AWS FOR THE REST OF US

September 23rd, 2020

AWS CLOUD IAAS PAAS

AWS is the premier cloud provider - they sell the infrastructure building blocks to build modern apps.

- Today, most applications run in the cloud - e.g. on rented servers - and AWS made **$35B last year** selling them
- AWS products fit into **3 buckets**: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)
- AWS provides a **web UI** to provision and configure servers, but bigger teams will use their **API** to do this through code instead
- Some of AWS's most popular products are **EC2** (simple compute), **S3** (simple storage), and **RDS** (a managed database)

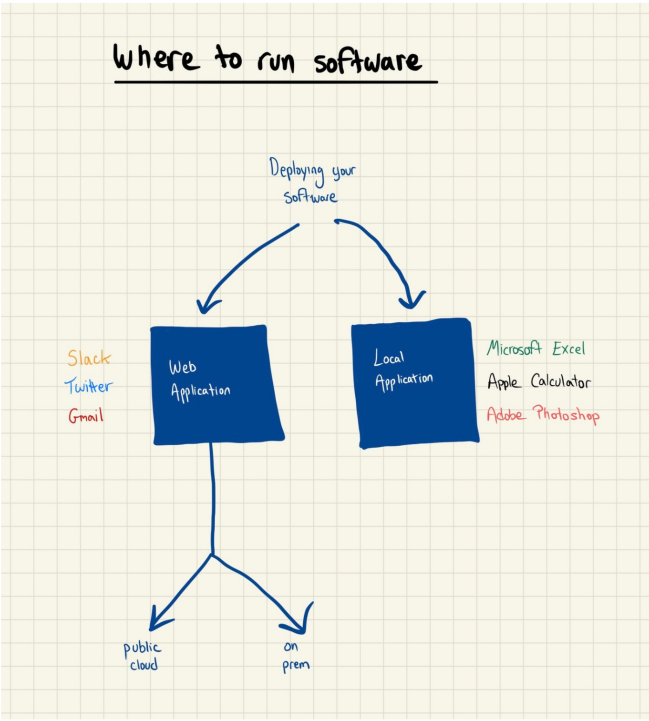
There are other big cloud providers – namely Google (GCP) and Microsoft (Azure), but AWS holds something close to a 50% market share. If you know or work with a developer, chances are they've used AWS.

CLOUD SERVICES: A BASIC PRIMER

To understand what AWS actually does, you need to understand the cloud in general. For a more in depth explanation, check out the Technically post about cloud here.

There are basically two ways to run an app - **locally or over the internet**. In the "old" days (i.e. high school), most apps ran locally - you'd get a copy of Excel via a CD, or download it from the web. All of the computing that Excel did - both the "graphical" frontend you interacted with, and all of the math that happened behind the scenes - took place on your laptop. Even if Excel did *sometimes* communicate with the web, it was only to pull in a data source and get updated. You usually paid a one time fee to buy the software, or licensed it yearly.

Things have changed a lot since then. Now, most software runs over the internet - you access it via your browser. And even if there's a desktop app, a lot of the hard work is getting done via the internet. So what does it *mean* exactly for an app to run on the web?



Cloud-based apps have most of their code deployed on a big, powerful server in

someone's data center - *not* on your computer. When you load up Twitter, your browser is sending a request to a web server - that server runs a bunch of code, generates your feed, and then sends back a bunch of HTML that makes up what you see. The same thing happens when you use Office365 over the web, or Gmail, or any other cloud-based service.

Web servers that handle and route requests from a browser are just one very small piece of a much larger puzzle. Most apps need a database; a lot of apps need multiple types of databases; and as apps get larger and more complicated, they start to require increasingly unique and specialized services. A few examples:

- A data warehouse for analytics
- Messaging services for streaming data (Technically post coming soon)
- Video encoding and decoding (Technically post not coming soon)
- ETL tools for moving data around

AWS has solutions for *all* of these, and also like 400 more things. The hard part is organizing them. And also naming them. AWS does an inexplicably horrible job of naming their products (it's a meme at this point).

IAAS, PAAS, AND SAAS

AWS literally has hundreds of different products, and *someone* is using all of them. Even the AWS team struggles to organize and market them effectively. But the easiest way to understand the breakdown is by splitting things into 3 categories, ordered from basic to more "white glove" or "managed:"

1. **Infrastructure as a Service (IaaS)**

IaaS is the basic, lower level infrastructure that you need to build an app. Generally, this translates into:

- Basic **compute** on a virtual machine (AWS product: EC2)
- Basic **storage** (AWS products: S3, EBS)
- Basic **networking** (AWS products: Route53, VPC)

These products don't do much beyond the basics. With EC2 (which stands for Elastic Compute Cloud, of course), you're just getting a virtual machine - you need to deal with upgrades, sizing correctly, and backing your data up. This is still *fantastic* compared to building your own goddamn data center, but it's barebones. If you're wondering: "is it just me, or are these names horrible?" - no, it's not just you. Yes, they are horrible.

2. **Platform as a Service (PaaS)**

PaaS (pronounced "pass" but the "a" is more like "acquiesce") is the next step up - these services are more expensive, but make life easier by taking care of some of the annoying operational tasks that you're still on the hook for with IaaS. A few examples:

- Managed **databases** (AWS products: RDS, Dynamo)
- Managed **data warehouses** (AWS products: Redshift)
- Managed **compute** (AWS products: Lambda, Elastic Beanstalk)
- Managed **machine learning** (AWS products: Sagemaker, Rekognition)

- Managed **machine learning** (AWS products: SageMaker, Rekognition)
- Other **misc.** managed stuff (AWS products: Elasticsearch, SQS)

These services are called "managed" because AWS takes care of upgrades, backups, handling downtime, and scaling up and down (among other things). Developers pay a premium so they don't need to deal with any of this.

3. Software as a Service (SaaS)

Even though PaaS is *more* managed, you still need to manage infrastructure, log into servers, and think about scaling / pricing. When you hear SaaS, you probably think of something like Salesforce or Intercom, and you're not wrong - AWS also provides services like these that are *fully managed* - i.e. you don't need to touch any infrastructure at all. A few examples:

- AWS Quicksight - a **data visualization** tool
- AWS Forecasting - **time series forecasting** as a service
- AWS Cognito - **identity management** as a service

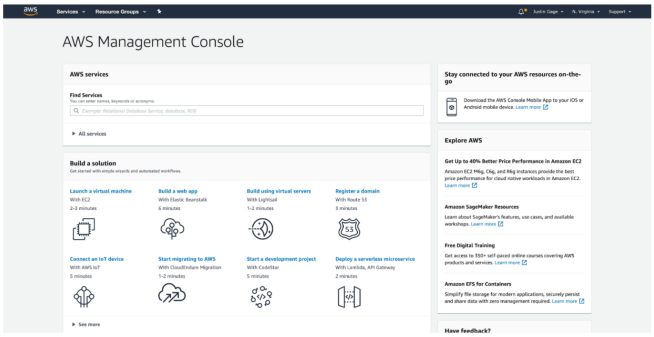
The lines between these three categories blur a lot, and it's not a perfect taxonomy, but it's helpful to put things in context, especially AWS's more niche stuff.

One thing to note: the *oomph* of AWS isn't just the breadth of what they offer (which I hope is clear by now), but also how *good* these products actually are. Products like RDS and Redshift are generally recognized as incredibly useful, reliable, and just generally get the job done as well as anything else on the market, if not better.

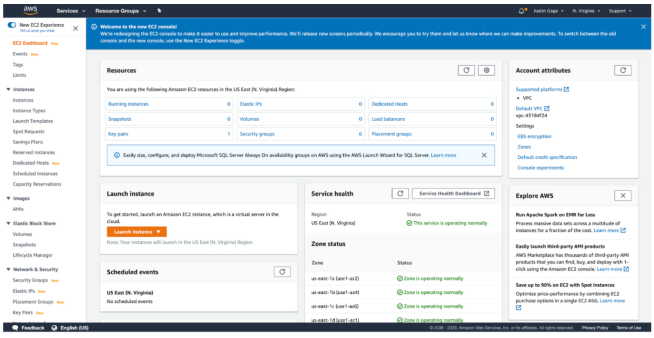
AWS makes a lot of bad products too, mind you. AWS's Elasticsearch service is notoriously horrible, and a lot of their products just leave engineers and analysts scratching their heads as to what the intent was. Billing and pricing - which we'll see is a core part of any IaaS offering - is potentially the worst in the industry. So ultimately, as with many stories of this nature, AWS isn't as simple as good or bad.

EXAMPLE: A TYPICAL AWS SETUP

The easiest way to get more comfortable with the AWS suite is to see it in action. Let's start with AWS's web app: it lets you provision new resources (generic word for products) and manage your existing setups. Here's the homepage:



Each AWS product has its own management console page. Here's the one for EC2:



This UI changes a lot, as AWS is still figuring out what functionality to include here vs. their API. From the EC2 console, you can launch a new instance, see service health, add elastic IPs, and do a lot of other fun stuff that developers need to do to keep their jobs.

Even medium sized startups will often be using 10+ AWS services from the get go, and more established businesses can easily go past 100. Let's imagine we're a startup that sells technical literacy and education software to tech businesses (let's imagine). We've got a basic web application, and a little data warehouse for our Growth Lead to report basic company metrics. We might be using:

1. **EC2** to deploy our web app in a few Docker containers
2. **Lambda** to process form submissions on the marketing site
3. **EBS** for block storage connected to our EC2 instance(s)
4. **S3** to store backups and files for the app and marketing site
5. **Route53** to connect our domain name to our AWS servers
6. **RDS** (Postgres) as our managed database for our web app
7. **Cloudfront** as our CDN for serving assets quickly
8. **VPC** to isolate our resources into a private, secure network
9. **Backup** to back up our data across services
10. **Redshift** to store analytics data as our data warehouse

> 😇 **DON'T SWEAT THE DETAILS** 😇
>
> If any of these individual use cases don't make sense, don't sweat it - just keep in mind that running a modern application is, for better or worse, kind of hard, and you need a lot of specialized tools to do it.

These are just the AWS *products* that you'll be using - but there are other parts of the ecosystem that help *support* this product usage; sort of like the glue that keeps things together. A good example is IAM, or Identity and Access Management - it's an AWS utility that allows you to allocate different permissions across your organization. You'd use this to make sure that the right teams have access to the right resources, and restrict mission critical resources to developers who know what they're doing.

If all of this seems like a lot, it is. And it costs a lot too - probably $10K+ per month. But this isn't actually that much in the scheme of things; AWS makes most of its money through very very large enterprise clients. Lyft spends $80M per year on AWS, and Netflix probably spends more than $100M. Next time you're wondering why you can't get any support agents on the phone to talk about your $10 invoice - this is why. It's because you're irrelevant.

PRICING MODELS AND CONFUSION

Infrastructure as a Service is a commodity business, or at least something very close to it, and that makes pricing an actual driving force of the business, not just an afterthought. In fact, you can argue that pricing is actually *the entire product* for AWS's IaaS offerings – the ability to pay **per use** as opposed to putting up a ton of **capex** in advance is what makes public cloud exciting in the first place. People refer to this change in business models as **shifting your capex** (capital expenditures) **to opex** (operating expenditures).

With that background in mind, it's pretty easy to understand why AWS prices things the way they do. There are basically two pricing schemes:

1. **Pay per time used**

Some AWS services get charged per hour, with different prices depending on how "big" the resources you're using are. The best example of this is EC2 (pricing link here), AWS's basic IaaS compute product. The t3.micro instance with 1GB of RAM costs $0.0104 per hour, the t3.small instance with 2GB of RAM costs $0.0208 per hour, and so on and so forth. In general, the more powerful the machine, the more you pay. If you shut down your machine, you cease to pay for it.

2. **Pay per resources used**

Some AWS services give you the option to pay per compute or storage usage as opposed to time. DynamoDB, AWS's managed NoSQL database (pricing link here), costs $1.25 per million write requests, and $0.25 per million read requests. A bunch of AWS products give you options for both of these pricing models, and some have even wackier ones.

But things don't stop there. If you're planning on using AWS for quite a while (1+ years), you can get steep volume discounts by committing to using resources for a longer period of time – called Reserved Instances. These kinds of commitments are very valuable to AWS, because it makes capacity planning a hell of a lot easier - so these discounts can often climb up past 50%. AWS recently unveiled a new program called "Savings Plans" (finally, something aptly named) that slightly adjusts this model.

If you've heard of AWS pricing before, it's probably because it has one of the worst reputations in the entire tech universe - it's opaque, unbelievably complicated, and generally not beginner friendly at all. If you're using 15 different AWS services (and as we saw in the previous section, this is not at all uncommon), you're dealing with 15 different pricing models. There are entire companies that exist just to help you understand and lower your AWS bill.

> 🔍 **DEEPER LOOK** 🔍
>
> DigitalOcean is a competing cloud provider with AWS, and a big part of their value

proposition is simpler, more predictable pricing. If that sounds like a weird thing to differentiate on, it's because you haven't worked in commodity businesses - in IaaS, pricing transparency is a huge selling point for smaller businesses.

MORE CLOUDS AND MORE MONEY

As you've probably heard, AWS isn't the only public cloud - and because of the nature of this business, it's not the only good one either. The infrastructure as a service landscape in general kind of splits into two distinct segments:

1. **High end built for scale**

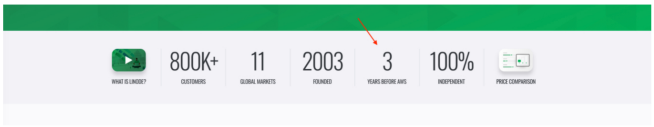
This is AWS, Google (GCP), and Microsoft (Azure). All three of these companies had extensive experience building data centers for themselves, and then pivoted that towards a public facing product. And all three clouds have the benefit of getting bankrolled by billions of dollars of cash from their parent companies. IBM, Oracle, and Alibaba fit in here too.

AWS completely dominated the market from the start, but today things are a bit more evened out. Plenty of startups and companies happily run their infrastructure on GCP or Azure (like yours truly). And GCP and Azure have some individual products that are best in-market - GCP's BigQuery is generally considered the best serverless data warehouse on the street.

2. **Low end built for small projects**

This is Hetzner, Scaleway, OVH, Linode, and Vultr - if you've never heard of them, you're not alone - most developers in the U.S. probably haven't. They sell very cheap, very barebones stuff mostly oriented towards small, personal projects or VPNs. Because they don't have to deal with hyper scale, managing capex is slightly more do-able, and these can be nice medium margin businesses.

There's a lot of yikes here though, like when Linode's marketing team thought that being earlier to the cloud game than AWS somehow makes up for the fact that AWS ate their f*cking lunch:



Sorry folks.

The one cloud (well, IaaS cloud) missing from this list is DigitalOcean - they sort of play in the middle of these two segments. All things considered, a DigitalOcean server is generally slightly cheaper than the equivalent on one of the big cloud providers - but the company also provides more advanced products like Load Balancers, Managed Kubernetes, and Managed Databases.

If you liked or hated this, share it on Twitter, Reddit, or HackerNews.
