



Willem de Kooning: Interchange (303 000 000 \$)



Bare Metal



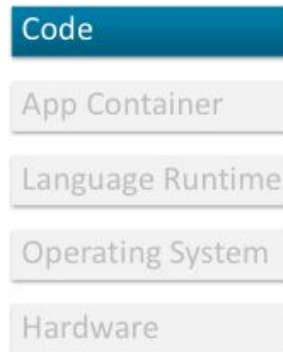
Virtual machines



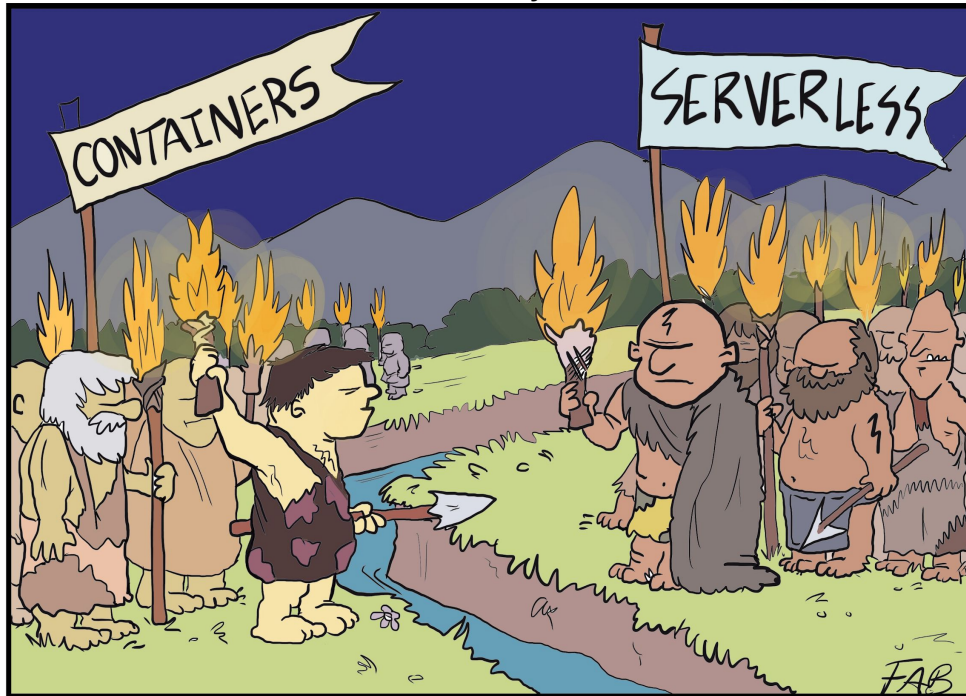
Containers



Functions



FaaS and Furious by Forrest Brazeal



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

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.

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 [keep functions warm](#) or provisioned concurrency (new 2019).

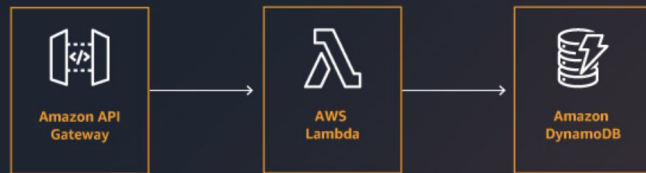
Heavier reliance on vendor ecosystems

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.

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.



Find a service by name or feature (for example, EC2, S3 or VM, storage).

Group A-Z

Compute <ul style="list-style-type: none">EC2LightSailECRECSEKSLambdaBatchElastic BeanstalkServerless Application RepositoryAWS OutpostsEC2 Image Builder	Customer Enablement <ul style="list-style-type: none">AWS IQSupportManaged Services Blockchain <ul style="list-style-type: none">Amazon Managed Blockchain Satellite <ul style="list-style-type: none">Ground Station Quantum Technologies <ul style="list-style-type: none">Amazon Braket Management & Governance <ul style="list-style-type: none">AWS OrganizationsCloudWatchAWS Auto ScalingCloudFormationCloudTrailConfigOpsWorksService CatalogSystems ManagerAWS AppConfigTrusted AdvisorControl TowerAWS License ManagerAWS Well-Architected ToolPersonal Health DashboardAWS ChatbotLaunch WizardAWS Compute Optimizer Media Services <ul style="list-style-type: none">Elastic TranscoderKinesis Video StreamsMediaConnectMediaConvertMediaLiveMediaPackageMediaStoreMediaTailorElemental Appliances & Software	Machine Learning <ul style="list-style-type: none">Amazon SageMakerAmazon CodeGuruAmazon ComprehendAmazon ForecastAmazon Fraud DetectorAmazon KendraAmazon LexAmazon Machine LearningAmazon PersonalizeAmazon PollyAmazon RekognitionAmazon TexttractAmazon TranscribeAmazon TranslateAWS DeepLensAWS DeepRacerAmazon Augmented AI Analytics <ul style="list-style-type: none">AthenaEMRCloudSearchElasticsearch ServiceKinesisQuickSightData PipelineAWS Data ExchangeAWS GlueAWS Lake FormationMSK Security, Identity, & Compliance <ul style="list-style-type: none">IAMResource Access ManagerCognitoSecrets ManagerGuardDutyInspectorAmazon MacieAWS Single Sign-OnCertificate ManagerKey Management ServiceCloudHSMDirectory ServiceWAF & ShieldArtifactSecurity HubDetective Mobile <ul style="list-style-type: none">AWS AmplifyMobile HubAWS AppSyncDevice Farm AR & VR <ul style="list-style-type: none">Amazon Sumerian	Application Integration <ul style="list-style-type: none">Step FunctionsAmazon EventBridgeAmazon MQSimple Notification ServiceSimple Queue ServiceSWF AWS Cost Management <ul style="list-style-type: none">AWS Cost ExplorerAWS BudgetsAWS Marketplace Subscriptions Customer Engagement <ul style="list-style-type: none">Amazon ConnectPinpointSimple Email Service Business Applications <ul style="list-style-type: none">Alexa for BusinessAmazon ChimeWorkMail End User Computing <ul style="list-style-type: none">WorkSpacesAppStream 2.0WorkDocsWorkLink Internet Of Things <ul style="list-style-type: none">IoT CoreAmazon FreeRTOSIoT 1-ClickIoT AnalyticsIoT Device DefenderIoT Device ManagementIoT EventsIoT GreengrassIoT SiteWiseIoT Things Graph Game Development <ul style="list-style-type: none">Amazon GameLift
--	--	---	--

Storage

- S3
- EFS
- FSx
- S3 Glacier
- Storage Gateway
- AWS Backup

Database

- RDS
- DynamoDB
- ElastiCache
- Neptune
- Amazon Redshift
- Amazon QLDB
- Amazon DocumentDB
- Managed Cassandra Service

Migration & Transfer

- AWS Migration Hub
- Application Discovery Service
- Database Migration Service
- Server Migration Service
- AWS Transfer for SFTP
- Snowball
- DataSync

Networking & Content Delivery

- VPC
- CloudFront
- Route 53
- API Gateway
- Direct Connect
- AWS App Mesh
- AWS Cloud Map
- Global Accelerator

Developer Tools

- CodeStar
- CodeCommit
- CodeBuild
- CodeDeploy
- CodePipeline
- Cloud9
- X-Ray

Robotics

- AWS RoboMaker

Serverless

Database



Amazon
DynamoDB

Gateways



Amazon
API Gateway

Security



AWS
IAM



AWS
KMS

Messaging and Queues



Amazon
SQS



Amazon
SNS

Compute



AWS Lambda

Storage



Amazon S3

Network



Amazon
VPC



Amazon
Route 53



Elastic Load
Balancing

Content Delivery



Amazon
CloudFront

Streaming Analytics



Amazon Kinesis

User Management



Amazon Cognito

Internet of Things



AWS IoT

Monitoring & Logging



Amazon
CloudWatch

Machine Learning

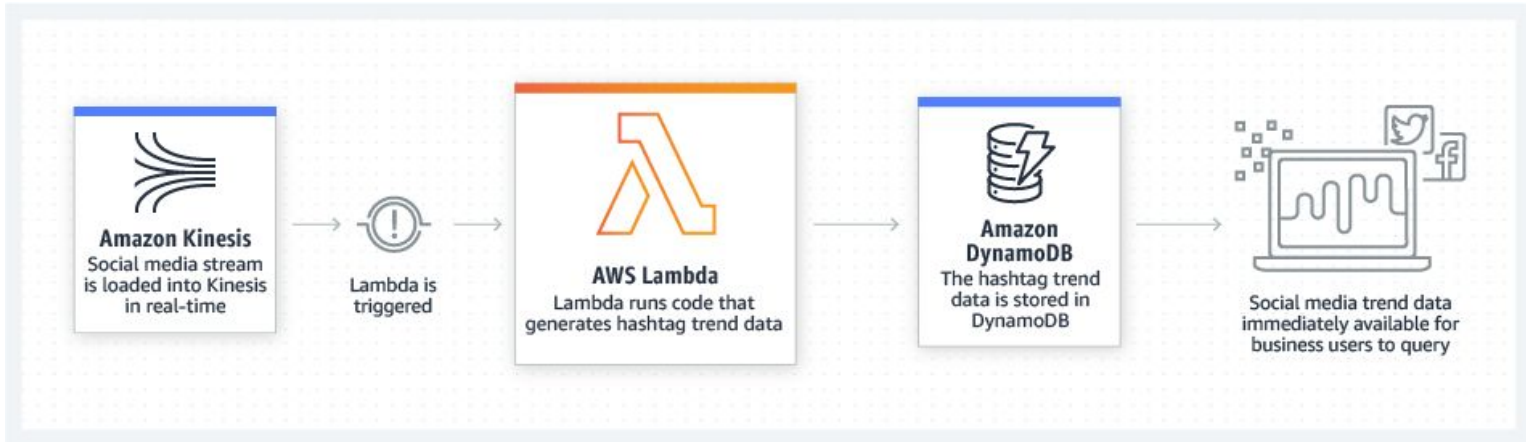
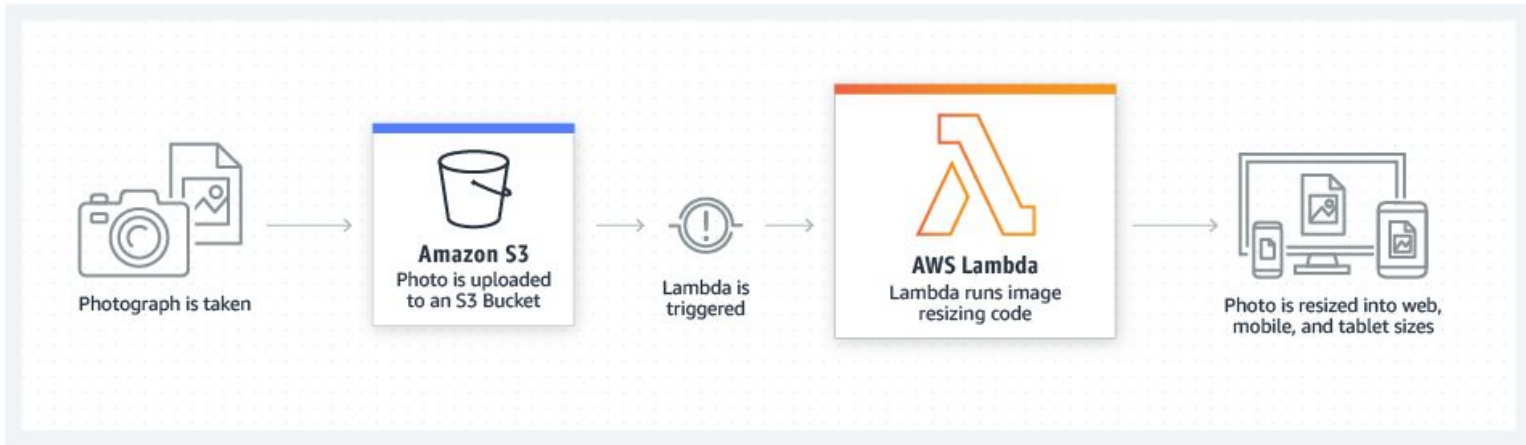


Amazon
Machine Learning

Web applications and backends



Data processing



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
<code>/tmp</code> directory storage	512 MB
File descriptors	1,024
Execution processes/threads	1,024

AWS Lambda Pricing

Region: EU (Frankfurt) ↕

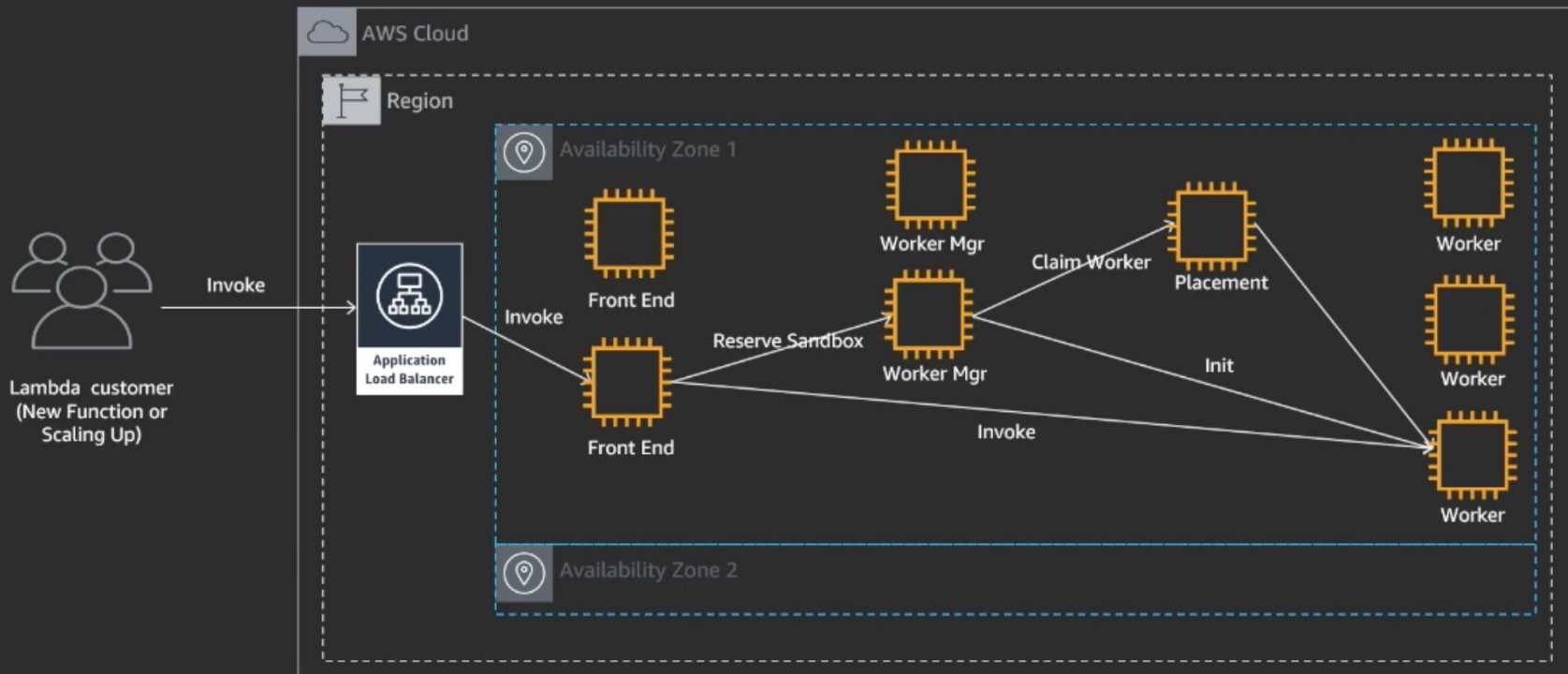
Price	
Requests	\$0.20 per 1M requests
Duration	\$0.000016667 for every GB-second

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



Your Workload

Lambda Runtime

Guest Kernel

KVM

Firecracker

Host Kernel

EC2 Instance

Dedicated
to your
function

What Does KVM Do?

- Programs the host CPU to do secure hardware virtualization (HVM)
- Low-level virtualization details
 - Memory management, paging, etc
- Abstracts hardware details

What Does Firecracker Do?

- Configures KVM
- Device emulation
 - Pretends to be an SSD, NIC, etc.
- Performance isolation
- Optimized for serverless
 - Fast moving, low overhead

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

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.

Finally ... CDK & SAM demo

