Cloud Computing

ECPE 276

AWS Hosted Services

Compute

Compute Options

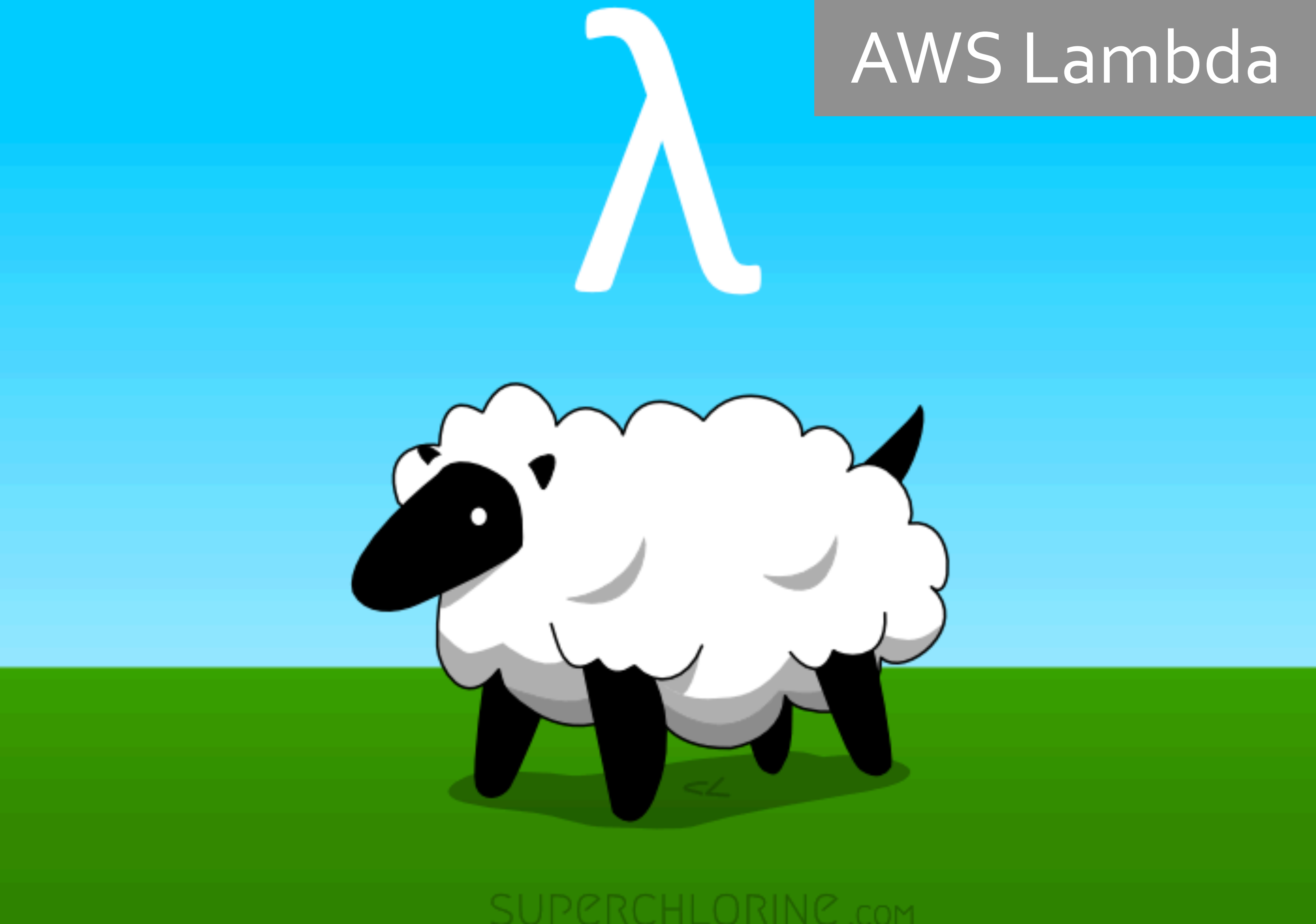
1. Amazon **Elastic Compute Cloud** (EC2)
2. Amazon **Lambda** (λ)
3. Amazon **Elastic Bean Stalk** (EBS)
4. Other services
    1. Elastic Load Balancing
    2. Auto Scaling
    3. CloudFront (content delivery)

EC2

Amazon Elastic Compute Cloud

- Marketing
  - Infinite supply of servers, on-demand
  - Rent by the hour
- You supply
  - The operating system (or use standard Amazon images)
  - The software stack
  - The application

AWS Lambda



AWS Lambda

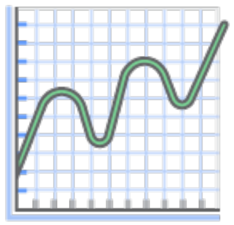
- **Event-driven coding**
- You write a custom Lambda function
  - Node.js, Java, or Python
- Amazon runs your function automatically upon event
  - File uploaded to S3 bucket
  - DynamoDB record changed
  - HTTP Request (Amazon API gateway or your own)

AWS Lambda - Marketing

- Sub-second metering ($$) (in 100ms)
  - (Charged flat rate per access
    plus actual compute time incurred)

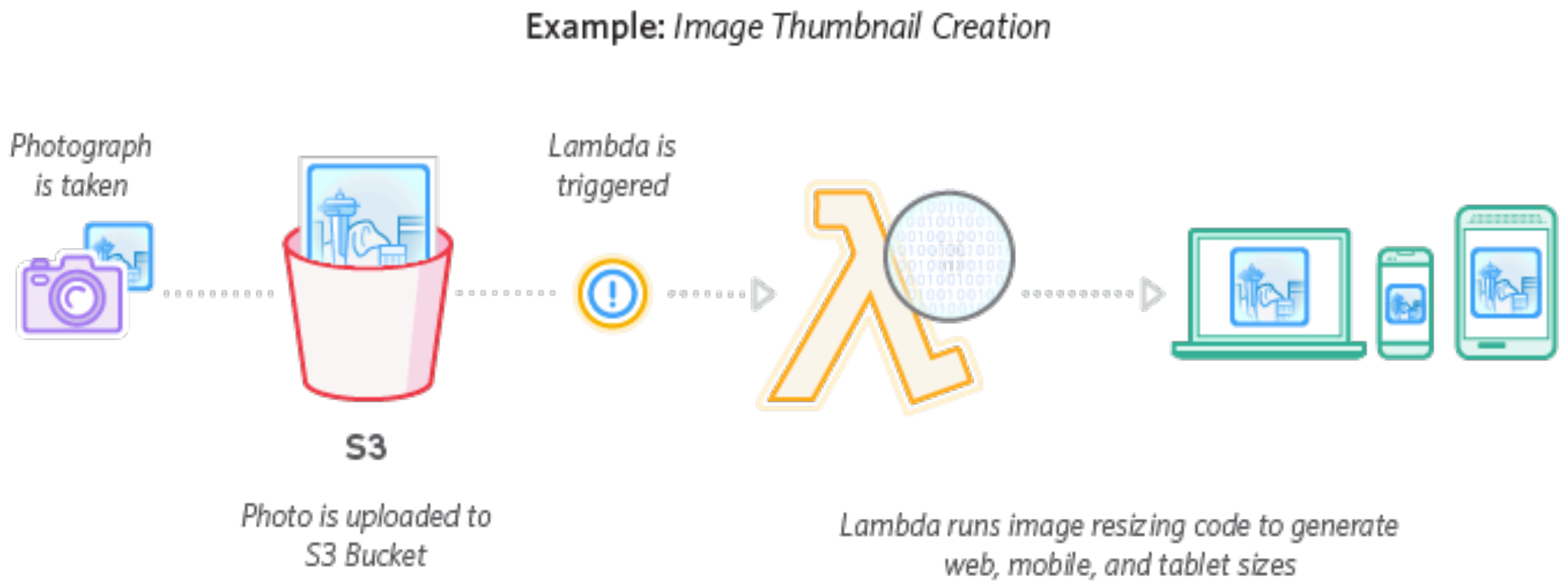
- No servers to manage

- Continuous scaling

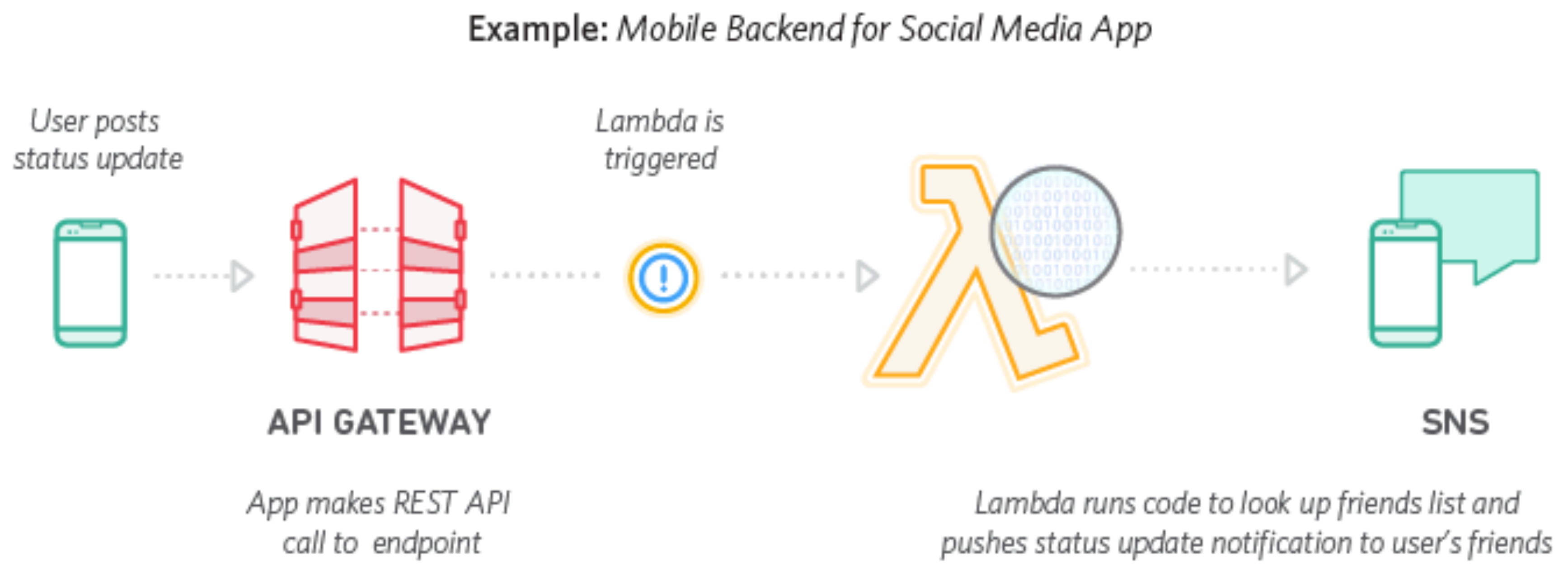


https://www.youtube.com/watch?v=eOBq__h4OJ4

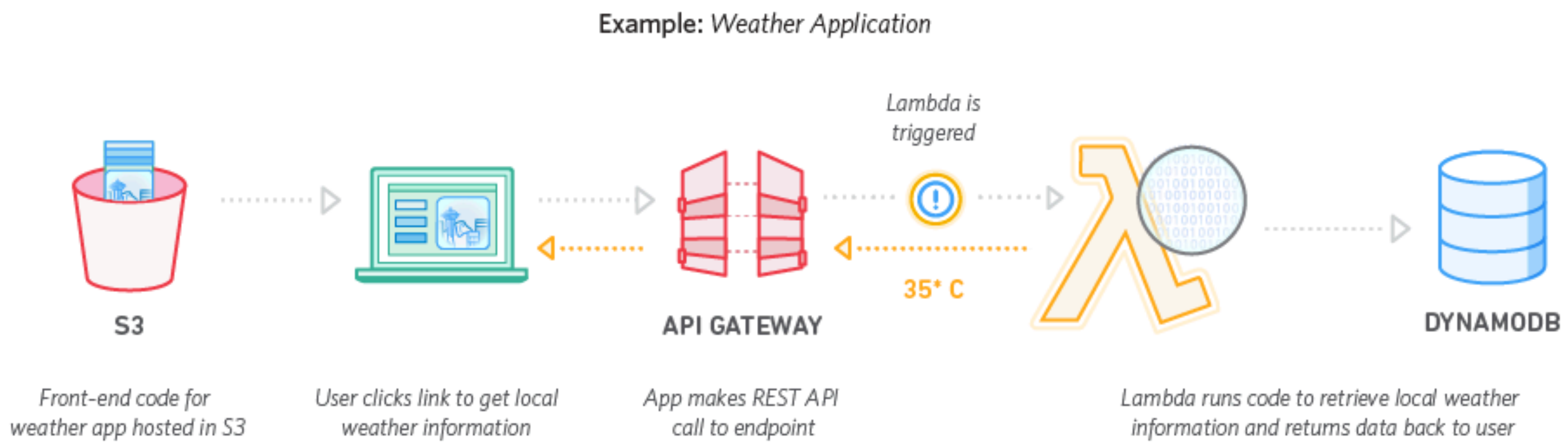
AWS Lambda – Example Use Case


Example: *Image Thumbnail Creation*
Photograph is taken
S3
Photo is uploaded to S3 Bucket
Lambda is triggered
Lambda runs image resizing code to generate web, mobile, and tablet sizes

AWS Lambda – Example Use Case


Example: *Mobile Backend for Social Media App*
User posts status update
Lambda is triggered
API GATEWAY
SNS
App makes REST API call to endpoint
Lambda runs code to look up friends list and pushes status update notification to user's friends

AWS Lambda – Example Use Case


Example: *Weather Application*
Lambda is triggered
35° C
S3
API GATEWAY
DYNAMODB
Front-end code for weather app hosted in S3
User clicks link to get local weather information
App makes REST API call to endpoint
Lambda runs code to retrieve local weather information and returns data back to user

AWS Elastic Beanstalk

AWS Elastic Beanstalk

- **Application Hosting**
  - *Specifically, web sites or web apps*
- You provide *only* the application *(one-click deployment!)*
  - AWS Management Console (manually upload)
  - Git repository (update app in *seconds!*)
  - IDE on local machine (Eclipse or Visual Studio)
- Amazon provides
  - Infrastructure!
  - Servers, databases, load balancers, firewalls, networks, etc…

AWS Elastic Beanstalk - Marketing

- Automatic everything!
  - Capacity provisioning
  - Load balancing (among pool of servers)
  - Auto-scaling (up and down)
  - Application health monitoring
- Full control of underlying infrastructure and ability to modify *if desired*

https://www.youtube.com/watch?v=SrwxAScdyT0

AWS Elastic Beanstalk - Environments

- **Supported environments are limited** because Amazon provides the entire software stack
  - Apache Tomcat for Java applications
  - Apache HTTP Server for PHP applications
  - Apache HTTP Server for Python applications
  - Nginx or Apache HTTP Server for Node.js applications
  - Passenger or Puma for Ruby applications
  - Microsoft IIS 7.5, 8.0, and 8.5 for .NET applications
  - Java SE
  - Docker
  - Go

Targeting web sites or web apps

AWS Elastic Beanstalk - Control

Targeted at programmer, not sysadmin

- Programmer options / programmer controls
  - Operating system (Linux, Windows, specific releases)
  - Database
  - Directly login to EC2 instances for immediate troubleshooting
  - Run in more than one Availability Zone (reliability)
  - HTTPS on load balancer
  - Amazon CloudWatch monitoring (cluster health and events)
  - Adjust application server settings (e.g. JVM settings) and pass environment variables
  - Run other application components, such as a memory caching service, side-by-side in Amazon EC2
  - Access log files without logging in to the application servers

Other Services – Elastic Load Balancing

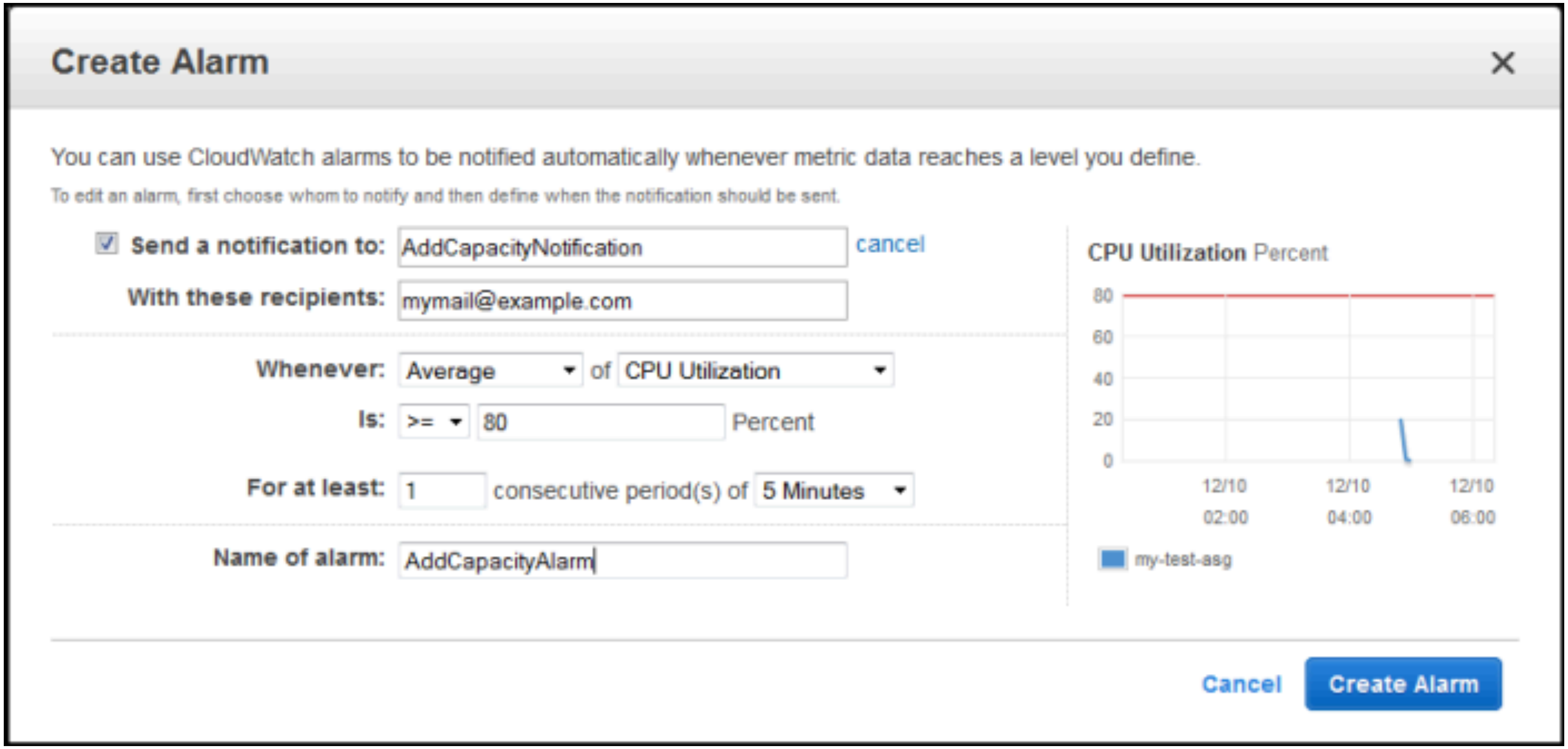
- Amazon provides load balancers
- Accept incoming HTTP, HTTPS, SSL, and generic TCP requests
- Forward request to multiple EC2 instances across different availability zones
- Improves reliability by distinguishing between healthy and unhealthy targets
  - Application health-check (e.g. try to load a special page from your website)

Design: Never have customers directly connect to EC2 instance – they always go through load balancer!

Other Services – Auto Scaling

- Launch or terminate EC2 nodes based on current demand
  - CPU utilization?
  - Available RAM?
  - Disk utilization? (% full? Read/Write bandwidth?)
  - Network utilization?
  - Many other AWS *CloudWatch* metrics (or custom)
- Launch new nodes to replace failed ones
- Coordinates with load balancer
  - New nodes? Notify load balancer
  - Terminating nodes? Notify load balancer

Other Services – Auto Scaling



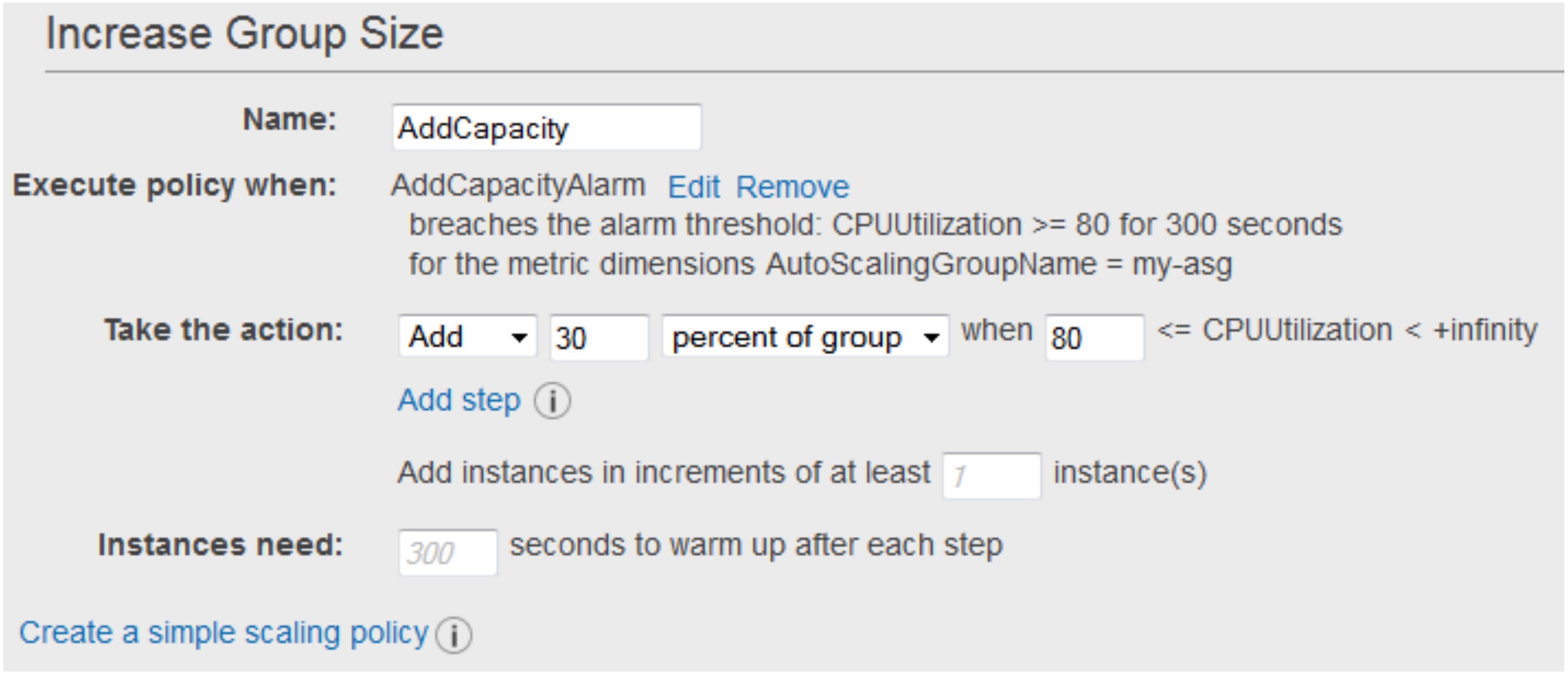
Create Alarm

You can use CloudWatch alarms to be notified automatically whenever metric data reaches a level you define.

To edit an alarm, first choose whom to notify and then define when the notification should be sent.

☑ Send a notification to:	AddCapacityNotification
With these recipients:	mymail@example.com
Whenever:	Average of CPU Utilization
Is:	>= 80 Percent
For at least:	1 consecutive period(s) of 5 Minutes
Name of alarm:	AddCapacityAlarm

Other Services – Auto Scaling



Increase Group Size

Name: AddCapacity

Execute policy when: AddCapacityAlarm Edit Remove
breaches the alarm threshold: CPUUtilization >= 80 for 300 seconds
for the metric dimensions AutoScalingGroupName = my-asg

Take the action: Add 30 percent of group when 80 <= CPUUtilization < +infinity

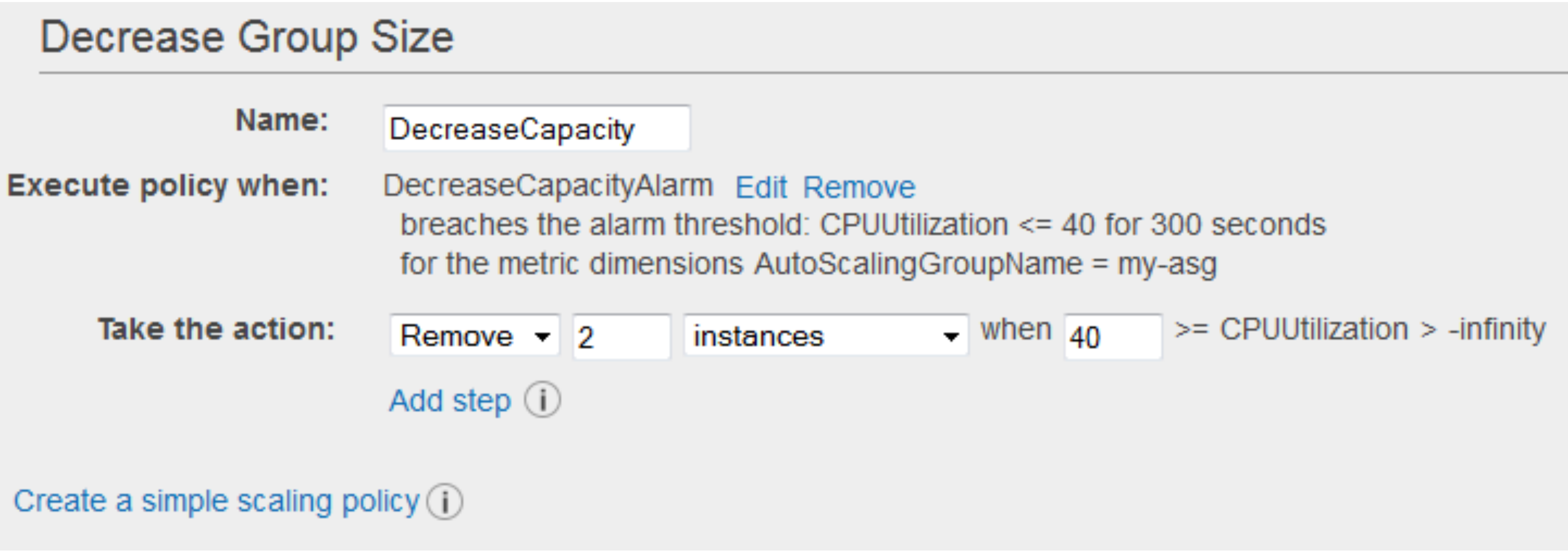
Add step

Add instances in increments of at least 1 instance(s)

Instances need: 300 seconds to warm up after each step

Create a simple scaling policy

Other Services – Auto Scaling



Decrease Group Size

Name: DecreaseCapacity

Execute policy when: DecreaseCapacityAlarm Edit Remove
breaches the alarm threshold: CPUUtilization <= 40 for 300 seconds
for the metric dimensions AutoScalingGroupName = my-asg

Take the action: Remove 2 instances when 40 >= CPUUtilization > -infinity

Other Services – CloudFront

- Global **Content Distribution Network** (CDN) for web content
- Single URL/domain can resolve to myriad local content servers (caches) close to customers
  - DNS trickery (routing based on latency or geography) via Amazon *Route53* DNS service
- Use cases:
  - Distribute static (infrequently changing) content
    - Images, style sheets, JavaScript
  - Distribute pre-recorded streaming media
  - Distribute live streaming events (short buffer in cache)

Networking

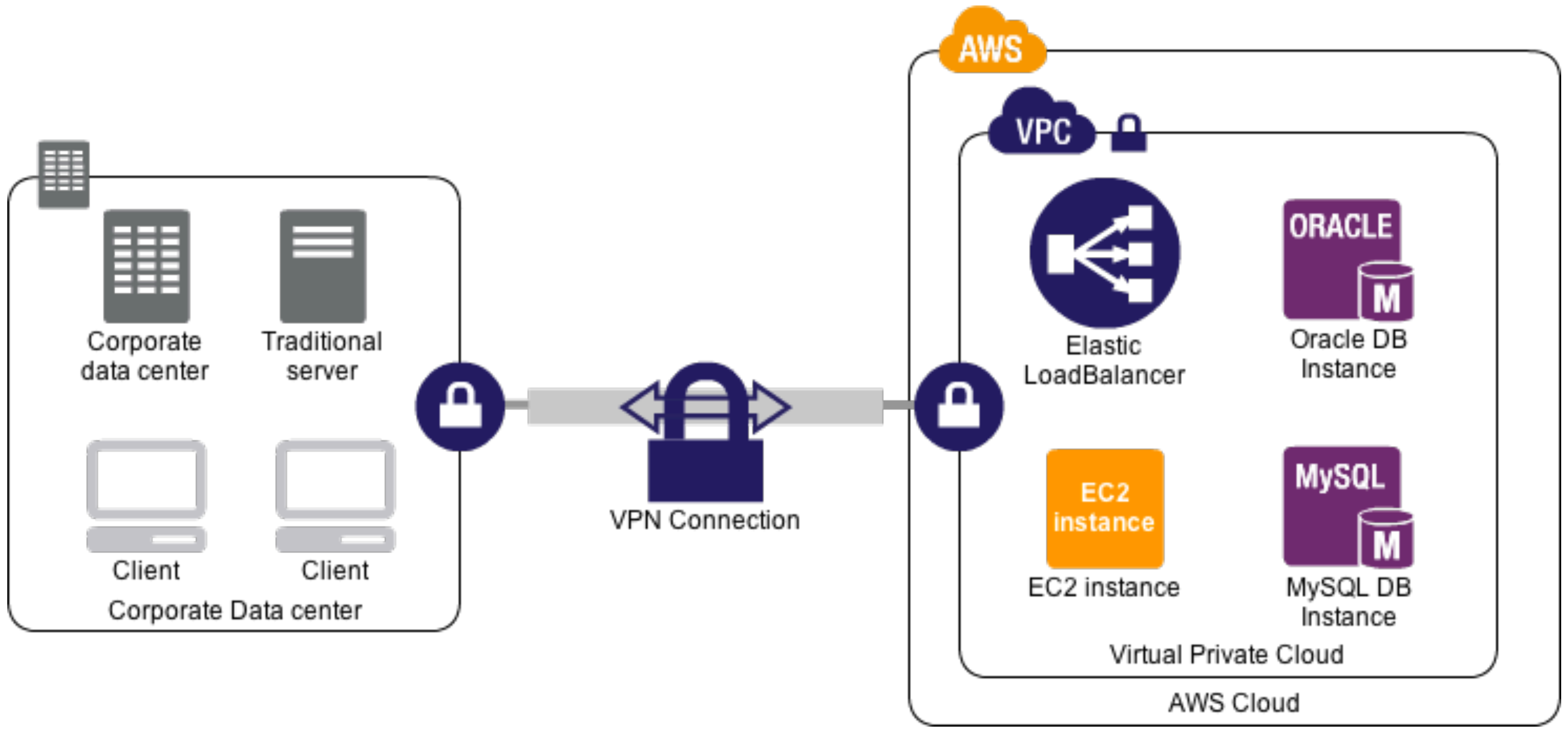
Networking

- Amazon **Virtual Private Cloud** (VPC)
- AWS **Direct Connect**
- Amazon **Route 53**

Amazon Virtual Private Cloud

- Virtual **networking**
  - Isolate your EC2 nodes in their own virtual network
  - Choose your own subnets, gateways, routes, address translation (NAT), firewall rules, …
- Use case: **Extend your corporate network** into the cloud
  - Hardware VPN brings AWS cloud servers inside your corporate firewall
- Use case: **Multi-tier applications**
  - Web servers in publically accessible subnet
  - Application servers and databases in private subnet

Amazon Virtual Private Cloud



Amazon Direct Connect

- Tired of waiting for your massive datasets to be sent over the **public Internet?**
- Got extra $$ burning a hole in your pocket?
- Solution: Amazon Direct Connect
  - **Private fiber** direct to Amazon's cloud
  - Reduced bandwidth price from Amazon
    - But you'll pay someone else for the fiber…
  - Better QOS / bandwidth?

Amazon Direct Connect – Peering

Your private fiber needs to reach a specific **network colocation facility** in one of these cities to join Amazon's network

"West"	"East"
New York City	Dublin
Northern Virginia	London
Dallas	Frankfurt
Las Vegas	Mumbai
San Francisco	Osaka
Seattle	Seoul
Sao Paulo	Sydney

Amazon Route 53

- **DNS Service:** "www.example.com" -> 1.2.3.4
- Marketing: "built using AWS's highly available and reliable infrastructure"
  - Servers across the globe
- Essential part of load balancing and scaling
  - Integrates with Amazon Elastic Load Balancer
- Fancy routing to best server based on
  - Server health check (still working?)
  - Latency
  - Geography
  - Round-robin

Communication / Coordination

Communication / Coordination

- Amazon **Simple Notification Service** (SNS)
- Amazon **Simple Queue Service** (SQS)

Amazon Simple Notification Service

- Messaging service for distributed clients
  - Publish/subscribe **push** model
    - A few nodes *publish* data to the service
    - Many other nodes *subscribe* to the service
  - Cloud pushes notifications to all clients immediately after publisher submits it
    - HTTP/HTTPS, email, SMS
- Client Platforms & Languages
  - Mobile: iOS, Android
  - Any of the AWS SDKs (Java, Python, PHP, Node.js, .NET)

Amazon Simple Queue Service

- Messaging service for distributed clients
  - **Pull** model
  - Clients send and receive data via cloud-based queues
  - Arbitrary data up to 256kB
  - Arbitrary senders (many!) and receivers (many!)
- API in AWS SDK
  - `SendMessage` (to a queue)
  - `ReceiveMessage` (from a queue)
  - `DeleteMessage` (from a queue)
  - `ChangeMessageVisibility` (of a previous message)
  - Batching (to reduce cost!)
  - Queue Management

Amazon Simple Queue Service - Lifecycle

1. System "A" needs to send a message
    1. Select an Amazon SQS queue
    2. Call `SendMessage()` to transfer to queue
2. System "Z" needs more messages to process
    1. Call `ReceiveMessage()`, and get message from System "A"
3. Once a message has been returned by `ReceiveMessage()`, it will not be returned by any other `ReceiveMessage()` until the visibility timeout has passed
    1. Keeps multiple computers from processing the same message at once
4. System "Z" successfully processes this message?
    1. Call `DeleteMessage()`
    2. Removes the message from the queue so no one else will ever process it
5. System "Z" fails to process the message?
    1. Message will be read by another `ReceiveMessage()` call as soon as the visibility timeout passes

Next Week: Netflix Case Study

NETFLIX