MENTORMATE™
DevCamp

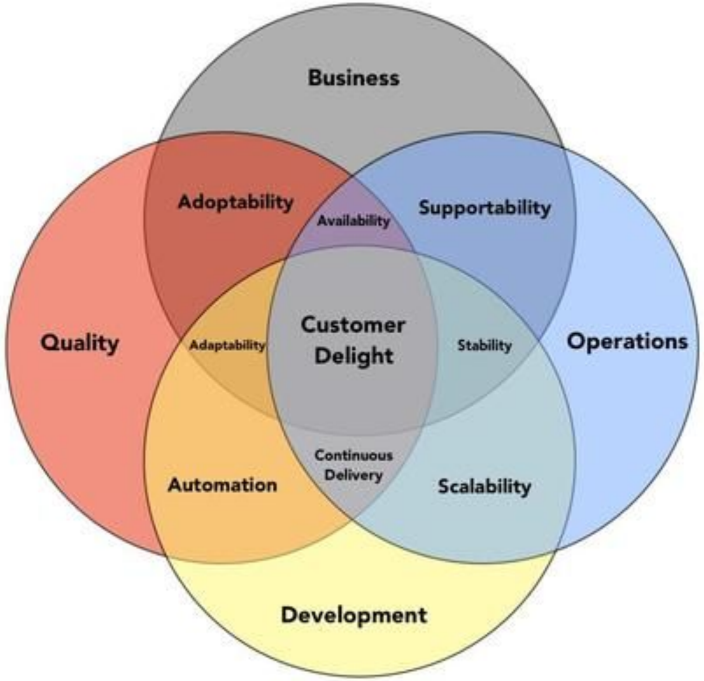
Building Cloud-Native Applications

Yulia Tenincheva

Senior Cloud Engineer, MentorMate

What is expected of you?

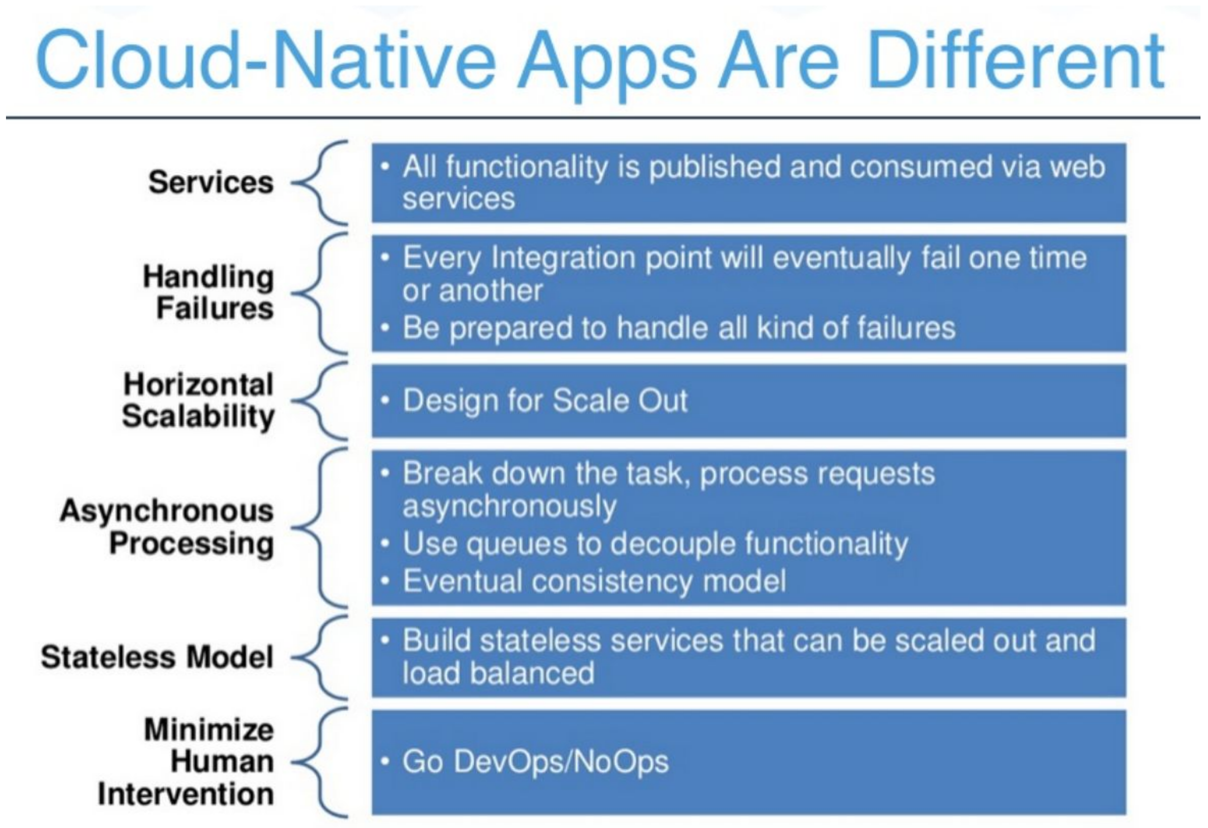
- To know how to develop efficient and secure Node.js applications that solve the business need
- To know how to use AWS SDK to use some of the more popular cloud services
- To adopt a Cloud Mindset
- To understand the Serverless architecture
- To continue learning and growing your skillset
- And if you get AWS Certified one day - I will be extremely proud of you :D



MENTORMATE™ DevCamp

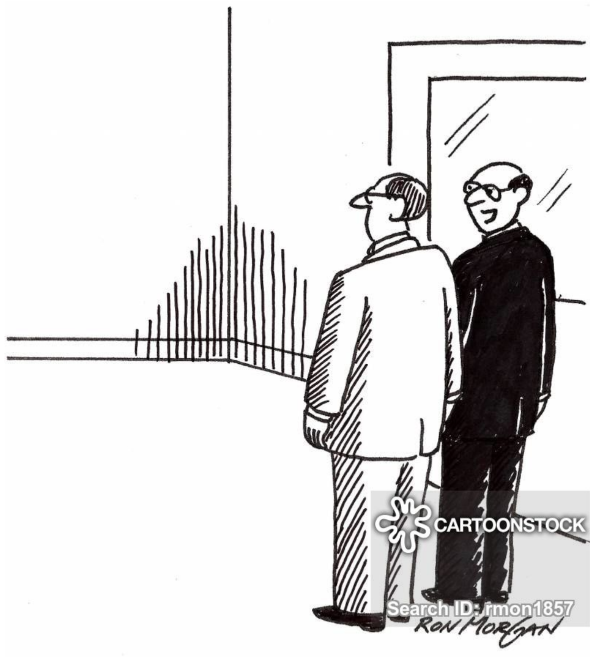
Cloud Native

Cloud native is an approach to building and running applications that fully exploit the advantages of the cloud computing model.



Cloud-Native Apps Are Different

Services	• All functionality is published and consumed via web services
Handling Failures	• Every Integration point will eventually fail one time or another • Be prepared to handle all kind of failures
Horizontal Scalability	• Design for Scale Out
Asynchronous Processing	• Break down the task, process requests asynchronously • Use queues to decouple functionality • Eventual consistency model
Stateless Model	• Build stateless services that can be scaled out and load balanced
Minimize Human Intervention	• Go DevOps/NoOps

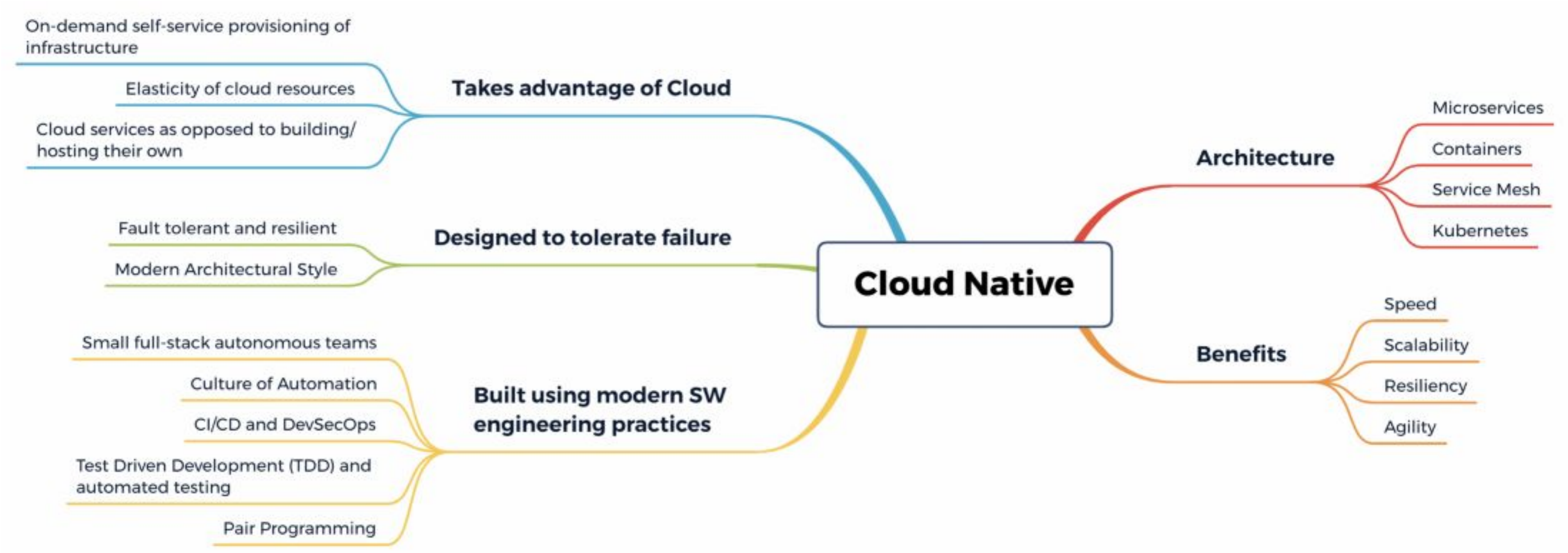


"Everything is in the Cloud."

"Everything fails all the time"

~ Werner Vogels, Vice President & CTO at Amazon.com

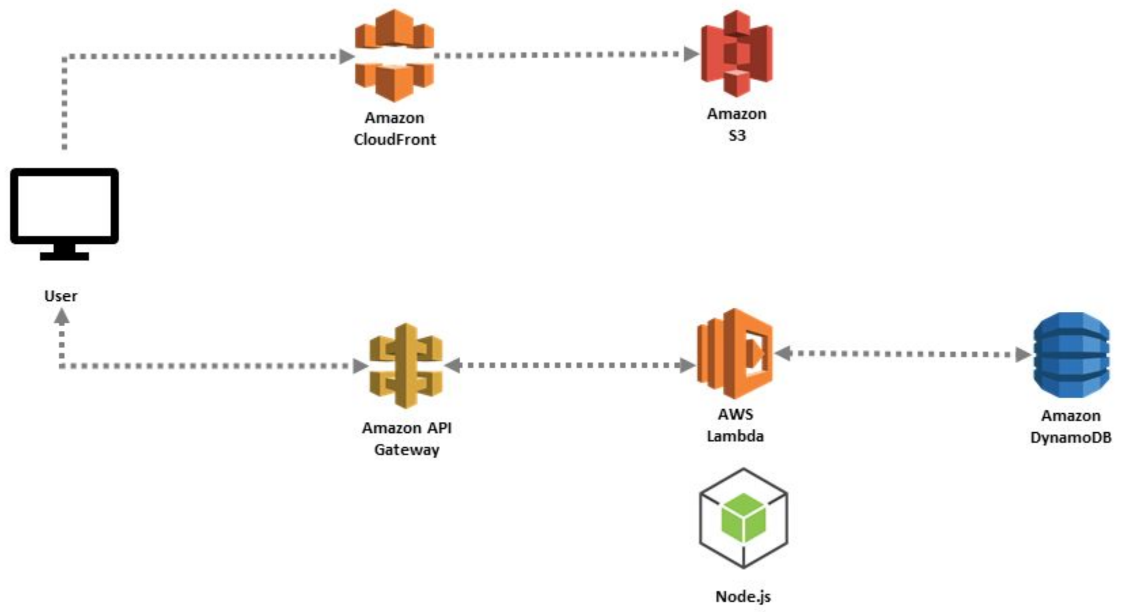
Cloud-Native Design Patterns / Mindset


Cloud Native
Takes advantage of Cloud
On-demand self-service provisioning of infrastructure
Elasticity of cloud resources
Cloud services as opposed to building/ hosting their own
Designed to tolerate failure
Fault tolerant and resilient
Modern Architectural Style
Built using modern SW engineering practices
Small full-stack autonomous teams
Culture of Automation
CI/CD and DevSecOps
Test Driven Development (TDD) and automated testing
Pair Programming
Architecture
Microservices
Containers
Service Mesh
Kubernetes
Benefits
Speed
Scalability
Resiliency
Agility

Cloud-Native Design Patterns / Mindset

- Cloud-Native apps need rapid scale
- At scale failures are inevitable
- Let apps handle own resiliency
- Build stateless services
- Scale out, not scale up
- Treat infrastructure differently
- Create immutable infrastructures
- Adopt Microservices Architecture

Base AWS Services for Serverless



S3

S3 (Simple Storage Service)

Amazon Simple Storage Service (Amazon S3) is an **object storage service** that offers industry-leading scalability, data availability, security, and performance.

Customers can use it **to store and protect any amount of data** for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics.

Amazon S3 is designed for **99.999999999%** (11 9's) of **durability**, and stores data for millions of applications for companies all around the world.

"The Cloud ate my homework."

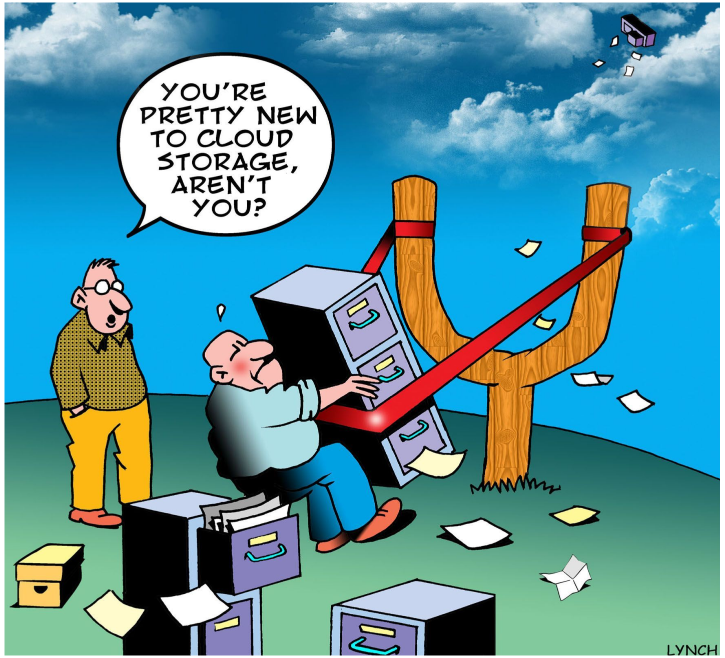
MENTORMATE™ DevCamp

Concepts of S3

- Bucket
  - Container for objects stored in S3
  - Unlimited size
  - Organize the namespace at the highest level
  - Internet accessible storage via HTTP/s
  - Global unique name
  - Limit of 100 buckets per account
- Key
  - Name of the object
  - Unique identifier for an object within a bucket
  - Use the object key to retrieve the object
- Object
  - Similar to files
  - No hierarchy
  - Objects are immutable
  - Size up to 5 TB
  - Uniquely identified within a bucket by a key(name) and a version ID

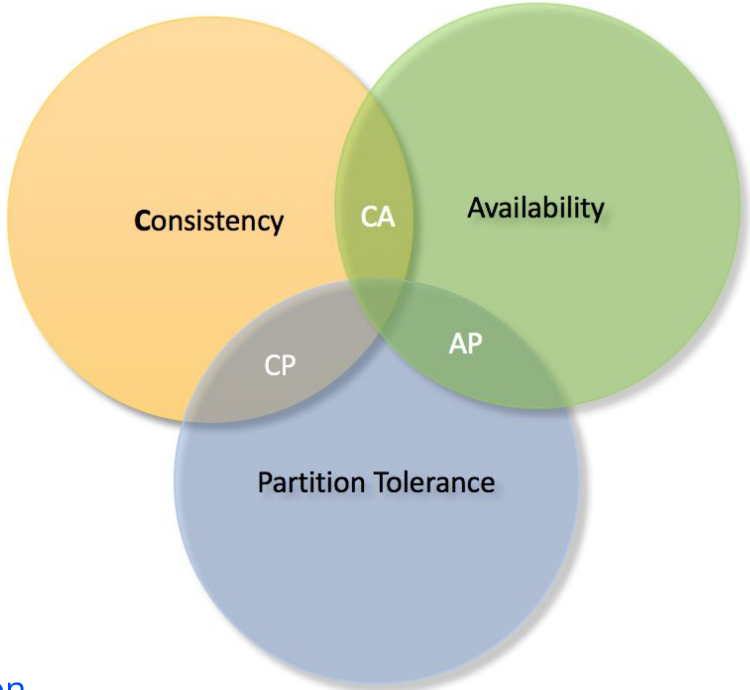
How to Access S3

- AWS CLI
  - aws s3 help
- REST API
  - Get / Put Object
- AWS SDK
  - getObject, getSignedUrl, headObject, ...
  - putObject, upload, uploadPart, ...
- Web Console
  - Like a monkey



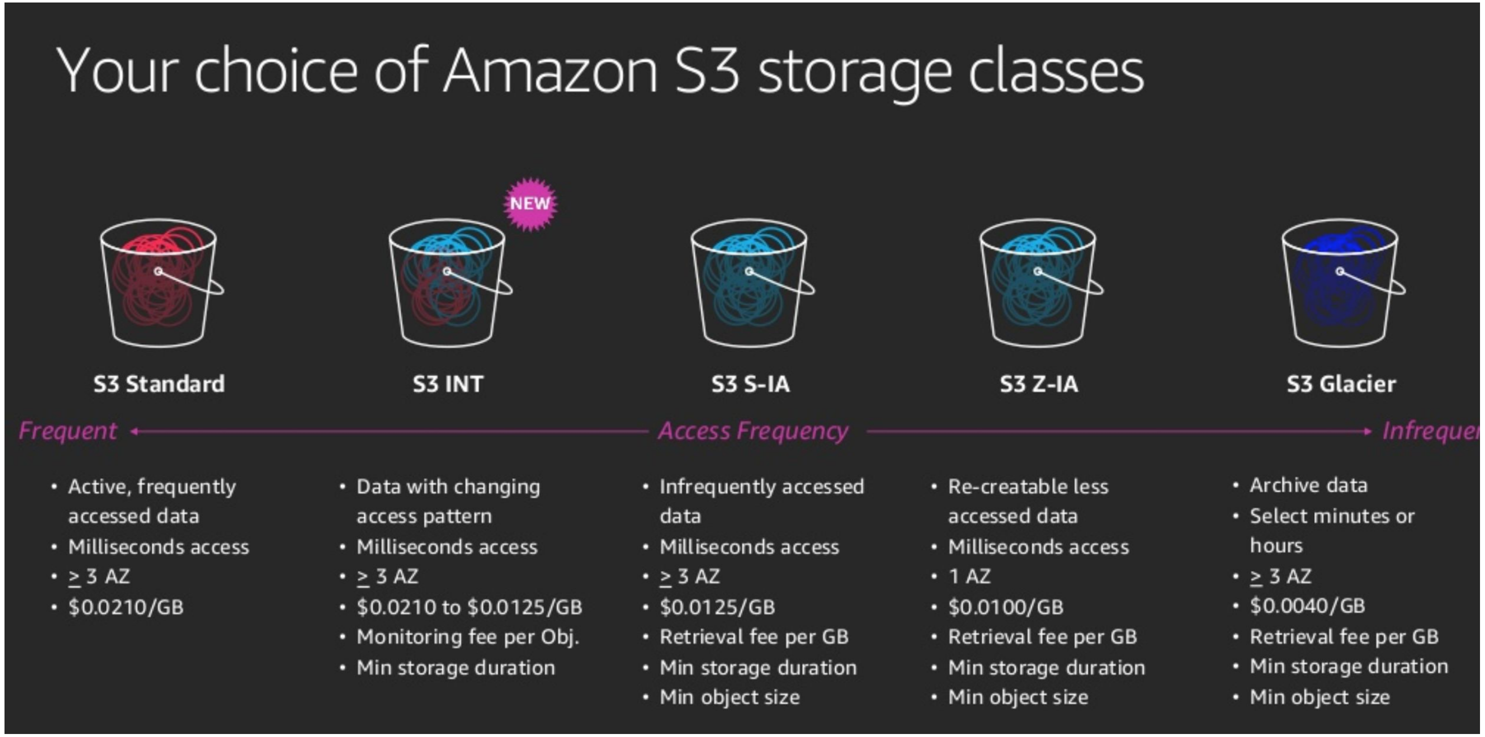
CAP Theorem

- Amazon S3 (AP)
  - **High availability**
  - **High durability** (99.999999999%)
  - **Eventual consistency** for overwrite and deletes
  - **Read-after-write consistency** for new uploads



AWS S3 Consistency model in details & quick explanation

S3 Storage Classes



Your choice of Amazon S3 storage classes

Frequent ← *Access Frequency* → *Infreque*

S3 Standard

- Active, frequently accessed data
- Milliseconds access
- ≥ 3 AZ
- $0.0210/GB

S3 INT

- Data with changing access pattern
- Milliseconds access
- ≥ 3 AZ
- $0.0210 to $0.0125/GB
- Monitoring fee per Obj.
- Min storage duration

S3 S-IA

- Infrequently accessed data
- Milliseconds access
- ≥ 3 AZ
- $0.0125/GB
- Retrieval fee per GB
- Min storage duration
- Min object size

S3 Z-IA

- Re-creatable less accessed data
- Milliseconds access
- 1 AZ
- $0.0100/GB
- Retrieval fee per GB
- Min storage duration
- Min object size

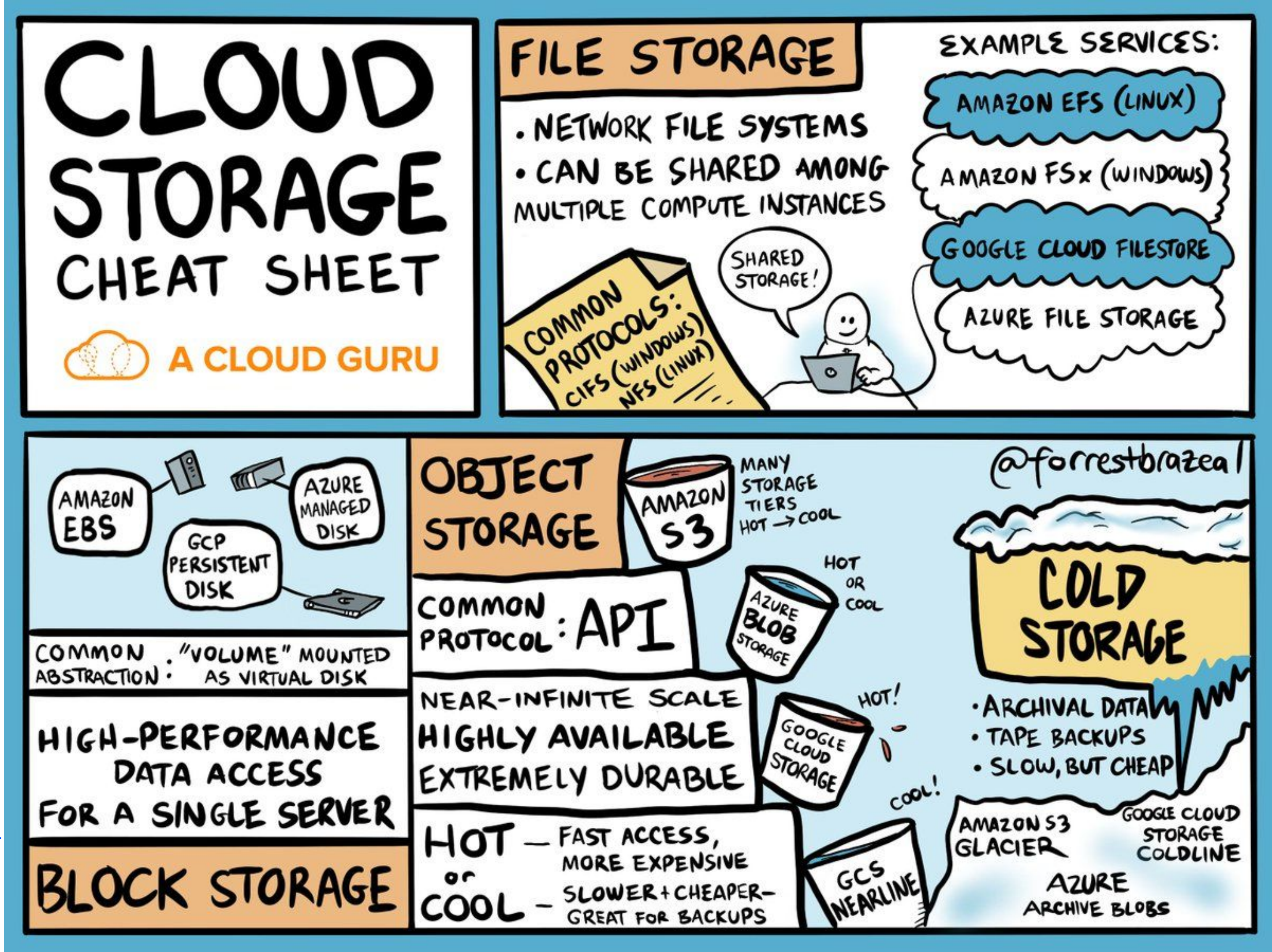
S3 Glacier

- Archive data
- Select minutes or hours
- ≥ 3 AZ
- $0.0040/GB
- Retrieval fee per GB
- Min storage duration
- Min object size

MENTORMATE™ DevCamp

S3 Features

- Versioning
- Static Website Hosting
- Cross-Region Replication
- Lifecycle Management
- Encryption
- Security - ACLs and Bucket Policies



CLOUD STORAGE CHEAT SHEET

A CLOUD GURU

FILE STORAGE

- NETWORK FILE SYSTEMS
- CAN BE SHARED AMONG MULTIPLE COMPUTE INSTANCES

COMMON PROTOCOLS: CIFS (WINDOWS) NFS (LINUX)

SHARED STORAGE!

EXAMPLE SERVICES:

- AMAZON EFS (LINUX)
- AMAZON FSx (WINDOWS)
- GOOGLE CLOUD FILESTORE
- AZURE FILE STORAGE

BLOCK STORAGE

AMAZON EBS

GCP PERSISTENT DISK

AZURE MANAGED DISK

COMMON ABSTRACTION: "VOLUME" MOUNTED AS VIRTUAL DISK

HIGH-PERFORMANCE DATA ACCESS FOR A SINGLE SERVER

OBJECT STORAGE

AMAZON S3 — MANY STORAGE TIERS HOT → COOL

AZURE BLOB STORAGE — HOT OR COOL

GOOGLE CLOUD STORAGE — HOT!

GCS NEARLINE — COOL!

COMMON PROTOCOL: API

NEAR-INFINITE SCALE
HIGHLY AVAILABLE
EXTREMELY DURABLE

HOT — FAST ACCESS, MORE EXPENSIVE

or

COOL — SLOWER + CHEAPER — GREAT FOR BACKUPS

@forrestbrazeal

COLD STORAGE

- ARCHIVAL DATA
- TAPE BACKUPS
- SLOW, BUT CHEAP

AMAZON S3 GLACIER

GOOGLE CLOUD STORAGE COLDLINE

AZURE ARCHIVE BLOBS

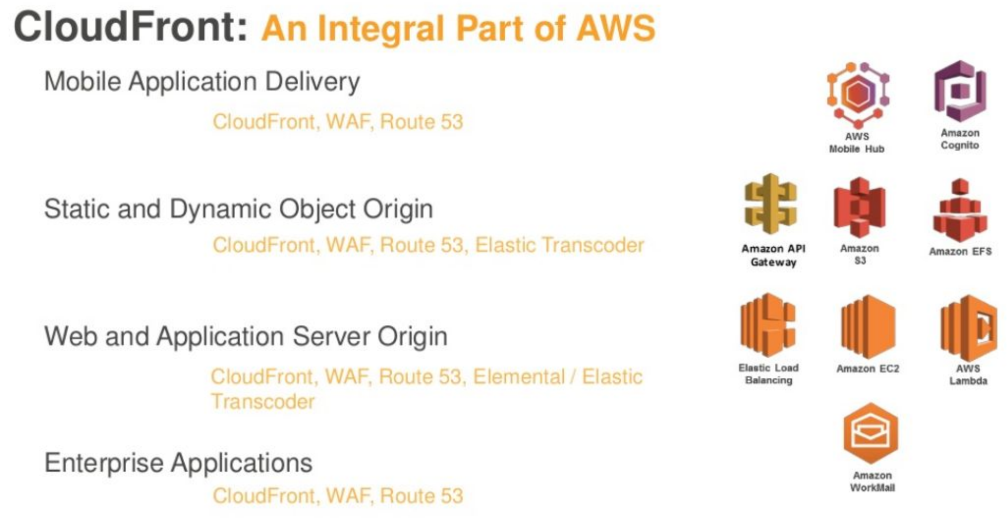
Static WebSite Hosting in S3 - Demo

MENTORMATE™
DevCamp

AWS CloudFront

What is a CDN and why use one?

- Global Content Delivery Network with mas capacity and scale
- Optimized for Performance and scale
- Built-in Security features
- Robust Real Time reporting
- Static and Dynamic object, video delivery



CloudFront: An Integral Part of AWS

Mobile Application Delivery
CloudFront, WAF, Route 53

Static and Dynamic Object Origin
CloudFront, WAF, Route 53, Elastic Transcoder

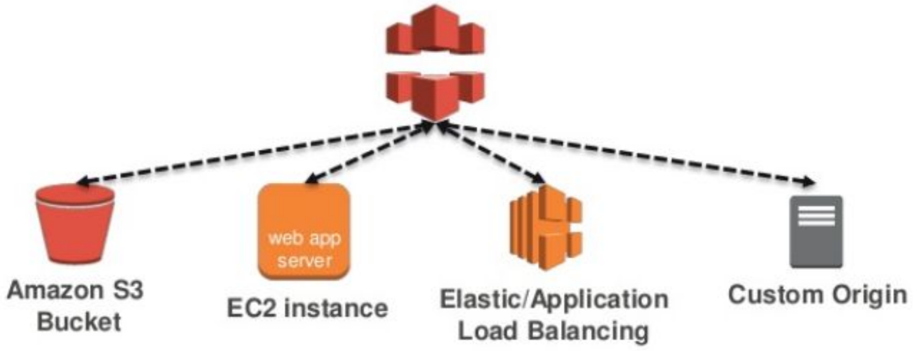
Web and Application Server Origin
CloudFront, WAF, Route 53, Elemental / Elastic Transcoder

Enterprise Applications
CloudFront, WAF, Route 53

AWS Mobile Hub
Amazon Cognito
Amazon API Gateway
Amazon S3
Amazon EFS
Elastic Load Balancing
Amazon EC2
AWS Lambda
Amazon WorkMail

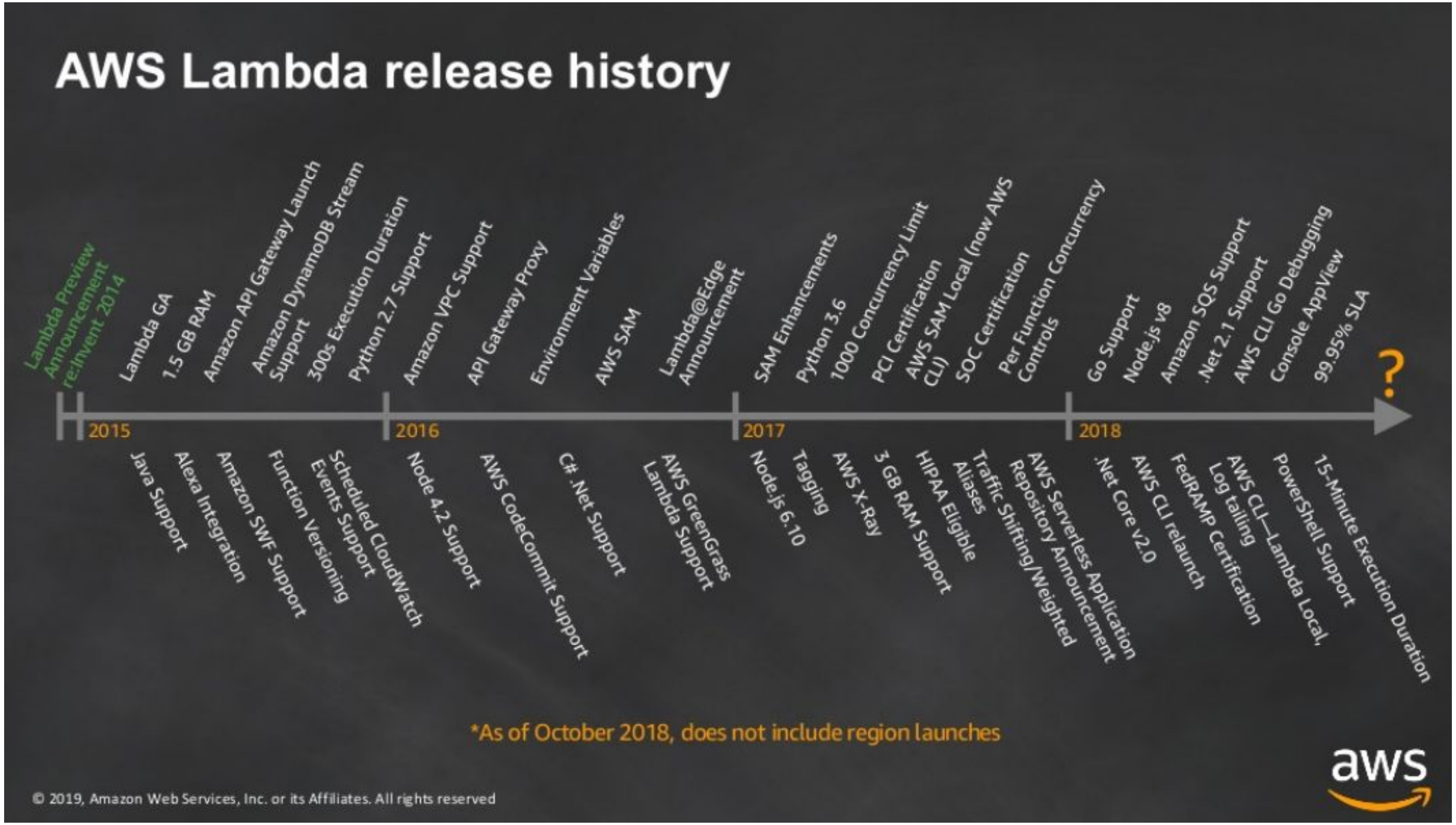
CloudFront Service Components

- Distributions
- Origins
- Behaviors
- Restrictions, Error pages, Tags
- AWS WAF Web ACLs
- Edge Locations
- Price Classes



AWS Lambda

MENTORMATE™
DevCamp



AWS Lambda release history

Lambda Preview Announcement re:Invent 2014

2015

- Lambda GA
- Java Support
- 1.5 GB RAM
- Alexa Integration
- Amazon API Gateway Launch
- Amazon SWF Support
- Amazon DynamoDB Stream Support
- Function Versioning
- 300s Execution Duration
- Scheduled CloudWatch Events Support
- Python 2.7 Support

2016

- Amazon VPC Support
- Node 4.2 Support
- API Gateway Proxy
- AWS CodeCommit Support
- Environment Variables
- C# .Net Support
- AWS SAM
- AWS GreenGrass Lambda Support
- Lambda@Edge Announcement

2017

- SAM Enhancements
- Node.js 6.10
- Python 3.6
- Tagging
- 1000 Concurrency Limit
- AWS X-Ray
- PCI Certification
- 3 GB RAM Support
- AWS SAM Local (now AWS CLI)
- HIPAA Eligible
- SOC Certification
- Aliases Traffic Shifting/Weighted
- Per Function Concurrency Controls
- AWS Serverless Application Repository Announcement

2018

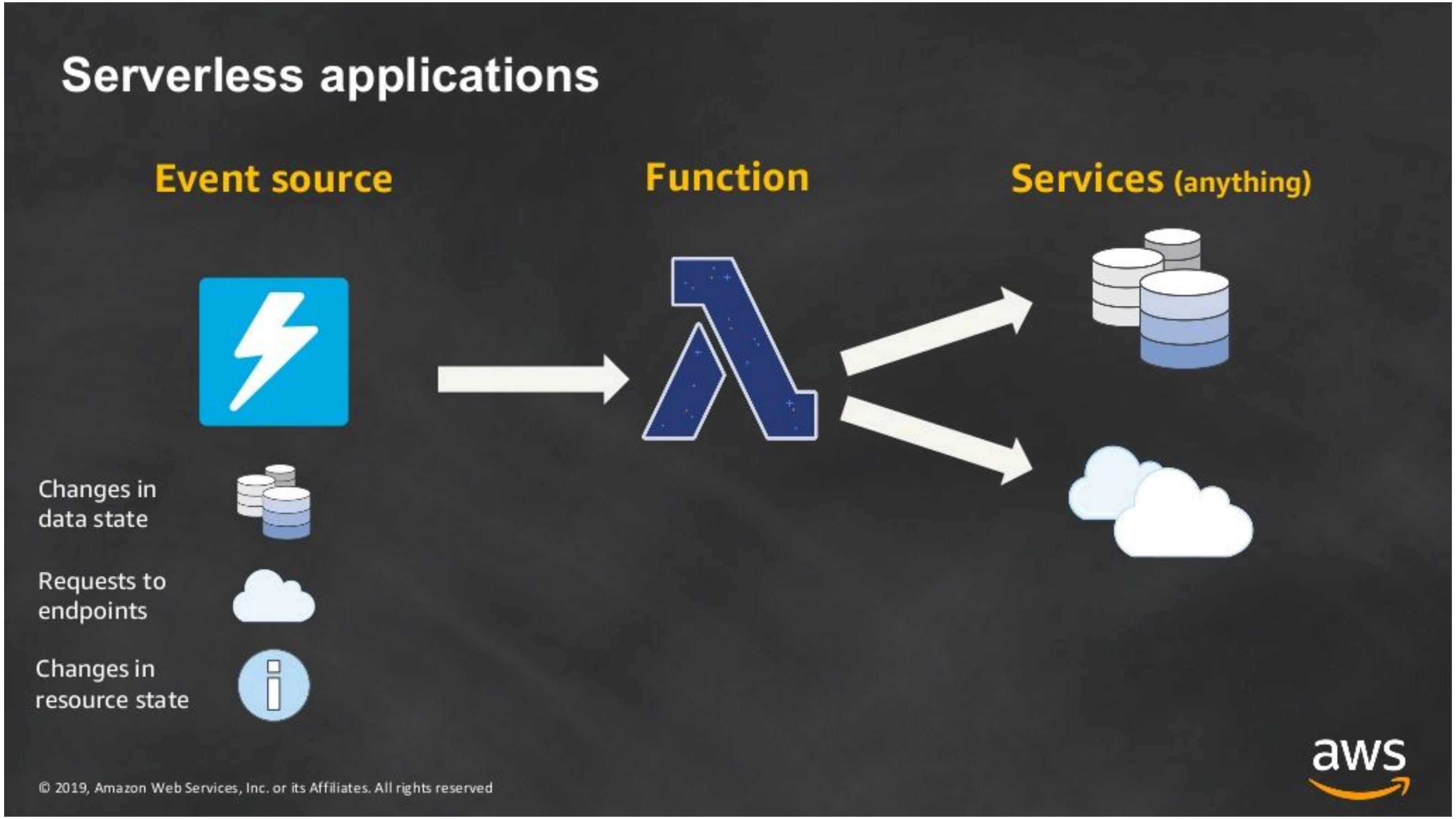
- Go Support
- .Net Core v2.0
- Node.js v8
- AWS CLI relaunch
- Amazon SQS Support
- FedRAMP Certification
- .Net 2.1 Support
- Log tailing
- AWS CLI Go Debugging
- AWS CLI—Lambda Local,
- Console AppView
- PowerShell Support
- 99.95% SLA
- 15-Minute Execution Duration

?

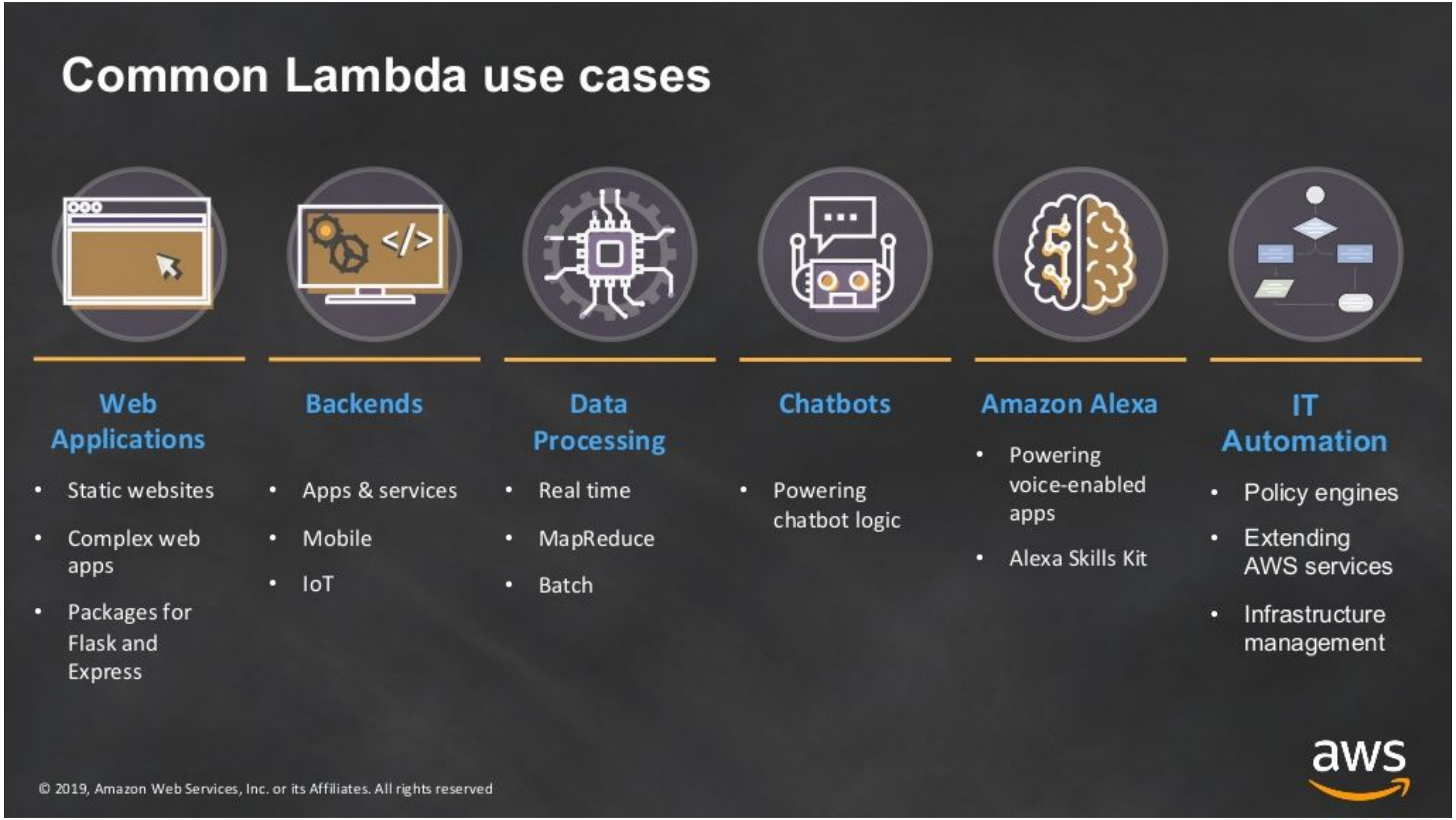
*As of October 2018, does not include region launches

aws

MENTORMATE™ DevCamp



Serverless applications



Common Lambda use cases

Web Applications
- Static websites
- Complex web apps
- Packages for Flask and Express

Backends
- Apps & services
- Mobile
- IoT

Data Processing
- Real time
- MapReduce
- Batch

Chatbots
- Powering chatbot logic

Amazon Alexa
- Powering voice-enabled apps
- Alexa Skills Kit

IT Automation
- Policy engines
- Extending AWS services
- Infrastructure management

Use cases

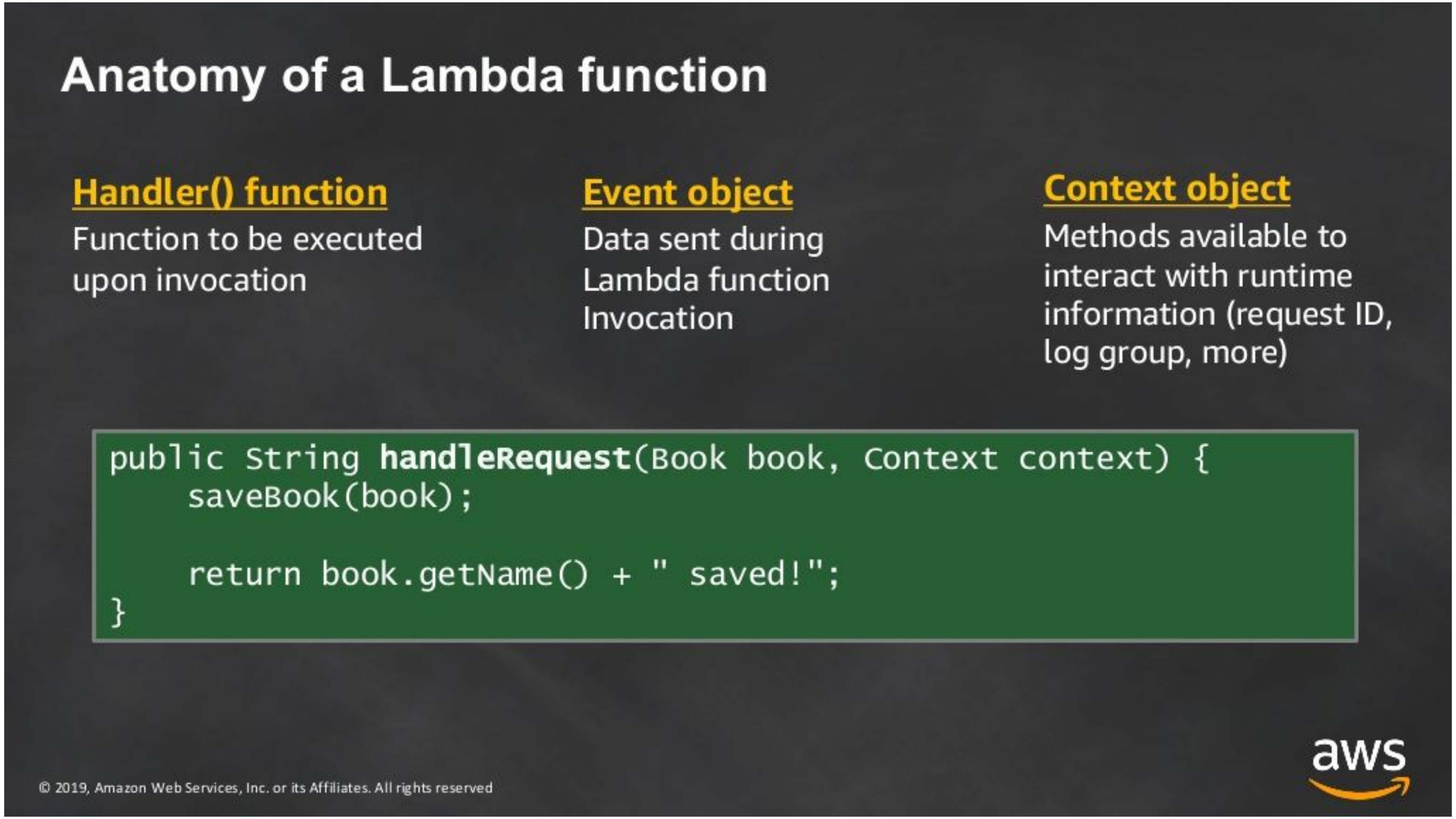
- When to use AWS Lambda
  - Working on aws resources directly
  - Building small one off applications
  - When you need to process something after an event that occurs on AWS

- When **NOT** to use AWS Lambda
  - When operating system resources are needed
  - When native libraries are needed (e.g ffmpeg)
  - When the application is large and complex
  - When you need lots of memory
  - When you need lots of execution time

Use cases

- Questionable
  - When your function is very complex
  - When your function takes an unknown amount of time to finish

- When in doubt:
  - Get familiar with the things you will be using
  - Build a PoC (Proof of concept) in a small and controlled way



Anatomy of a Lambda function

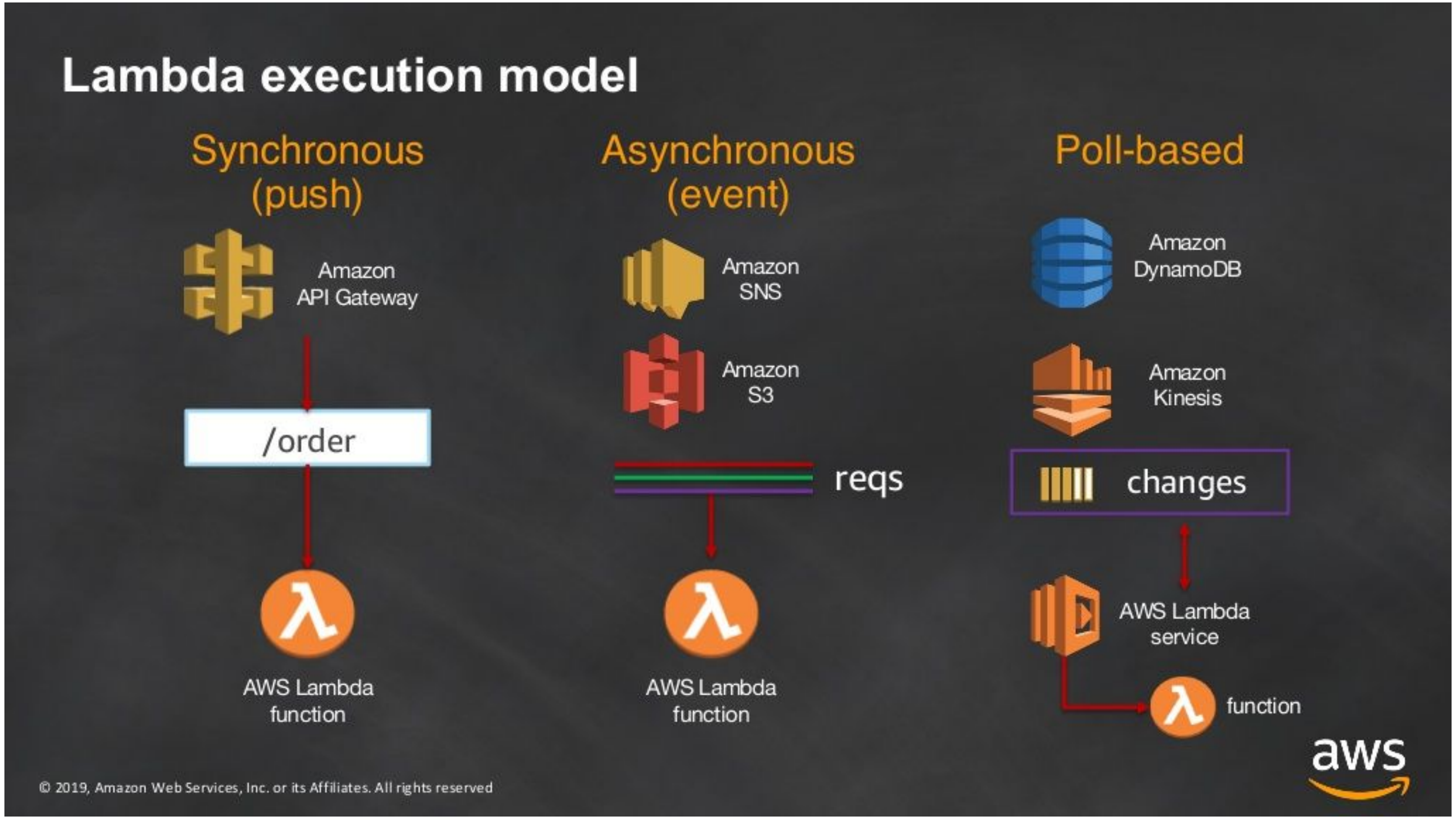
Handler() function
Function to be executed upon invocation

Event object
Data sent during Lambda function Invocation

Context object
Methods available to interact with runtime information (request ID, log group, more)

```
public String handleRequest(Book book, Context context) {
    saveBook(book);

    return book.getName() + " saved!";
}
```


Lambda execution model
Synchronous (push)
Amazon API Gateway
/order
AWS Lambda function
Asynchronous (event)
Amazon SNS
Amazon S3
reqs
AWS Lambda function
Poll-based
Amazon DynamoDB
Amazon Kinesis
changes
AWS Lambda service
function
aws

Source of these Slides and full lecture:

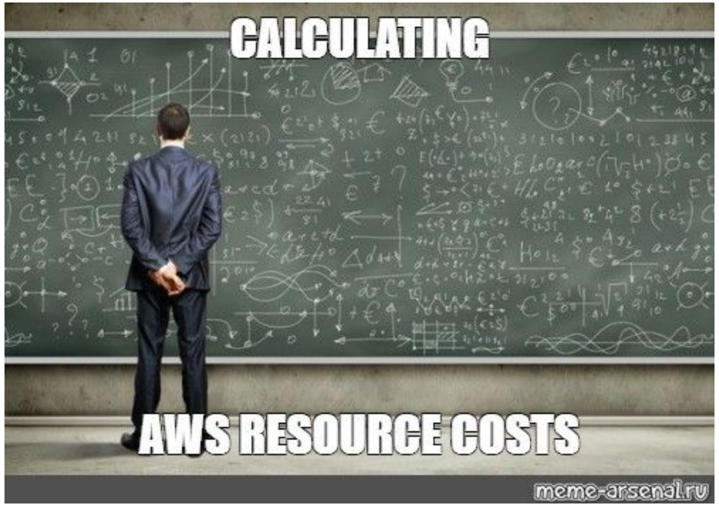
https://www.youtube.com/watch?v=EBSdyoO3goc&ab_channel=AmazonWebServices

AWS Lambda Best Practices

- Limit your function size
- Node.js - remember executions is asynchronous
- Don’t assume function container reuse - but take advantage of it when it does occur
- Don’t forget about disk (500Mb /tmp directory provided to each function)
- Create custom metrics (operations-centric and business-centric

One more thing

- AWS Lambda Pricing
  - The AWS Lambda **free usage** tier includes **1M free requests per month** and 400,000 **GB-seconds** of compute time per month.
  - In the AWS Lambda resource model, you choose the amount of memory you want for your function, and are allocated proportional CPU power and other resources. An increase in memory size triggers an equivalent increase in CPU available to your function.

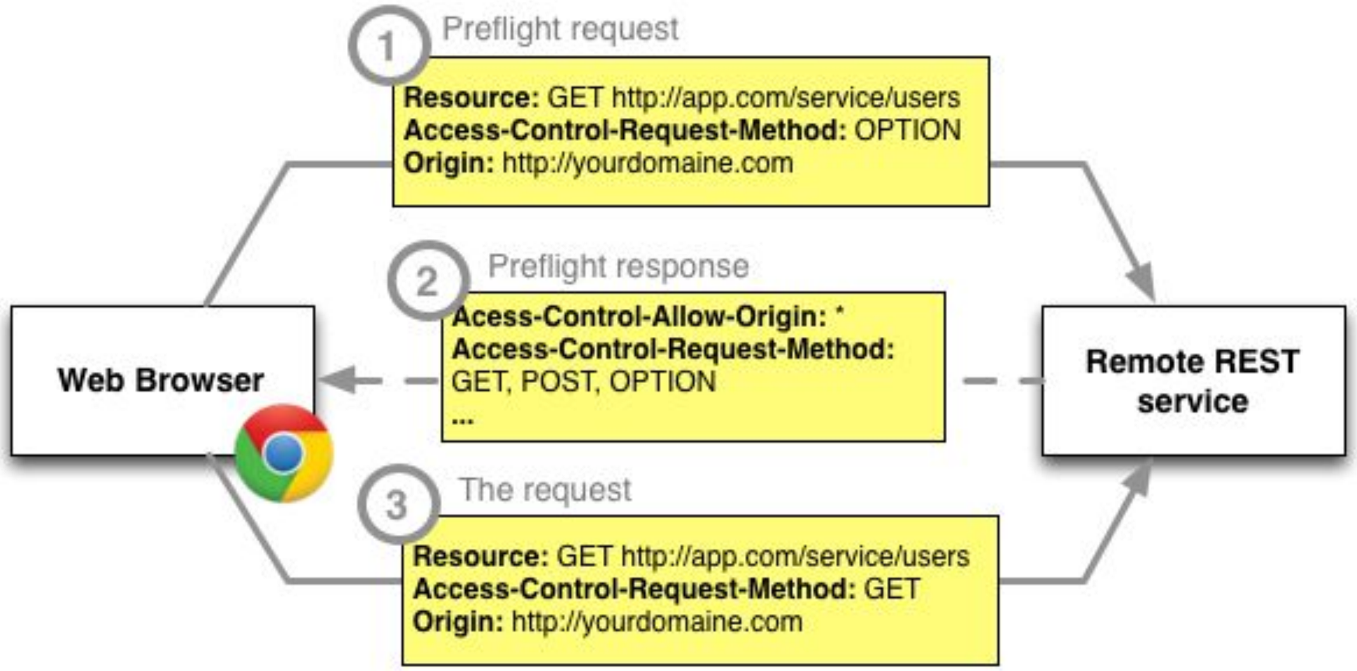


Wire S3 with AWS Lambda via API Gateway (Demo)

MENTORMATE™
DevCamp

CORS

CORS


1
Preflight request
Resource: GET http://app.com/service/users
Access-Control-Request-Method: OPTION
Origin: http://yourdomaine.com
2
Preflight response
Acess-Control-Allow-Origin: *
Access-Control-Request-Method:
GET, POST, OPTION
...
3
The request
Resource: GET http://app.com/service/users
Access-Control-Request-Method: GET
Origin: http://yourdomaine.com
Web Browser
Remote REST service

- Everything you would like to know about Cross Origin Resource Sharing - here

Homework

MENTORMATE™
DevCamp

Homework

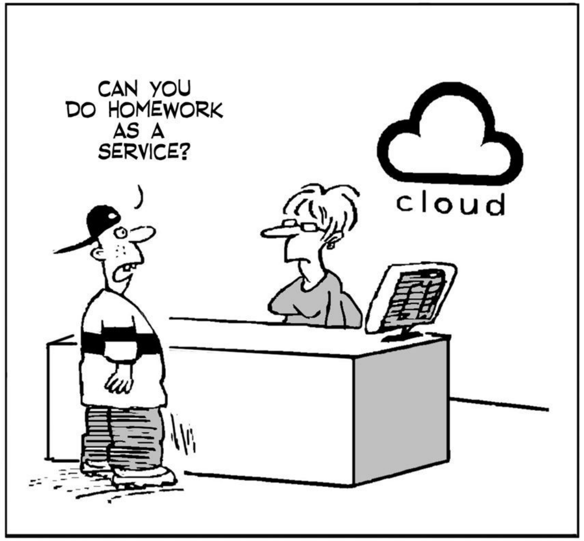
1. Host a static web site in your bucket (it can be a plain “hello world”)

2. Create a web server with a single endpoint POST /upload. Use Postman as a client to invoke and *test* your endpoint. In your server you should write a handler that processes the multipart/form-data request and re-upload that file to S3.

   Tricky parts & considerations:

- Keep your access/secret keys… secret!
  Do not expose them in any way!
- Make sure that the permissions of the file does not allow public access to it.
- Test your API with bigger files. Is it working efficiently? Consider implementing multipart uploads.
- Can you apply streams and pipes here? If so, do it.

Final requirement: When the file upload is ready and successful, return to the client a *signed url* of the uploaded file that. That URL should expire after 15 minutes.



MENTORMATE™
DevCamp

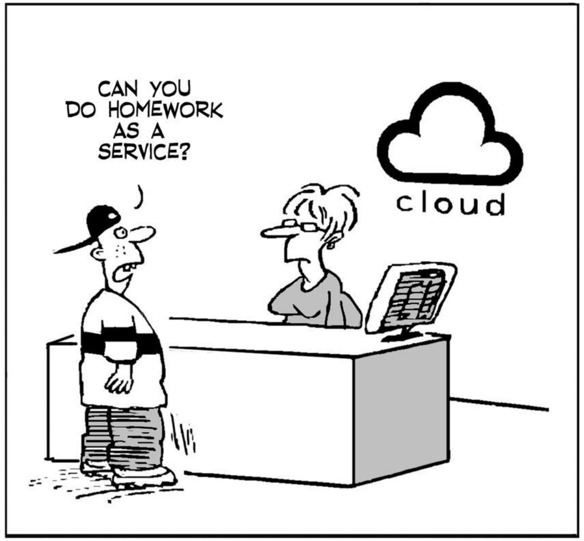
Homework

3. (Optional) Following this example, create an AWS Lambda function that resizes images. Make it so that function is triggered automatically whenever a new file is uploaded to your bucket source "*folder*".

Name your AWS Lambda function uniquely, so it is easy to identify you as the author. Configure IAM permissions of that Lambda in such a way, so it can access your bucket only.

Important: Write the result to another "*folder*" in S3 to avoid circular invocations (infinite loop)!!

If you do this exercise, you will learn how to import NPM modules to your Node.js Lambda functions, how to use the AWS CLI to deploy your code, how to configure fine-grained access and you will also learn how to configure asynchronous events.



4. (NOT a homework!! This is for Cloud enthusiasts only) - Wild Rydes hands-on workshop* for serverless architecture.
*You can use S3 + CloudFront, instead of AWS Amplify.
Please, destroy the created resources when you are done with the workshop.