**What is AWS Lambda?**

From the [AWS documentation](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html" \t "_blank):

"AWS Lambda is a compute service that lets you run code **without provisioning or managing servers**."

* "AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second.
* You pay only for the compute time you consume - there is no charge when your code is not running."

"You can use AWS Lambda to run your code in response to events," such as HTTP requests.

**SQS Queue**

Amazon Simple Queue Service (Amazon SQS) offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components.

* A **queue** is just a type of list that orders data in a particular way; typically in a first-item-in = first-item-out order (FIFO), as shown below.

A picture containing text, clipart

Description automatically generated

Learn more about SQS [in the documentation](https://aws.amazon.com/sqs/" \t "_blank).

**MXNet and Lambda**

* Learn more about deploying models using MXNet and AWS Lambda in [this AWS blog post](https://aws.amazon.com/blogs/compute/seamlessly-scale-predictions-with-aws-lambda-and-mxnet/" \t "_blank), where they link to a repository of example code!

**Models for Serverless**

Though we'll be using AWS cloud tools, there are a number of other providers that you can use to build serverless applications with auto-scaling capabilities. The skills you learn in this lesson will be widely applicable to these other platforms. Some of the larger, cloud providers include:

* Amazon Web Services [ [Chalice](https://github.com/aws/chalice" \t "_blank) & [Cloud Formation](https://aws.amazon.com/cloudformation/" \t "_blank)]
* [Terraform](https://www.terraform.io/)
* [Google Cloud Platform](https://console.cloud.google.com/?pli=1)
* [Microsoft Azure](https://azure.microsoft.com/en-us/)

And you are welcome to check these out, especially if you have a platform that you already prefer.

**Why FaaS?**

* Higher developer ROI because server infrastructure management no longer required.
* More time focused on writing code, which leads to a higher developer velocity.
* Functional programming is a design well suited to distributed computing. Instead of scaling your entire application, you can scale your functions automatically and independently with usage.

### What are the Top Ten Benefits of Cloud Computing?

#### Cost

There is no up-front cost and resources can be precisely metered to meet demand.

#### Speed

The cloud offers self-service, so an expert user can leverage the resources to build solutions quickly.

#### Global scale

All major cloud providers have a global scale, which means services can be provisioned all over the world to meet demand in a geographic region.

#### Productivity

Many tasks, such as racking servers, configuring network hardware, and physically securing a data center, no longer exists. Companies can focus on building core intellectual property versus reinventing the wheel.

#### Performance

The performance of applications can leverage a continuous upgrade cycle. The network, the storage and the compute improve over time consistently. Additionally, Cloud-Native applications are able to leverage new capabilities in the cloud to further increase performance.

#### Reliability

The core architecture of the cloud offers redundancy at every step. There are multiple regions and multiple data centers in each region. Cloud-native architecture can design around these capabilities, leading to highly available architectures. Additionally, many-core cloud services are themselves highly available, like Amazon S3, which has nine nines, or 99.999999999% reliability.

#### Security

You are only as good as your weakest link with security. By consolidating to centralized security, a higher level of security occurs. Problems such as physical access to a data center or encryption at rest become industry standard on day one.

#### Relational Database

A [relational database](https://en.wikipedia.org/wiki/Relational_database" \t "_blank) is a tabular system that stores a **finite** amount of information. You can think of a relational database as a spreadsheet of values/data, where each row in the spreadsheet has a unique ID.

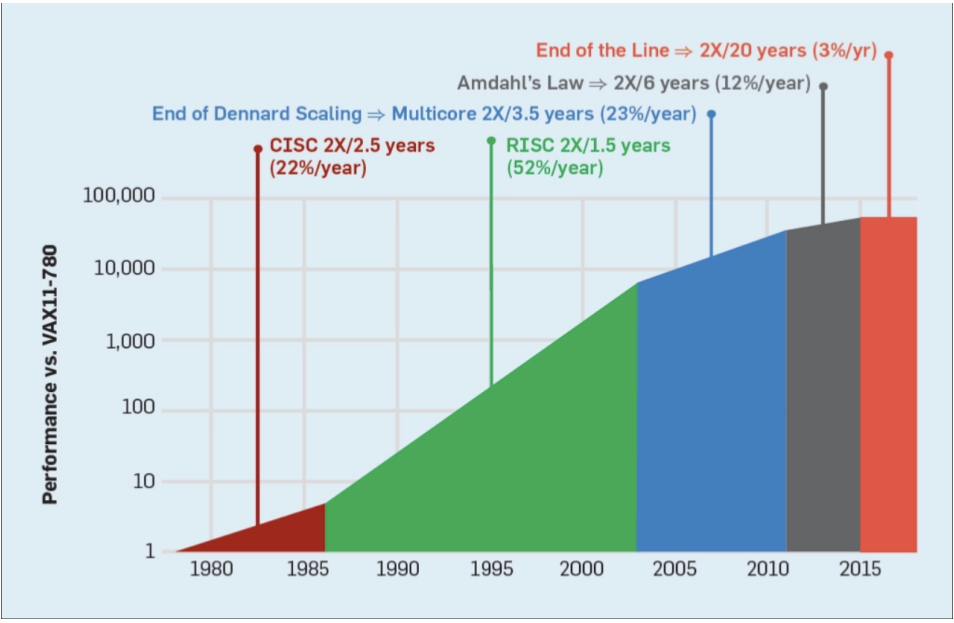
### What is Moore's Law?

The general premise of Moore’s Law is “that the speed and capability of computers can be expected to double every two years, as a result of increases in the number of transistors a microchip can contain.” - [Oxford dictionary](https://www.investopedia.com/terms/m/mooreslaw.asp" \t "_blank)

In 2015, Moore’s Law was effectively over, and gains are at 3% a year. One of the most challenging areas of computer science is distributed computing. In the modern era of cloud computing, there are a few fundamental shifts that have changed everything. One of the most significant shifts is the rise of multicore machines and the end of Moore’s Law.

Moore’s Law exposed two fundamental problems that manifest themselves in the cloud era. The first problem is that CPUs are designed to be multipurpose processors. They are not specialized to run parallel workloads. If you couple that with the ultimate physical limits of increasing CPU speed, the CPU has become less critical in the cloud era.

The second problem is that making multiple core machines to counteract the limitations of single-processor speed has led to a ripple effect on software languages. Many languages previously had substantial problems taking advantage of multiple cores because they were designed in an era before multiple processors existed, let alone the internet. Python is a good case in point here. What makes things even more challenging is that there is “no free lunch” by adding more cores to a problem that is not mostly parallel.



[**End of Moore’s Law: ACM**](https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext)

### What are Some Characteristics of Cloud-Native Systems?

#### Microservice oriented

Microservices more closely map business logic to code. These systems can be updated and developed independently. A good example of a Microservice architecture would be a Python AWS Lambda application that uses API Gateway.

#### Elastic

Elastic systems can automatically scale to meet increased load without the involvement of humans. These same systems can then scale down again when load diminishes. By eliminating human touchpoints, which are error-prone, the quality increases. Likewise, because the system can scale up and down for demand, it is more efficient to run and costs less.

#### Continuous Delivery

Cloud-native systems leverage IAC (infrastructure as code) to fully define the infrastructure. This means that deployment can target a dynamically created environment and that software can be automatically deployed to a new environment as it is created. Humans again are eliminated from the process of deployment increasing quality.

#### DevOps

Cloud-native systems utilize DevOps. DevOps is a combination of automation, processes, and tools that increase automation, collaboration, and operational efficiency. Using Python for DevOps is a common way this automation is achieved.

#### Agility

The agility of developing solutions improves. The cloud speeds up development time and increases quality through the use of IaC (Infrastructure as Code) and Continuous Delivery.

#### Composable

Cloud-Native solutions are highly composable. The service design is one of integration. At the minimum, each service has an Application Programming Interface (API) that is consistent and discoverable. Other traits include well-defined behaviors for registration, discovery, and request management.

#### Pros:

* Ability to leverage near infinite resources of the cloud: Compute, Disk I/O, Storage, and Memory.
* No up-front costs and resources can be metered to meet demand like an electric or water utility.
* Applications are able to “go global” immediately with no extra investment.
* Increased reliability is increased as many cloud services are themselves highly available. A good example is Amazon S3 which has nine nine’s availability or is 99.999999999% reliable.
* Security is improved by consolidating to a centralized security model where there is a shared security partnership with the cloud vendor. They take care of portions of security such as access to the physical data center.
* The speed applications can be developed and tested are dramatically improved. With concepts like IAC (Infrastructure as Code), complete replicas of a production environment can be provisioned, tested, and then destroyed. This leads to increased quality of software and speed in which software can be developed.

#### Cons:

* Risk of creating systems that rely on a specific cloud vendor.
* The cost involved in migrating an application to a different architecture.
* A current organization may need to hire a new workforce trained to use the cloud or retrain their workforce.

### What is Fault-Tolerance?

**Fault Tolerance** - the property that enables a system to continue operating properly in the event of the failure of one or more of its components.

* An example is a typical car, which is designed so it will continue to be drivable if one of the tires is punctured or damaged.
* In computer systems, a fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.

## What is AWS Lambda?

AWS Lambda is a new category of computing where business logic can be directly deployed to a production environment without thinking about servers. Another word for this is serverless or FaaS (Function As A Service). Some of the benefits of AWS Lambda include no servers to manage, continuous scaling, and billing at the sub-second level.

Another benefit of AWS Lambda includes being part of an ecosystem designed to exploit the capability. Here are two examples of AWS Lambda. The first example is AWS Step Functions. They build upon AWS Lambda to create more sophisticated workflows like polling for a job to finish and performing an action. The second example is AWS DeepLens (a computer vision device). It uses AWS Lambda to serve out predictions. Finally, if you look closely, almost every service on AWS can utilize AWS Lambda.

The AWS ecosystem allows Lambda functions to respond to events instead of constantly running. This is similar to a motion detector that turns on a light in the garage. The duration the light may run for could be only a few hours per year. If the light switch was manually triggered it is possible the light would be run constantly for 365 days in the year.

The AWS Lambda ecosystem allows integration with other core services in AWS through triggers. An example of a trigger would be a movie file that is uploaded to Amazon S3 storage. An AWS Lambda function could be triggered that uses an AWS API to transcode the movie file to a different code or to add captioning.

**Operationalizing AWS Lambda**

Continuing with the analogy of a motion-sensing light in the garage, imagine if there were many things that could turn on the light in the garage. What are some triggers that could be set up to turn on the light in the garage?

* Opening of the garage door
* Home burglar alarm
* A nightly timer that starts at 8 PM and ends at 8 AM

Just like the home automation example, there are many ways to trigger a Lambda function. One of the most common ways to trigger the Lambda function would be through API Gateway. A web service is created by adding API Gateway to the lambda, and then, if the URL is requested by a browser or an API call, it would, in turn, trigger the Lambda function.

Another common way to trigger the AWS Lambda function would be to attach it to Amazon S3. Instead of the Lambda function being triggered from the web service, it could be triggered every time a new file is placed in an S3 bucket. Yet another trigger could be a CloudWatch Event Timer. A timer could be set up to run every hour. At the top of each hour, the Lambda function would be called.

Let’s get into a real-world example now. A company that needs to collect a competitor’s pricing, say for a similar washing machine they are selling, could create a Lambda function that scrapes the competitor’s website. Next, a CloudWatch Event timer could run nightly to scrape the website and put the results into an S3 bucket. When the S3 bucket receives the HTML results, a second AWS Lambda function could be triggered that extracts the pricing information from the HTML file and writes it to DynamoDB, if it is lower than the current value in the database. Finally, the website itself could host a third AWS Lambda function that uses API Gateway to serve out the companies current prices, which will always be at least the same price or lower than their competitors.

Graphical user interface, application

Description automatically generated