# AWS Developer: The Big Picture

This is a survey course.

## AWS versus the Competition

The most prominent competitor is Microsoft Azure. AWS is similar to Azure with regard to cost, service offerings, and global reach. (“global reach” refers to the geographic diversity of places where a user can connect electronically to services.) The major differences between the 2 are . . .

* Azure offers the best place to run Microsoft products (he mentioned SQL Server; there may be other products, but I can’t think of any).
* AWS has a better web console. (I understand that to mean the user interface).

The instructor recommended the Pluralsight course [Understanding the Difference Between Microsoft Azure and Amazon AWS](https://app.pluralsight.com/library/courses/understanding-difference-microsoft-azure-amazon-aws/table-of-contents).

Google’s Cloud is rapidly becoming very competitive. They appear to be similar with regard to global reach.

* Google offers specializations in big data, Kubernetes, and AI.
* AWS has a much larger market share, and has a better web console.

Heroku deserves mention. Its major advantage is its simplicity to deploy and scale an application. But it lacks many of the services provided by the big players. It also has a significantly smaller global reach.

## Core Services of AWS

The services of AWS used the most are as follows.

* **Elastic Cloud Compute (EC2) –** performs like a virtual machine. Run applications, virtual desktop, 3rd party software, etc. The basic unit is an EC2 instance, a virtual server that is operating-system agnostic. Amazon displays a web console where the client can launch one (or many) EC2 instances; when doing so, the client specifies the virtual CPU, the operating system required, the amount of memory pertinent to that instance (not to be confused with S3 memory – see below), and other attributes. The cost appears to be very modest.
* **Simple Storage Service (S3) –** one can store files here. The basic unit is a bucket. I believe that some databases may be stored here. AWS assigns a URL to a bucket. The cost is reasonable.
* **Relational Database Service (RDS)** – databases – managed by Amazon – are housed here. “Managed” means all backups, software updates, and infrastructure. (However, I don’t know what “infrastructure” means.) It is recommended that one use RDS, instead of installing one’s own database via EC2 (see above). AWS offers a variety of database types – including SQL Server. The cost seems to be pretty steep (more than $.01/hour, but given the small size of database that Current Pixel requires, it might turn out to be cheaper. Cost might also be mitigated by using other types of database (maybe Amazon Aurora).
* **Route53** – DNS (Domain Name System) service that powers EC2 instances and S3 buckets to be accessible via URL. Cost is $.50/month per hosted zone, $.40/(million queries), and nothing for added DNS entries.

## Other AWS Services

1. **Elastic Beanstalk** – used to deploy a client’s application (desktop or web) to EC2. It relieves the client from
   * manual configuration
   * manual code deployment
   * restricted command-line interface
   * solve scaling problems via AMIs (Amazon Machine Image)
   * manual monitoring

This seems to be a convenient alternative to using EC2 and S3 manually. Elastic Beanstalk is launched from the AWS web console, the AWS command-line tool, or through the SDK. Elastic Beanstalk monitoring provides logs for the application. Elastic Beanstalk, itself, is free. The only costs that client incurs are the EC2 instances, the S3 buckets, and the load balancing.

1. **Lambda** – provides code execution which is charged to the client only when the code is running. There is no cost to the client for code that sits idle. This can save the client substantially if the function is executed infrequently. The basic unit is the **Function**. The client provides the code to a Lambda Function with …
   * a specified execution entry point
   * a specified platform (Node, Python, Go, or other languages)
   * triggers (invocation targets)
   * optional configuration (execution time-outs, memory requirements, IAM roles)

For a specific geographic area the costs are as follows. (The cost may be different in other areas.)

* + 1st 100 million requests per month are free
  + 1st 400,000 Gbyte-seconds per month are free

(A Gbyte-second is a 1 Gbyte function executing for 1 second.)

Subsequently

* $.20 / (million requests)
* approximately $.18 / ( 128Mbyte executing for 24 hours)

1. **Dynamo DB** – A database that is significantly scaled back. It is simply a collection of tables, each of which has one or more indexes. (The instructor called this a NoSQL type of database. In my opinion I doubt that this means that it is not relational. There is nothing that prevents a column in one table from serving as a foreign key for a related table.)

The client configures each table in terms of a specified number of **write-units** and a specified number of **read-units**. A **write-unit** contains 1 Kbyte, and a **read-unit** contains 4 Kbytes. (It is not clear to me what happens if the amount of data read or written exceeds the configured values.)

For a specific geographic area the costs are as follows. (The cost may be different in other areas.)

* The 1st 25 GBytes are free.
* Subsequent storage costs $0.25 / GByte each month.
* The cost for writing is $0.0065 for every 10 **write-units** every hour.
* The cost for reading is $0.0065 for every 50 **write-units** every hour.

1. **Virtual Private Cloud** provides additional security. The client sets up private fire walls and supplies rules that govern communications among his AWS entities. There is no cost for this service.
2. **Cloud Watch** monitors resources, and initiates an alarm when a specified condition arises. For example the client may want to set up alarms that are triggered when thresholds are crossed. (What comes to my mind are (1) too many clients connecting to an AWS service, (2) too many reads from a particular database, (3) too many exceptions recorded in a log, etc.) The alarm action might take the form of communication via e-mail, SMS notification (mobile-phone texts), or automatic scaling of EC2 instances.

For a specific geographic area the costs are as follows. (The cost may be different in other areas.)

* $0.10 for each configured alarm per month.
* $0.50 per GByte for ingesting logs.
* $0.03 per GByte for archiving logs.

**Cloud Watch** also allows the client to create custom dashboards.

* $3.00 per dashboard each month.

1. **Cloud Front** is a Content Delivery Network (CDN) provided by AWS to combat latency. From this course’s description I infer that it would be used by businesses whose customers are geographically wide spread, and where there might be a concern about internet response time.

# Interacting with AWS

The AWS **Web Console** is the tool that will very likely be used most of the time as an AWS client. The instructor provided a brief tour, and at first glance it is overwhelming because of the wide variety of services that AWS provides (far more than the 10 listed above in this document). But I believe the Pluralsight instructor’s assertion - that the **Web Console** is well organized, and that with a small amount of training the client (I and some Current Pixel employees) will find it easy to use a subset of its features.

AWS offers several **SDKs (Software Development Kits)** each of which allows the programmer to interact with AWS via software. The instructor listed 12 SDK flavors, each of which serves a different programming language. (He said that there may be others, but these were the ones that he found.)

1. RAILS
2. nodeJs
3. GO
4. .NET
5. Android
6. unity
7. iOS
8. JavaScript
9. Ruby
10. Php
11. C++
12. Java

Notably missing are C# and Visual Basic, but these are probably part of .NET. At this point I am confused. Unless Current Pixel uses a new data-storage strategy (e.g. Dynamo DB), I don’t see any need for a program connection to an AWS service. If there is a program connection, it would probably be .NET or JavaScript.

OK, now that I think about it there do occur unusual circumstances when a program is written (1) to create a temporary database, or (2) to modify the production database (e.g. to recover from a bug that has polluted the database). Even if the database is SQL Server, and even if we don’t need an SDK to engage that database in AWS’s RDS, it might be significantly easier to do this using the .NET SDK. We’ll see.

The **AWS Command Line Interpreter** will automate the interactions that one would perform via the Web Console. It’s not too likely that a client would type commands individually at a computer console. Instead, the CLI’s primary value is the ability to create shell (batch) scripts of CLI commands (e.g. to automate Builds). The **AWS Command Line Interpreter** does require some installation steps, but these are straightforward.