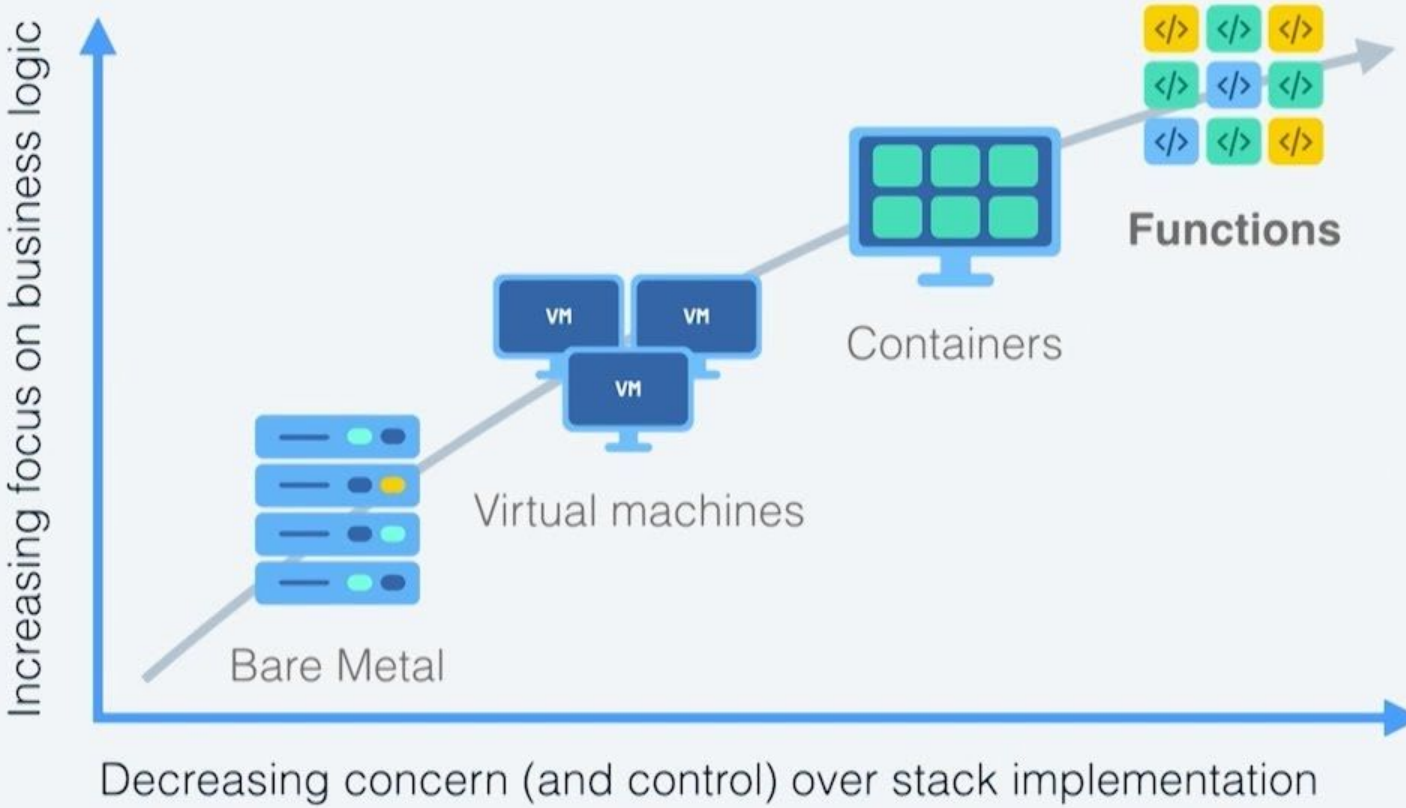
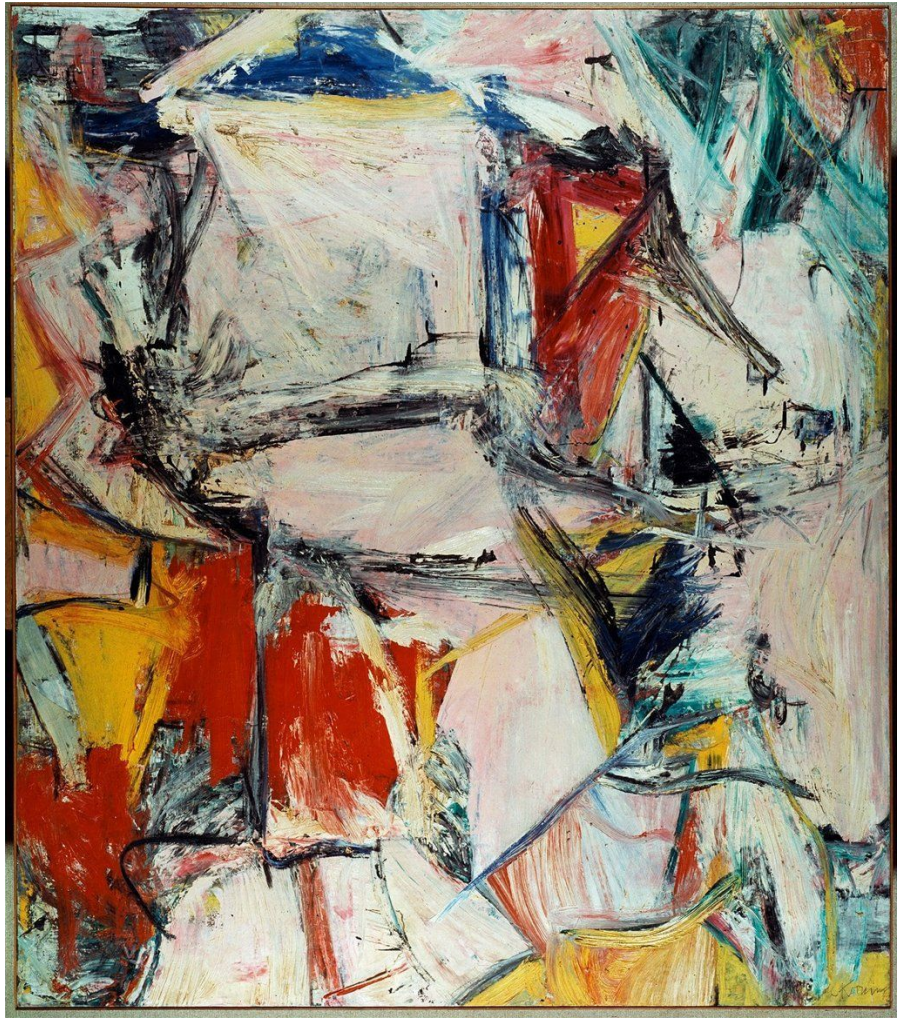

Let me in, I want to see with my own eyes!
No, it's no use...
Aha! I knew it! Servers!
Um...
And these? Aren't these hard drives? And these cables?
Yeah...
You told me I'd be getting SERVERLESS architecture!
You've been caught red handed! What a scam!
CommitStrip.com

Evolution of serverless

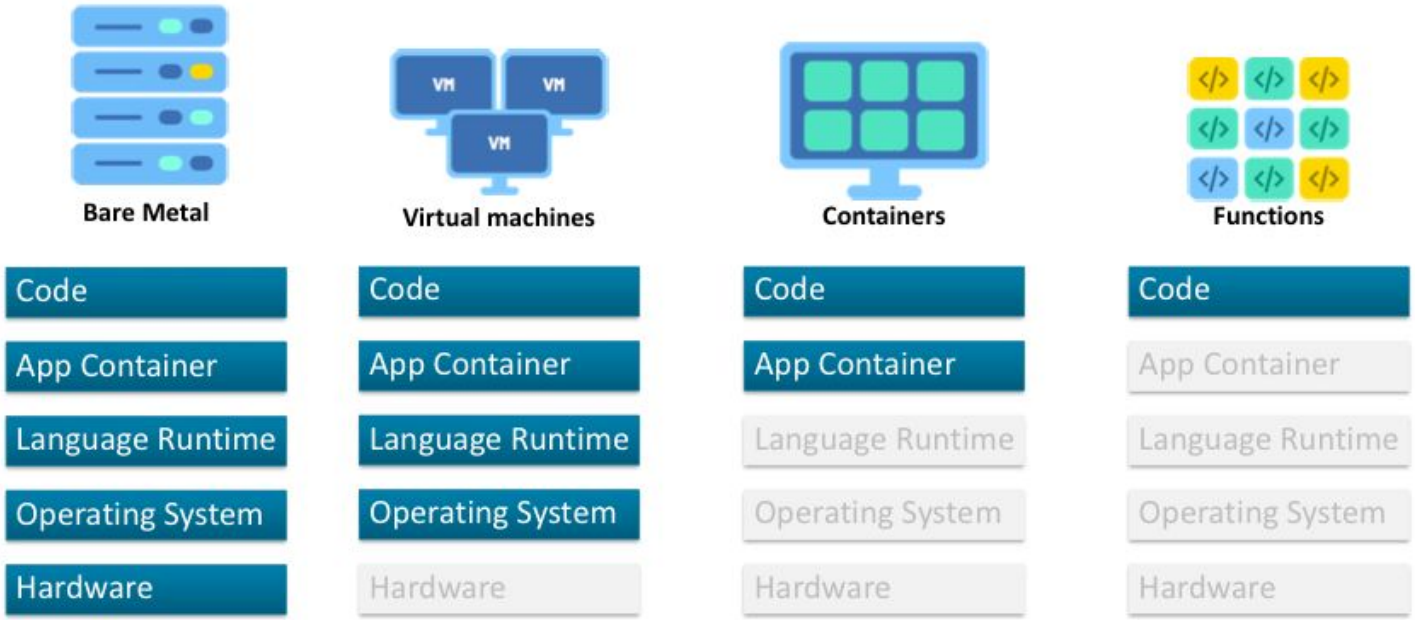


https://www.youtube.com/watch?v=3vaqTIIcfvY



https://www.pinterest.at/pin/574560864960649564/

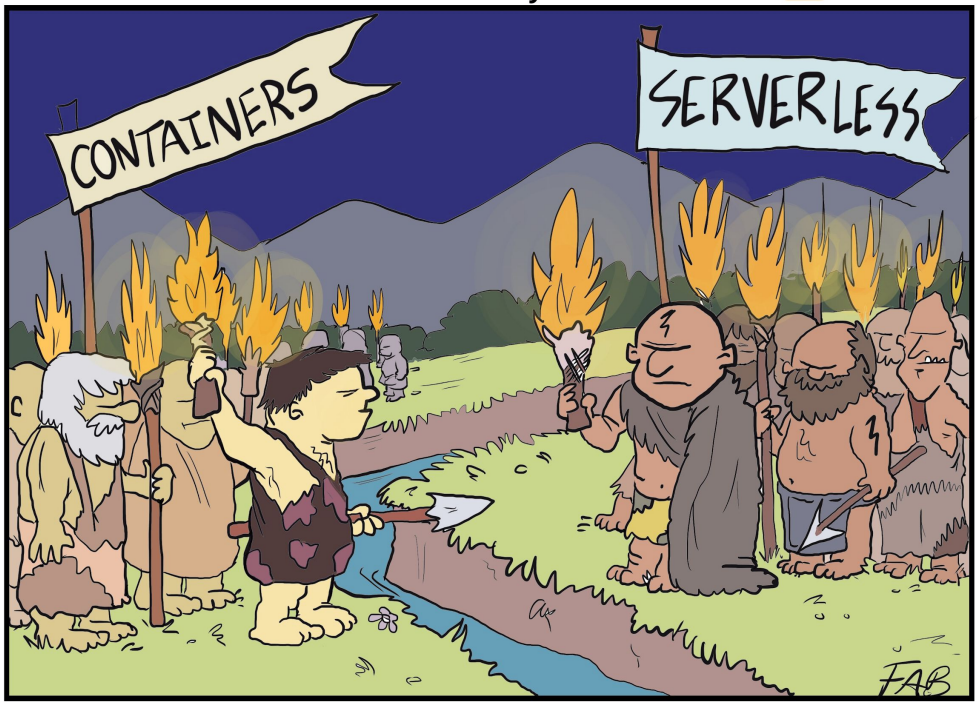
Willem de Kooning: Interchange (303 000 000 $)



https://blogs.oracle.com/developers/functions-as-a-service:-evolution,-use-cases,-and-getting-started

FaaS and Furious by Forrest Brazeal

A CLOUD GURU



The two tribes regarded each other suspiciously in the glow of their brightly blazing production environments.

https://faasandfurious.com/

Why serverless?

Serverless enables you to build modern applications with increased agility and lower total cost of ownership.

No server management
There is no need to provision or maintain any servers. There is no software or runtime to install, maintain, or administer.

Flexible scaling
Your application can be scaled automatically or by adjusting its capacity through toggling the units of consumption (e.g. throughput, memory) rather than units of individual servers.

Pay for value
Pay for consistent throughput or execution duration rather than by server unit.

Automated high availability
Serverless provides built-in availability and fault tolerance. You don't need to architect for these capabilities since the services running the application provide them by default.

https://aws.amazon.com/serverless/

Why not?

Observability is more difficult
It's probably the biggest critique of serverless right now: you just lose some amount of critical insight into your functions. Serverless encourages event-based architectures, which a lot of people aren't familiar with. Add to that, that serverless is a new enough space that the available tooling is relatively immature. It can be hard to do things as simple as stack traces.

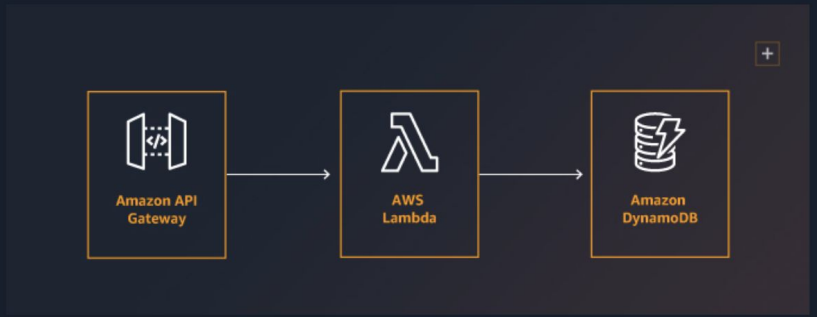
Latency
Serverless functions mean you'll be dealing with cold starts. Need to keeping functions warm or provisioned concurrency (new 2019).

Heavier reliance on vendor ecosystems
With serverless, you don't manage the server. That also means you lose control over server hardware, runtimes and runtime updates (at the time of writing, Node.js 8 is out but AWS is still on Node.js 6).
The specifics of your application architecture can suddenly become determined by the provider you're using.

https://serverless.com/blog/when-why-not-use-serverless/

Architecture: Microservices

Monolithic applications are popular because they are fast to develop. However, they become difficult to scale and update as the code base grows because each aspect of the application is tightly coupled. When applications are built with modular independent components, called microservices, release velocity can increase because changes to any component are easier to make. Microservices make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market.



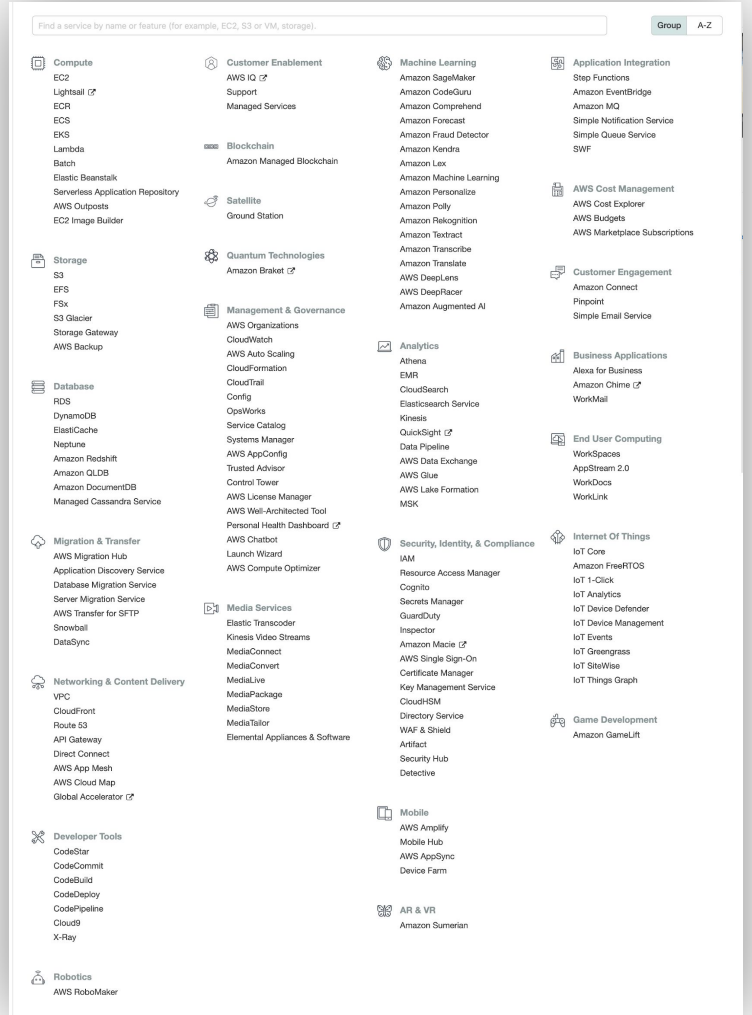
Operations: As Serverless as Possible

Modern applications have a lot of moving parts – many microservices with unique databases, all releasing features frequently. Operating applications with serverless services eliminates environment management, easing that burden. There are serverless services for the entire application stack: compute, storage, and integration. These services run without the need for infrastructure provisioning and scaling, have built in availability and security, and use a pay-for-value billing model.

Data: Decoupled & Purpose Built

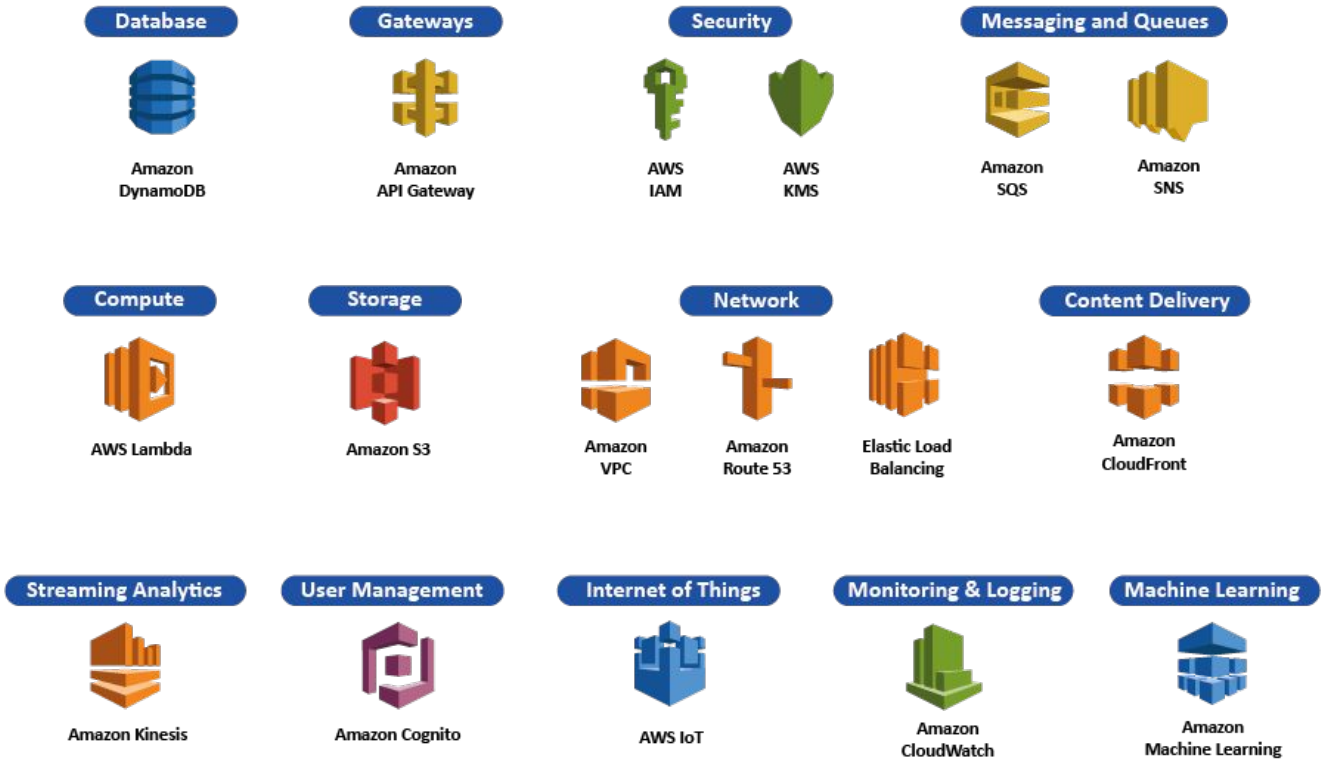
Much like a monolithic application, a single database is also difficult to scale. It can become a single point of failure with fault tolerance challenges. Modern applications take advantage of decoupled data stores in which there is a one-to-one mapping of database and microservice. By decoupling data along with microservices, teams are free to choose the database that best fits the needs of the service – choosing a database that is purpose built for the task at hand.

https://aws.amazon.com/modern-apps/



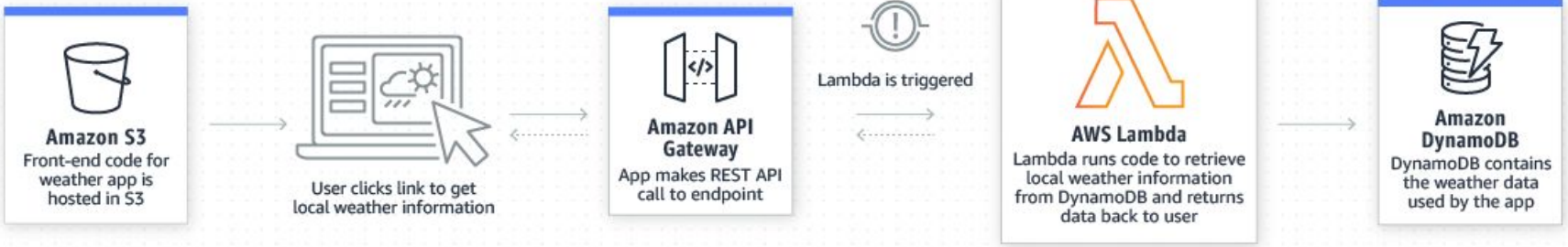
http://console.aws.amazon.com/

Serverless



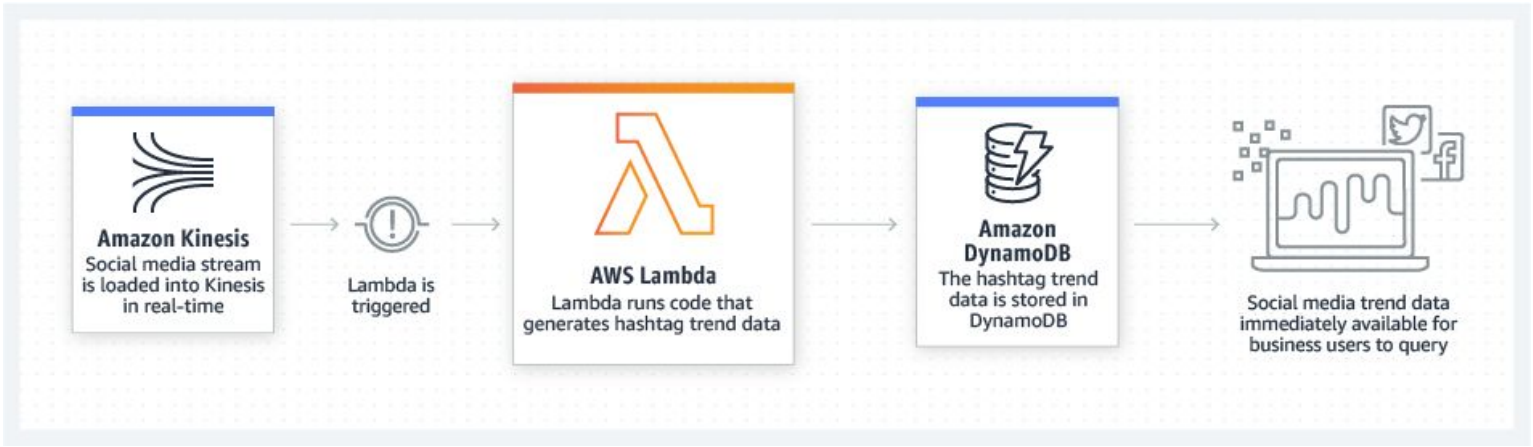
https://www.ecloudvalley.com/solutions/serverless/

Web applications and backends



https://aws.amazon.com/lambda/

Data processing



https://aws.amazon.com/lambda/

AWS Lambda Limits

Resource	Default Limit
Concurrent executions	1,000
Function and layer storage	75 GB
Elastic network interfaces per VPC	250

Resource	Limit
Function memory allocation	128 MB to 3,008 MB, in 64 MB increments.
Function timeout	900 seconds (15 minutes)
Function environment variables	4 KB
Function resource-based policy	20 KB
Function layers	5 layers
Function burst concurrency	500 - 3000 (varies per region)
Invocation frequency (requests per second)	10 x concurrent executions limit (synchronous – all sources) 10 x concurrent executions limit (asynchronous – non-AWS sources) Unlimited (asynchronous – AWS service sources)
Invocation payload (request and response)	6 MB (synchronous) 256 KB (asynchronous)
Deployment package size	50 MB (zipped, for direct upload) 250 MB (unzipped, including layers) 3 MB (console editor)
Test events (console editor)	10
`/tmp` directory storage	512 MB
File descriptors	1,024
Execution processes/threads	1,024

AWS Lambda Pricing

Region: EU (Frankfurt)

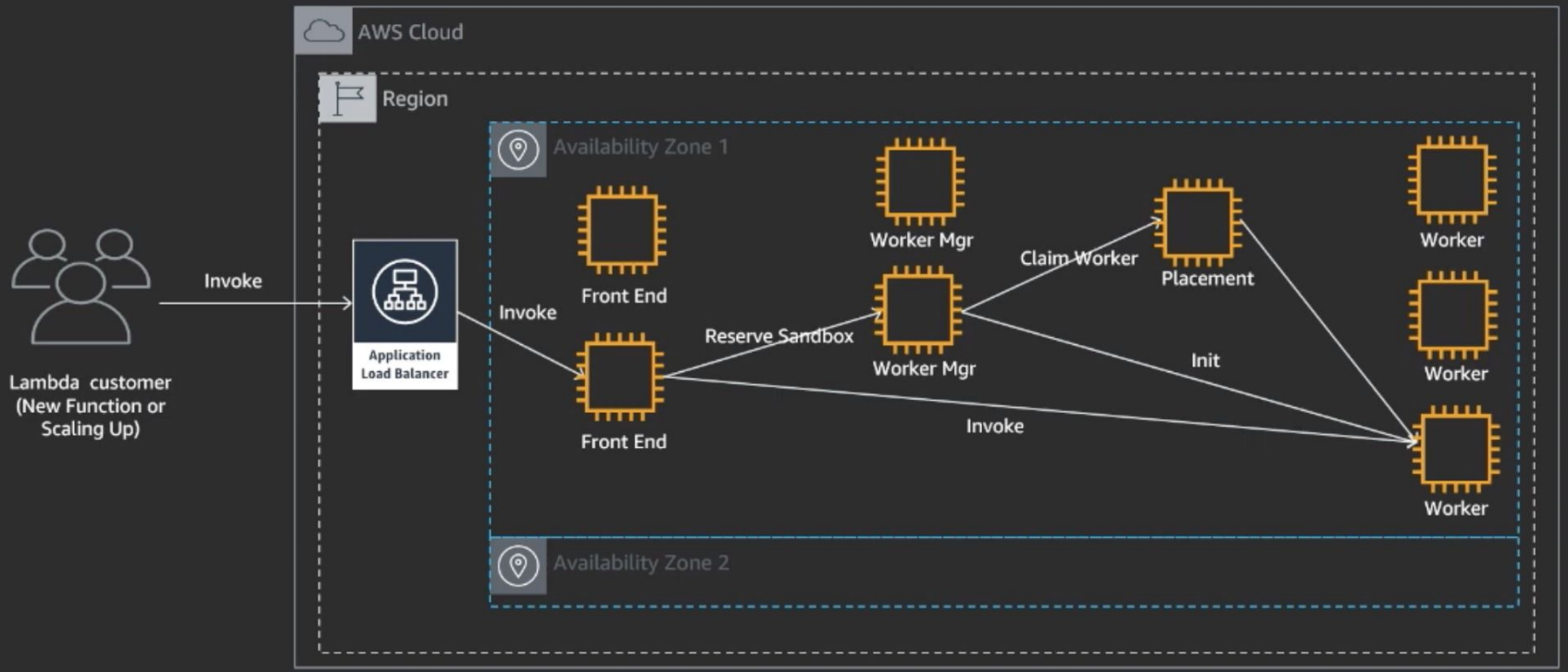
	Price
Requests	$0.20 per 1M requests
Duration	$0.000016667 for every GB-second

The price for **Duration** depends on the amount of memory you allocate to your function. You can allocate any amount of memory to your function between 128MB and 3008MB, in 64MB increments. The table below contains a few examples of the price per 100ms associated with different memory sizes.

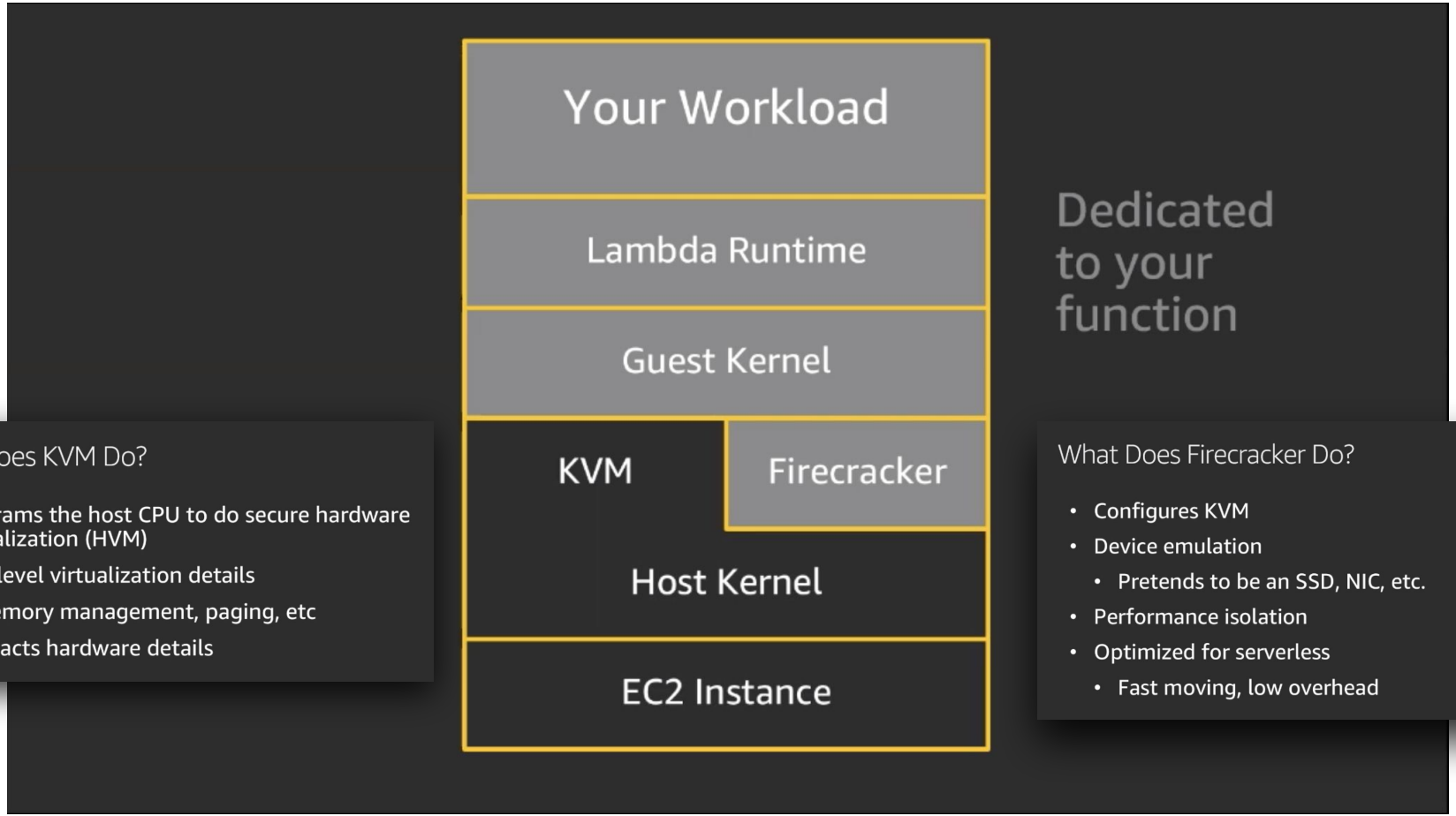
Memory (MB)	Price per 100ms
128	$0.000000208
512	$0.000000833
1024	$0.000001667
1536	$0.000002500
2048	$0.000003333
3008	$0.000004896

https://aws.amazon.com/lambda/pricing/
https://docs.aws.amazon.com/lambda/latest/dg/limits.html

Synchronous First Time Invoke or Scale Up



https://www.youtube.com/watch?v=xmacMfbrG28



https://github.com/firecracker-microvm/firecracker/blob/master/docs/design.md

https://www.youtube.com/watch?v=xmacMfbrG28 AWS re:Invent 2019: [REPEAT 1] A serverless journey: AWS Lambda under the hood (SVS405-R1)

Infrastructure as a Code (actual code)

Use the AWS CDK to define your cloud resources in a familiar programming language. The AWS CDK supports TypeScript, JavaScript, Python, Java, and C#/.Net.

Other advantages of the AWS CDK include:

- Use logic (if statements, for-loops, etc) when defining your infrastructure
- Use object-oriented techniques to create a model of your system
- Define high level abstractions, share them, and publish them to your team, company, or community
- Organize your project into logical modules
- Share and reuse your infrastructure as a library
- Testing your infrastructure code using industry-standard protocols
- Use your existing code review workflow
- Code completion within your IDE

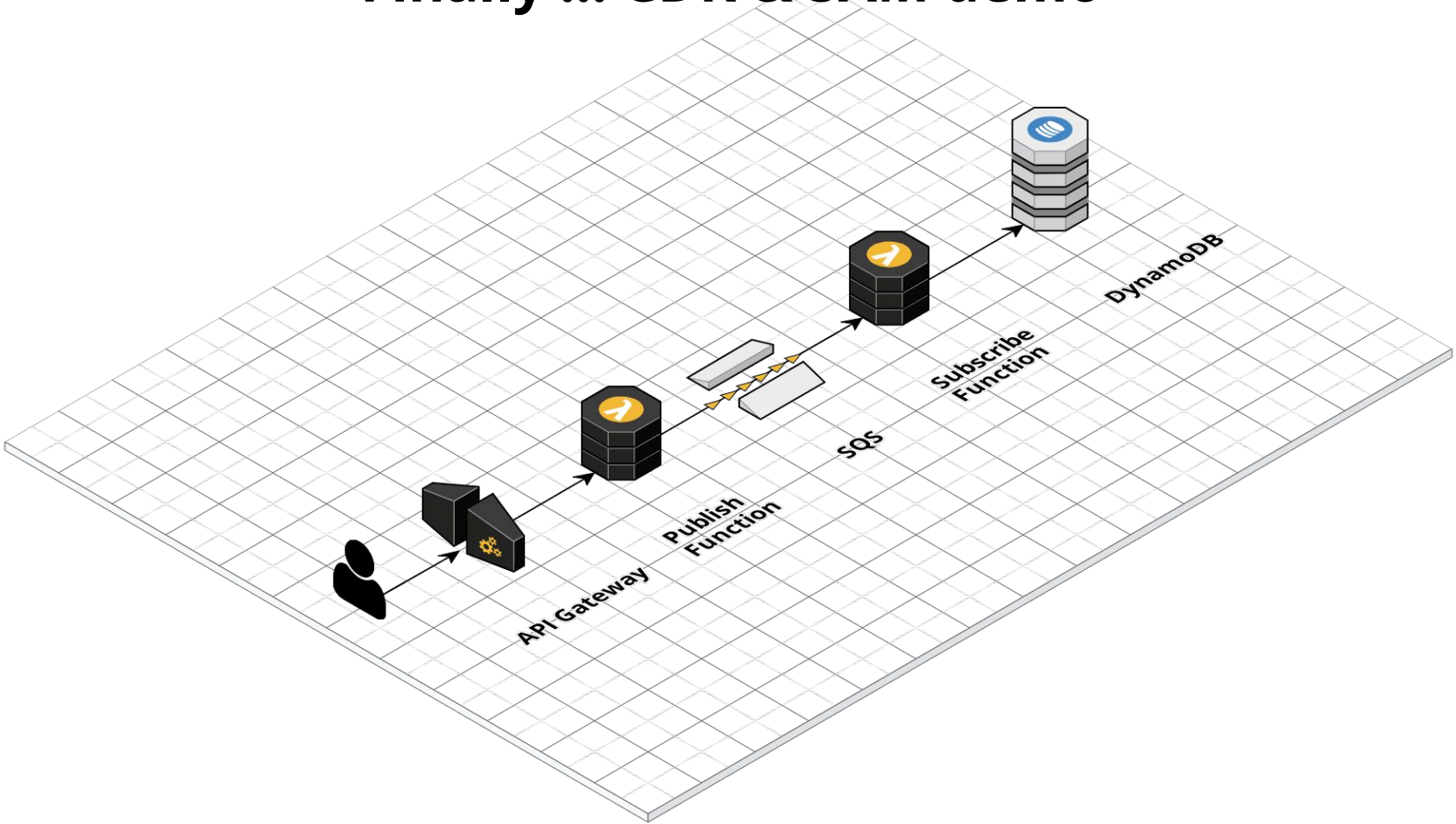
https://aws.amazon.com/cdk/

Serverless Framework (AWS SAM)

- **Single-deployment configuration**. AWS SAM makes it easy to organize related components and resources, and operate on a single stack.
- **Extension of AWS CloudFormation**. Because AWS SAM is an extension of AWS CloudFormation, you get the reliable deployment capabilities of AWS CloudFormation.
- **Built-in best practices**. You can use AWS SAM to define and deploy your infrastructure as config.
- **Local debugging and testing**. The AWS SAM CLI lets you locally build, test, and debug serverless applications that are defined by AWS SAM templates. The CLI provides a Lambda-like execution environment locally.
- **Deep integration with development tools**. You can use AWS SAM with a suite of AWS tools for building serverless applications.

https://aws.amazon.com/serverless/sam/

Finally … CDK & SAM demo



https://sanderknape.com/2019/05/building-serverless-applications-aws-cdk/