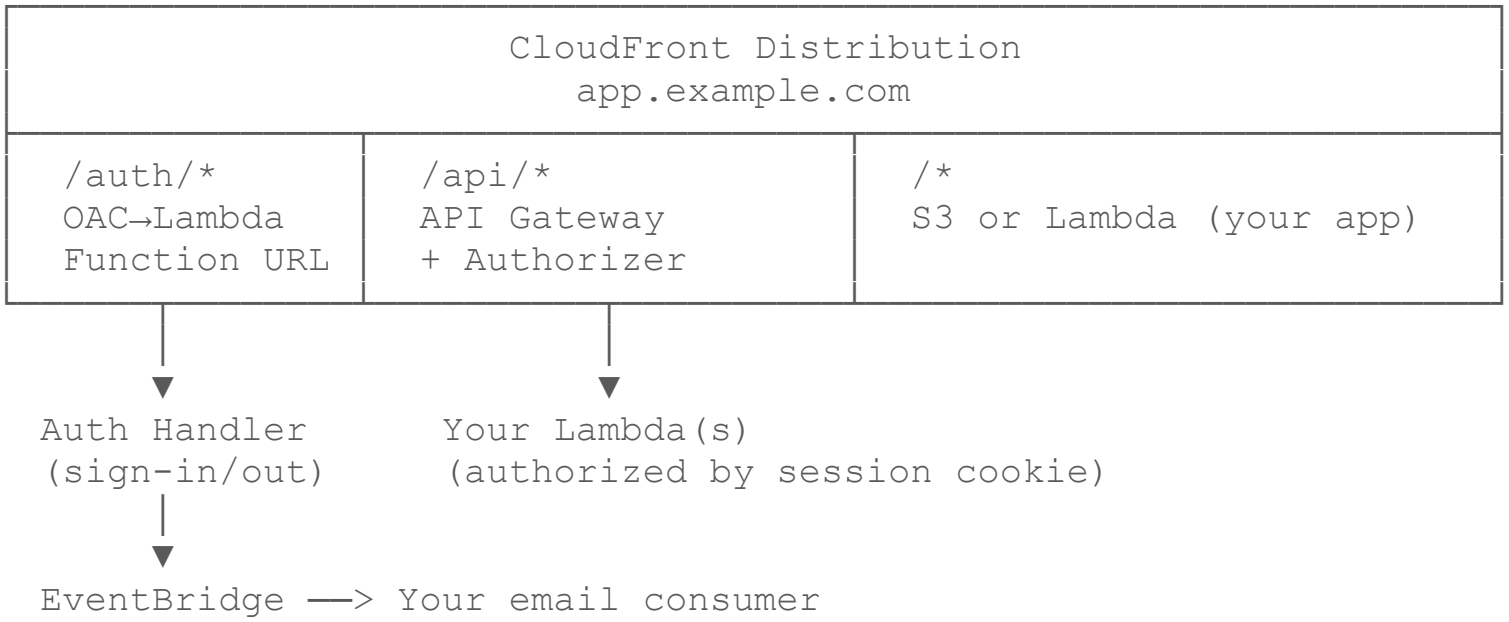
```ts
import api from "@app/api/api";
import app from "@app/web/src/app.html";
import auth from "@beesolve/auth-service/api";
import { runWithAwsContext } from "@beesolve/lambda-fetch-api";
import { serve } from "bun";
import { certificateFor } from "devcert";

const server = serve({
  routes: {
    "/*": app, // Serve index.html for all unmatched routes.
    "/api/*": (request: Request) =>
      runWithAwsContext(
        // change user here if you want to see different role in dev
        toFakeEvent("actual-user-id", request),
        fakeContext,
        () => api.fetch(request),
      ),
    "/auth/*": auth.fetch,
  },
  development: process.env.NODE_ENV !== "production" && {
    hmr: true, // Enable browser hot reloading in development
    console: true, // Echo console logs from the browser to the server
  },
  tls: await certificateFor("demo.dev"),
  hostname: "demo.dev",
});

console.log(`🚀 Server running at ${server.url}`);

```

CloudFront, API Gateway and Function URL



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 (additional cost)

	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
beesolve

Email ivan@barlog.sk

Web barlog.sk
beesolve.com

LinkedIn ivan-barlog

Go build something!