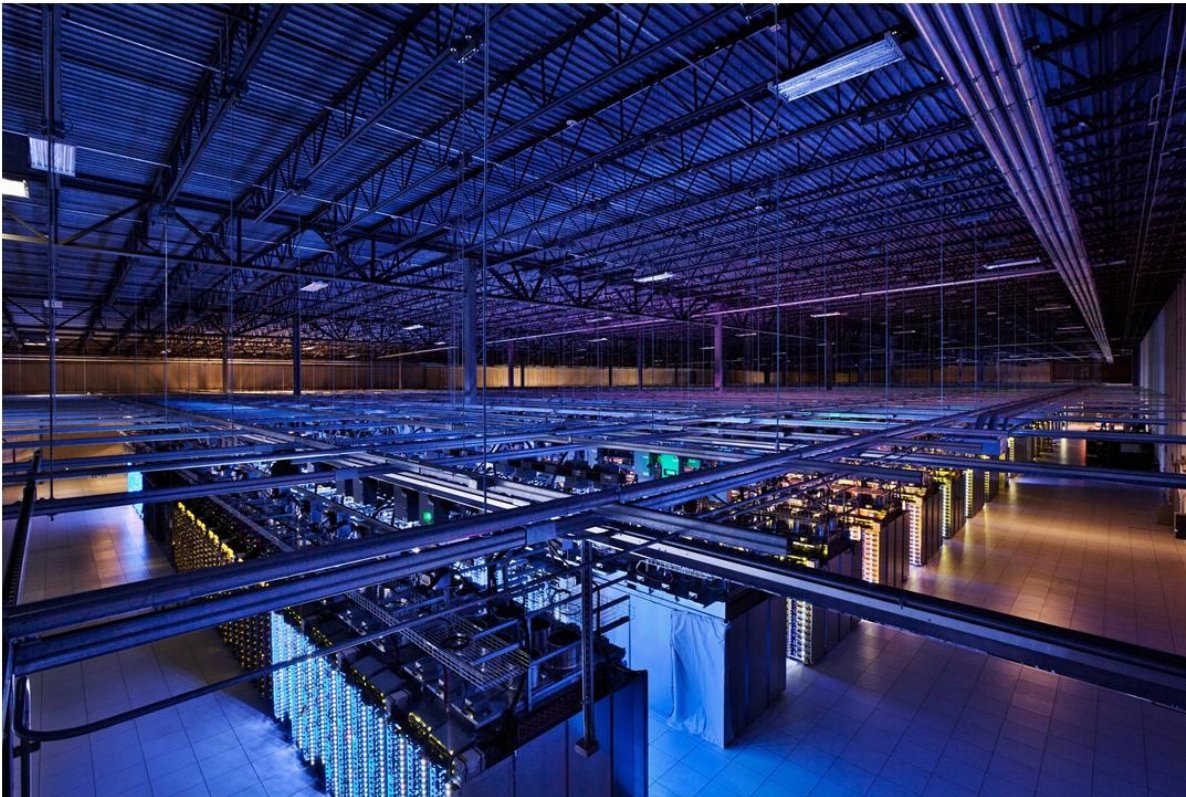


Large Scale Data Engineering

Cloud Computing



Cloud computing

- **What?**
 - Computing resources as a metered service (“pay as you go”)
 - Ability to dynamically provision virtual machines
- **Why?**
 - Cost: capital vs. operating expenses
 - Scalability: “infinite” capacity
 - Elasticity: scale up or down on demand
- **Does it make sense?**
 - Benefits to cloud users
 - Business case for cloud providers

Everything as a service

- Utility computing = **Infrastructure as a Service** (IaaS)
 - Why buy machines when you can rent cycles?
 - Examples: Amazon's EC2, Rackspace
- **Platform as a Service** (PaaS)
 - Give me nice API and take care of the maintenance, upgrades
 - Example: Google App Engine
- **Software as a Service** (SaaS)
 - Just run it for me!
 - Example: Gmail, Salesforce

Several Historical Trends (1/3)

- Shared Utility Computing
 - 1960s – MULTICS – Concept of a Shared Computing Utility
 - 1970s – IBM Mainframes – rent by the CPU-hour. (Fast/slow switch.)
- Data Center Co-location
 - 1990s-2000s – Rent machines for months/years, keep them close to the network access point and pay a flat rate. Avoid running your own building with utilities!
- Pay as You Go
 - Early 2000s - Submit jobs to a remote service provider where they run on the raw hardware. Sun Cloud (\$1/CPU-hour, Solaris +SGE) IBM Deep Capacity Computing on Demand (50 cents/hour)

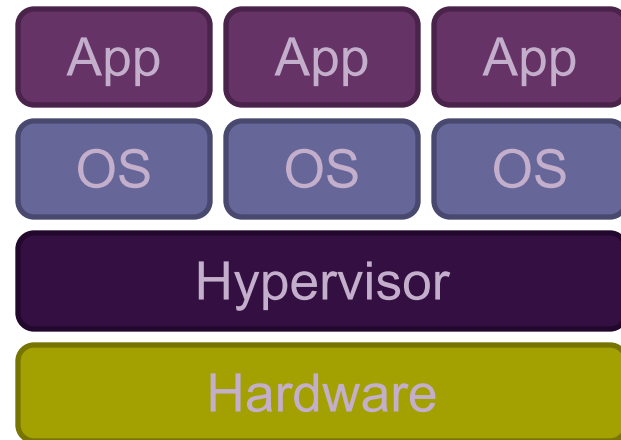
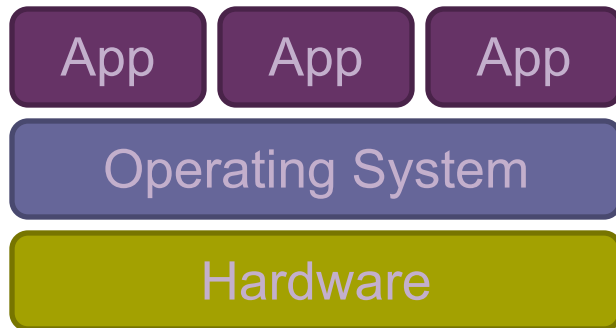
Several Historical Trends (2/3)

- Virtualization
 - 1960s – OS-VM, VM-360 – Used to split mainframes into logical partitions.
 - 1998 – VMWare – First practical implementation on X86, but at significant performance hit.
 - 2003 – Xen paravirtualization provides much perf, but kernel must assist.
 - Late 2000s – Intel and AMD add hardware support for virtualization.

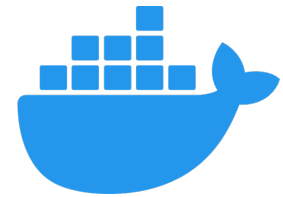
Several Historical Trends (3/3)

- Minicomputers (1960-1990)
 - IBM AS/400, DEC VAX
- The age of the x86 PC (1990-2010)
 - IBM PC, Windows (1-7)
 - Linux takes the server market (2000-)
 - Hardware innovation focused on Gaming/Video (GPU), Laptop
- Mobile and Server separate (2010-)
 - Ultramobile (tablet,phone) .→ ARM
 - Server → x86 still but also own ARM and RISC-V
 - **Large cloud providers build their own hardware**
 - Amazon: ARM CPUs (**Graviton**), **Nitro** (security and resource limiting) in own **motherboards, network cards, SSDs**
 - Google: **Tensor Processing Units (TPU)** hardware for ML

Enabling technology: virtualisation



Enabling technology: containers



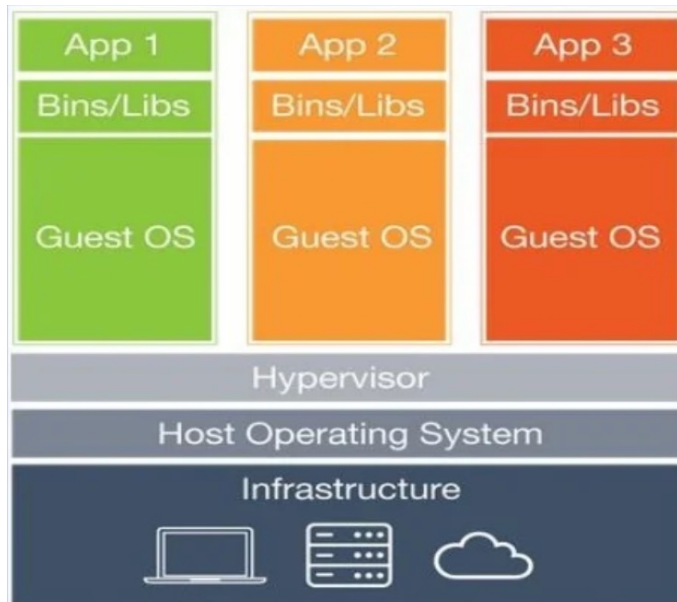
docker

- Most popular: **Docker**
 - Operating systems started to support **resource isolation**
 - Groups of processes that do not see each other
 - Allows different versions of software libraries to co-exist
 - software distributed with all dependency libraries in one container
 - No need to install dependent libraries. No problems with version clashes

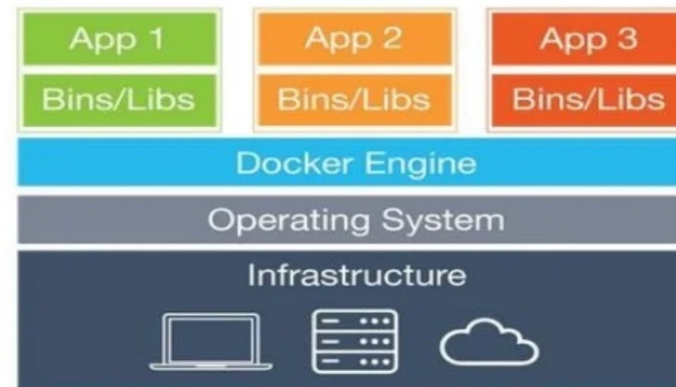
developments:

#1: containers replacing VMs (more fine-grained)

#2: containers running in VMs (e.g. in the cloud)



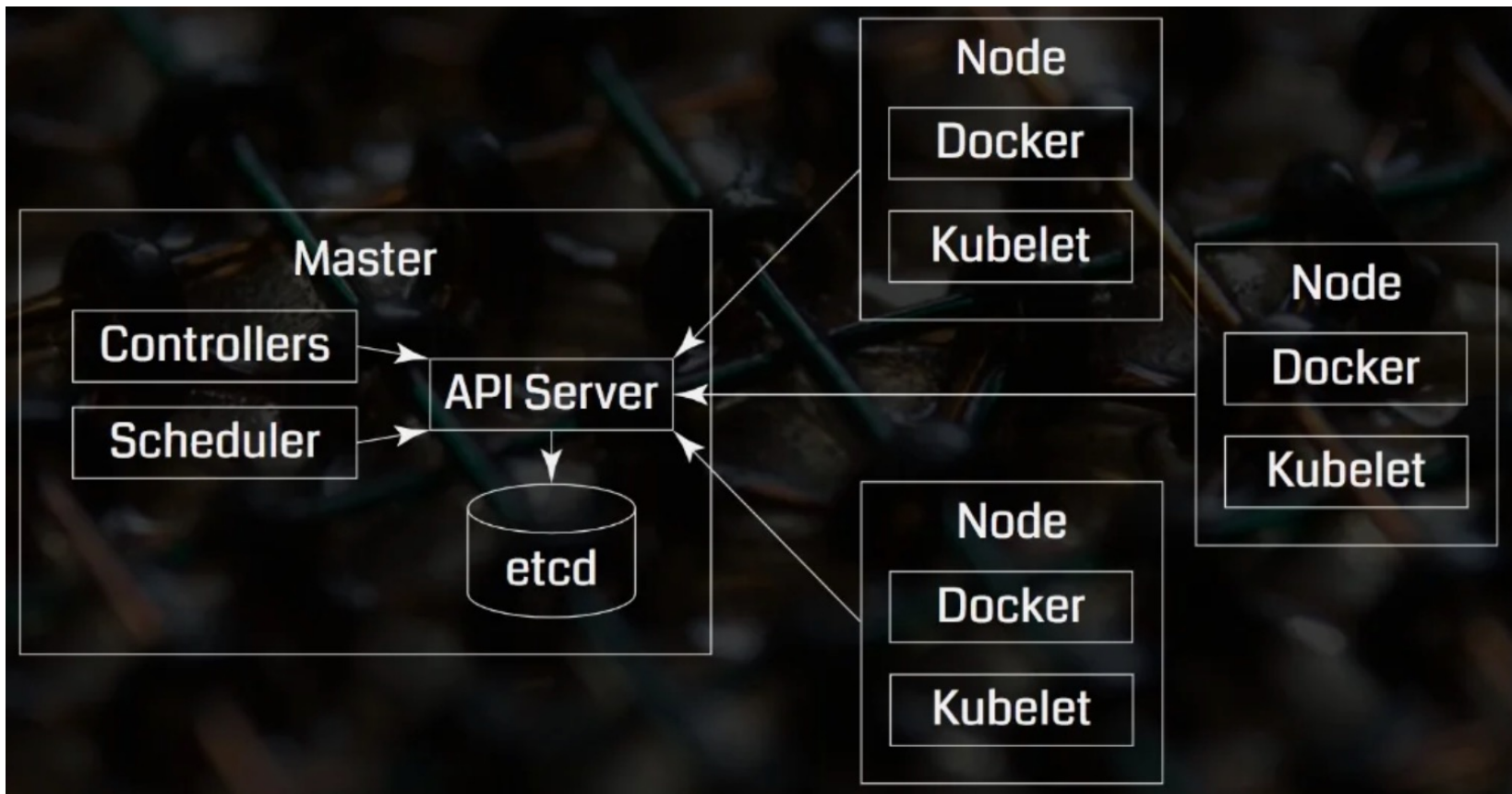
Virtual Machines



Containers

Enabling technology: cluster orchestration

- Most popular: **Kubernetes** (originally from Google)
 - Cluster orchestration software to schedule VMs and containers

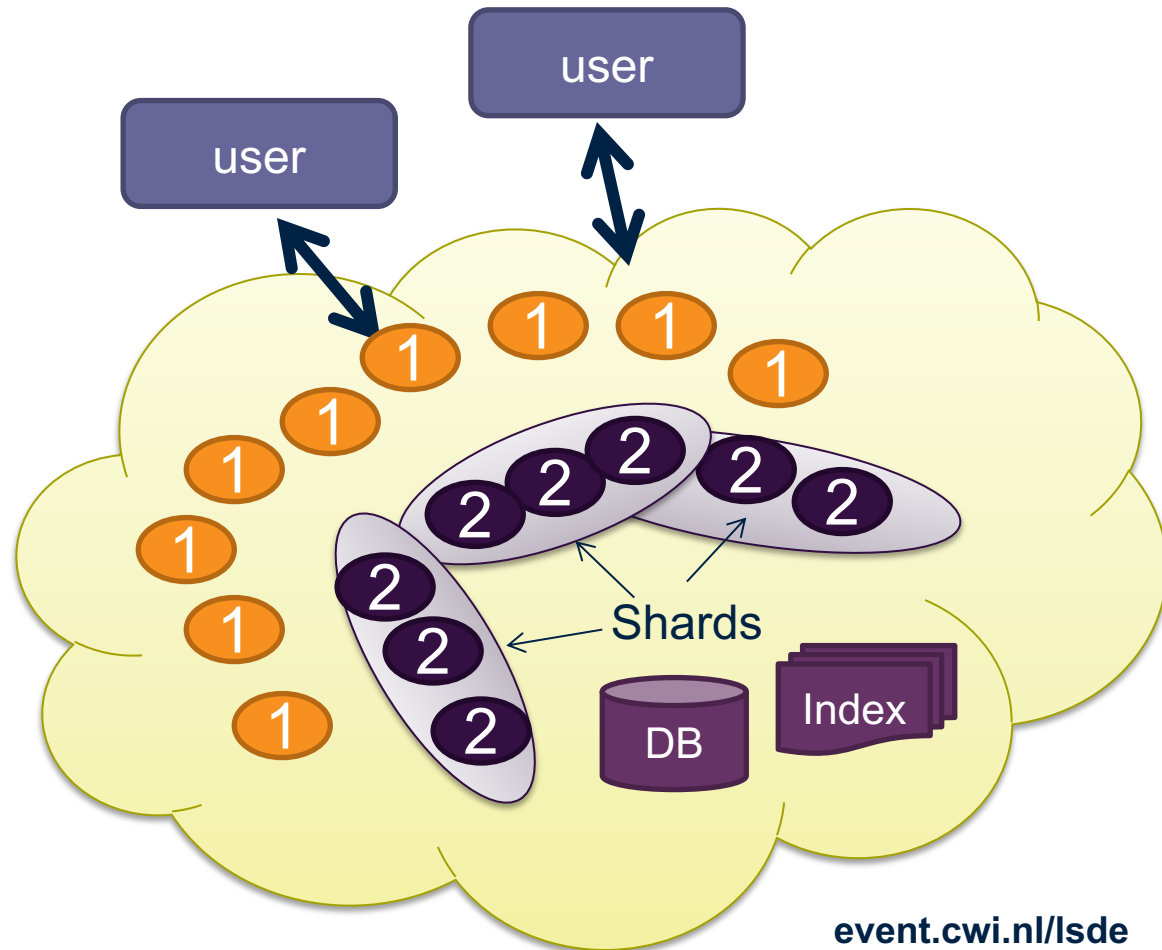


Seeks vs. scans

- Consider a 1TB database with 100 byte records
 - We want to update 1 percent of the records
- Scenario 1: random access
 - Each update takes ~30 ms (seek, read, write)
 - 10^8 updates = ~35 days
- Scenario 2: rewrite all records
 - Assume 100MB/s throughput
 - Time = 5.6 hours(!)
- Lesson: avoid random seeks!

Big picture overview

- Client requests are handled in the first tier by
 - PHP or ASP pages
 - Associated logic
- These lightweight services are fast and very nimble
- Much use of caching: the second tier



Many styles of system

- Near the edge of the cloud focus is on vast numbers of clients and rapid response
 - Web servers, Content Delivery Networks (CDNs)
- Inside we find high volume services that operate in a pipelined manner, asynchronously
 - like Kafka (streaming data), Cassandra (key-value store)
- Deep inside the cloud we see a world of virtual computer clusters that are
 - Scheduled to share resources
 - Run frameworks like Hadoop and Spark (data analysis) or Presto (distributed databases)
 - Perform the heavy lifting

In the outer tiers replication is key

- We need to replicate
 - Processing
 - Each client has what seems to be a private, dedicated server (for a little while)
 - Data
 - As much as possible!
 - Server has copies of the data it needs to respond to client requests without any delay at all
 - Control information
 - The entire system is managed in an agreed-upon way by a decentralised cloud management infrastructure

What about the shards?

- The caching components running in tier two are central to the responsiveness of tier-one services
- Basic idea is to always use cached data if at all possible
 - So the inner services (here, a database and a search index stored in a set of files) are shielded from the online load
 - We need to replicate data within our cache to spread loads and provide fault-tolerance
 - But not everything needs to be fully replicated
 - Hence we often use shards with just a few replicas

Read vs. write

- Parallelisation works fine, so long as we are reading
- If we break a large read request into multiple read requests for sub-components to be run in parallel, how long do we need to wait?
 - Answer: as long as the slowest read
- How about breaking a large write request?
 - Duh... we still wait till the slowest write finishes
- But what if these are not sub-components, but alternative copies of the same resource?
 - Also known as replicas
 - We wait the same time, but when do we make the individual writes visible?

Replication solves one problem but introduces another

More on updating replicas in parallel

- Several issues now arise
 - Are all the replicas applying updates in the same order?
 - Might not matter unless the same data item is being changed
 - But then clearly we do need some agreement on order
 - What if the leader replies to the end user but then crashes and it turns out that the updates were lost in the network?
 - Data center networks are surprisingly lossy at times
 - Also, bursts of updates can queue up
- Such issues result in inconsistency

Eric Brewer's CAP theorem

- In a famous 2000 keynote talk at ACM PODC, Eric Brewer proposed that
 - *“You can have just two from Consistency, Availability and Partition Tolerance”*
- He argues that data centres need very fast response, hence availability is paramount
- And they should be responsive even if a transient fault makes it hard to reach some service
- So they should use cached data to respond faster even if the cached entry cannot be validated and might be stale!
- Conclusion: weaken consistency for faster response
- We will revisit this as we go along

Is inconsistency a bad thing?

- How much consistency is really needed in the first tier of the cloud?
 - Think about YouTube videos. Would consistency be an issue here?
 - What about the Amazon “number of units available” counters. Will people notice if those are a bit off?
 - Probably not unless you are buying the last unit
 - End even then, you might be inclined to say “oh, bad luck”

CASE STUDY: AMAZON WEB SERVICES

Amazon AWS

Grew out of Amazon's need to rapidly provision and configure machines of standard configurations for its own business.

- **Early 2000s** – Both private and shared data centers began using virtualization to perform “server consolidation”
- **2003** – Internal memo by Chris Pinkham describing an “infrastructure service for the world.”
- **2006** – **S3 first deployed in the spring, EC2 in the fall.** *Infrastructure*
- **2008** – **EBS (Elastic Block Store)**
- **2009** – **Amazon RDS (hosted relational databases)**
- **2012** – **DynamoDB (key-value store)** *Platform*
- **2013** – **Redshift (analytical database)**
- **2013** – **Kinesis (data streaming service)** *Software*
- **2013** – **Lambda (compute service)** *(serverless)*
- **2014** – **Aurora (transactional databases)**
- **2016** – **Athena (analytics db)**
- **2017** – **SageMaker (machine learning)**

Terminology

- Instance = One running virtual machine.
- Instance Type = hardware configuration: cores, memory, disk.
- Instance Store Volume = Temporary disk associated with instance.
- Image (AMI) = Stored bits which can be turned into instances.
- Key Pair = Credentials used to access VM from command line.
- Region = Geographic location, price, laws, network locality.
- Availability Zone = Subdivision of region the is fault-independent.

AWS Core Infrastructure and Services

Traditional Infrastructure



Firewalls



ACLs



Administrators

Security

Amazon Web Services



Security Groups



Network ACLs



AWS IAM



Router



Network Pipeline



Switch

Networking



ELB



VPC



On-Premises Servers

Servers



AMI



Amazon EC2 Instances



DAS



SAN



NAS

RDBMS

Storage and Database



Amazon EBS



Amazon EFS



Amazon S3



Amazon RDS

AWS Foundation Services

Compute

-  Amazon EC2
-  Amazon EC2 Container Registry
-  Amazon EC2 Container Service
-  Amazon Lightsail
-  Amazon VPC
-  AWS Batch
-  AWS Elastic Beanstalk
-  AWS Lambda
-  Elastic Load Balancing

Network

-  Amazon CloudFront
-  Amazon Route 53
-  Amazon VPC
-  AWS Direct Connect
-  Elastic Load Balancing

Storage

-  Amazon EFS
-  Amazon Glacier
-  Amazon S3
-  AWS Snowball
-  AWS Storage Gateway






































Security & Identity

-  Amazon Inspector
-  AWS Artifact
-  AWS Certificate Manager
-  AWS CloudHSM
-  AWS Directory Service
-  IAM
-  AWS KMS
-  AWS Organizations
-  AWS Shield
-  AWS WAF

Applications

-  Amazon WorkDocs
-  Amazon WorkMail
-  Amazon AppStream
-  Amazon WorkSpaces

AWS Platform Services

| Databases | Analytics | Application Services | Management Tools | Developer Tools | Mobile Services | Internet of Things |
|---|--|---|---|--|---|--|
|  Amazon DynamoDB |  Amazon Athena |  Amazon API Gateway |  Amazon CloudWatch |  AWS CodeBuild |  Amazon API Gateway |  AWS IoT |
|  Amazon ElastiCache |  Amazon CloudSearch |  Amazon AppStream 2.0 |  AWS CloudFormation |  AWS CodeCommit |  Amazon Cognito |  AWS Greengrass |
|  Amazon RDS |  Amazon EMR |  Amazon Elastic Transcoder |  AWS CloudTrail |  AWS CodeDeploy |  Amazon Mobile Analytics | |
|  Amazon Redshift |  Amazon ES |  Amazon SWF |  AWS Config |  AWS CodePipeline |  Amazon Pinpoint | |
| |  Amazon Kinesis |  AWS Step Functions |  AWS Managed Services |  AWS X-Ray |  AWS Device Farm | |
| |  Amazon QuickSight | |  AWS OpsWorks | |  AWS Mobile Hub | |
| |  Amazon Redshift | |  AWS Service Catalog | | | |
| | | |  AWS Trusted Advisor | | | |

AWS Global Infrastructure



AWS Global Infrastructure

At least 2 Availability Zones
per region.

Examples:

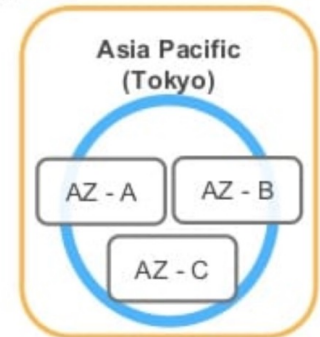
- US East (N. Virginia)

- us-east-1a
- us-east-1b
- us-east-1c
- us-east-1d
- us-east-1e

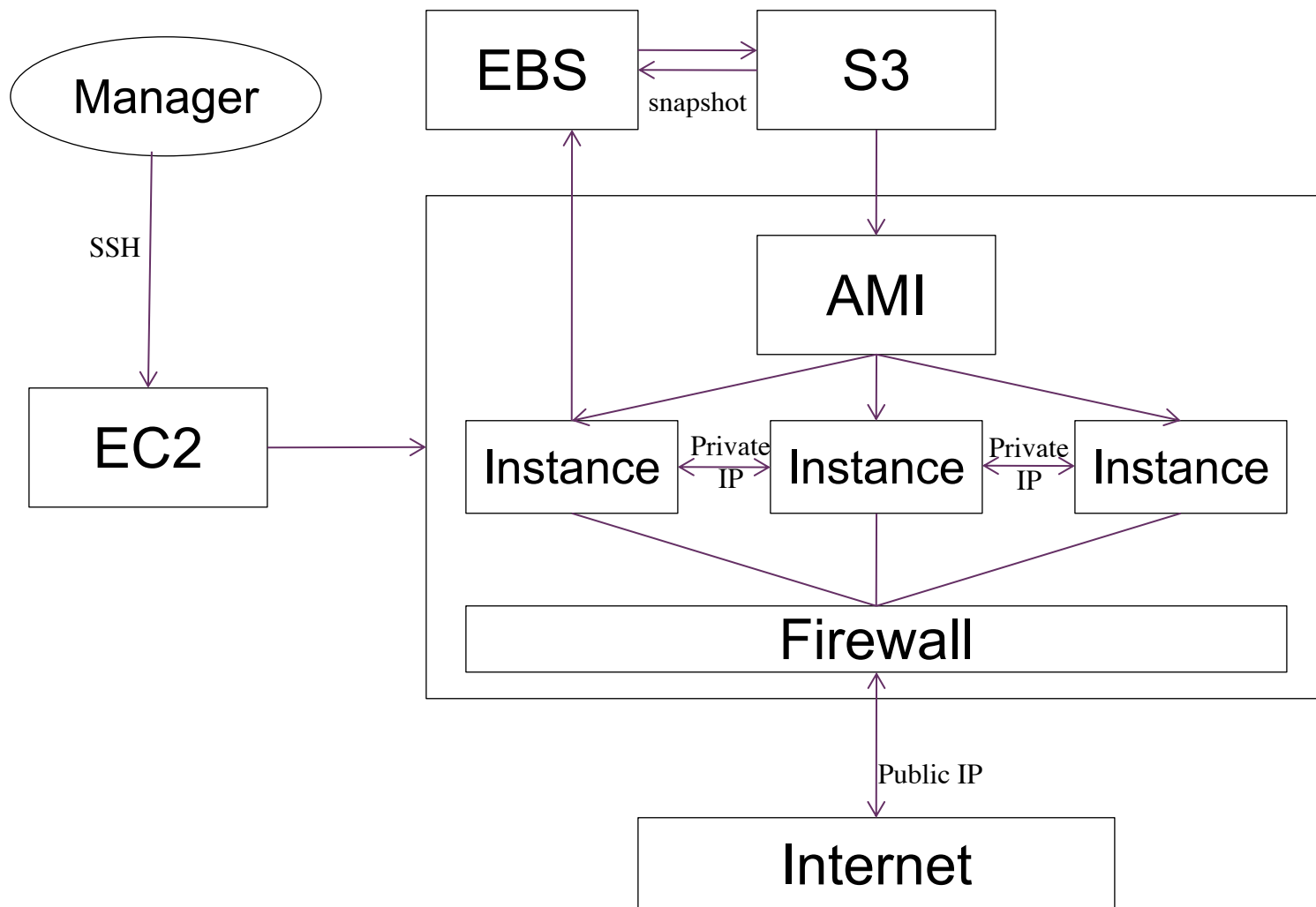


- Asia Pacific (Tokyo)

- ap-northeast-1a
- ap-northeast-1b
- ap-northeast-1c



EC2 Architecture



| Model | vCPU | CPU Credits / hour | Mem (GiB) | Storage (GB) |
|-----------|------|--------------------|-----------|--------------|
| t2.micro | 1 | 6 | 1 | EBS Only |
| t2.small | 1 | 12 | 2 | EBS Only |
| t2.medium | 2 | 24 | 4 | EBS Only |

| Model | vCPU | Mem (GiB) | SSD Storage (GB) |
|------------|------|-----------|------------------|
| c3.large | 2 | 3.75 | 2 x 16 |
| c3.xlarge | 4 | 7.5 | 2 x 40 |
| c3.2xlarge | 8 | 15 | 2 x 80 |
| c3.4xlarge | 16 | 30 | 2 x 160 |
| c3.8xlarge | 32 | 60 | 2 x 320 |

Use Cases

High performance front-end fleets, web-servers, on-demand batch processing, distributed analytics, high performance science and engineering applications, ad serving, batch processing, MMO gaming, video encoding, and distributed analytics.

| Model | vCPU | Mem (GiB) | SSD Storage (GB) |
|------------|------|-----------|------------------|
| m3.medium | 1 | 3.75 | 1 x 4 |
| m3.large | 2 | 7.5 | 1 x 32 |
| m3.xlarge | 4 | 15 | 2 x 40 |
| m3.2xlarge | 8 | 30 | 2 x 80 |

| Model | vCPU | Mem (GiB) | SSD Storage (GB) |
|------------|------|-----------|------------------|
| r3.large | 2 | 15.25 | 1 x 32 |
| r3.xlarge | 4 | 30.5 | 1 x 80 |
| r3.2xlarge | 8 | 61 | 1 x 160 |
| r3.4xlarge | 16 | 122 | 1 x 320 |
| r3.8xlarge | 32 | 244 | 2 x 320 |

Use Cases

We recommend memory-optimized instances for high performance databases, distributed memory caches, in-memory analytics, genome assembly and analysis, larger deployments of SAP, Microsoft SharePoint, and other enterprise applications.

EC2 Pricing Model

- Free Usage Tier
- On-Demand Instances
 - Start and stop instances whenever you like, costs are rounded up to the nearest hour. (Worst price)
- Reserved Instances
 - Pay up front for one/three years in advance. (Best price)
 - Unused instances can be sold on a secondary market.
- Spot Instances
 - Specify the price you are willing to pay, and instances get started and stopped without any warning as the market changes.

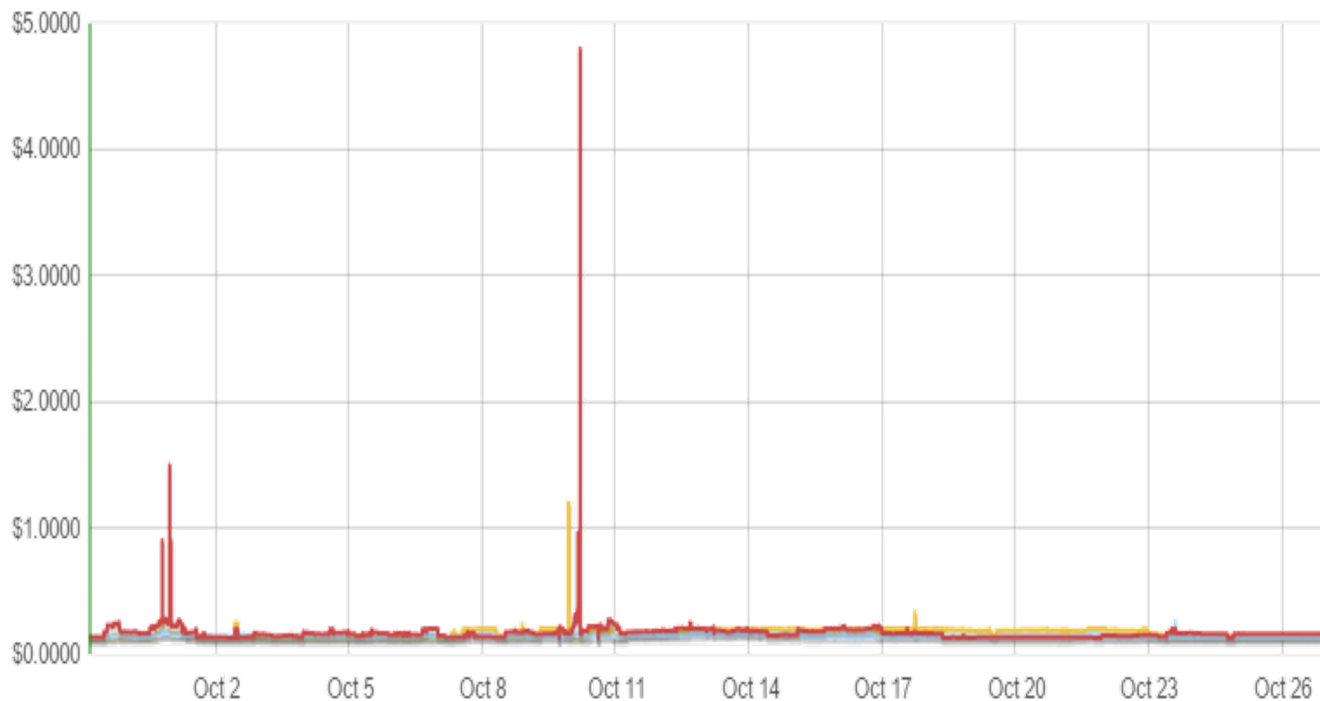
Free Usage Tier

- 750 hours of EC2 running Linux, RHEL, or SLES t2.micro instance usage
- 750 hours of EC2 running Microsoft Windows Server t2.micro instance usage
- 750 hours of Elastic Load Balancing plus 15 GB data processing
- 30 GB of Amazon Elastic Block Storage in any combination of General Purpose (SSD) or Magnetic, plus 2 million I/Os (with Magnetic) and 1 GB of snapshot storage
- 15 GB of bandwidth out aggregated across all AWS services
- 1 GB of Regional Data Transfer

Spot Instance Pricing History



Product : Linux/UNIX ▾ Instance type: c3.4xlarge ▾ Date range : 1 month ▾ Availability zone: All zones ▾



| Availability zone | Price |
|-------------------|-------|
|-------------------|-------|

| | |
|--|----------|
|  us-west-2a | \$0.1389 |
|--|----------|

| | |
|--|----------|
|  us-west-2b | \$0.1390 |
|--|----------|

| | |
|--|----------|
|  us-west-2c | \$0.1322 |
|--|----------|

| | |
|------|-------------------------------------|
| Date | September 28, 2014 9:08:53 AM UTC-4 |
|------|-------------------------------------|

Simple Storage Service (S3)

- A **bucket** is a container for objects and describes location, logging, accounting, and access control. A bucket can hold any number of **objects**, which are files of up to 5TB. A bucket has a name that must be **globally unique**.
- Fundamental operations corresponding to HTTP actions:
 - `http://bucket.s3.amazonaws.com/object`
 - POST a new object or update an existing object.
 - GET an existing object from a bucket.
 - DELETE an object from the bucket
 - LIST keys present in a bucket, with a filter.
- A bucket has a **flat directory structure** (despite the appearance given by the interactive web interface.)

S3 Weak Consistency Model

Direct quote from the Amazon developer API:

“Updates to a single key are **atomic**....”

“Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. If a PUT request is successful, your data is safely stored. However, information about the changes must replicate across Amazon S3, which can take some time, and so you might observe the following behaviors:

- A process writes a new object to Amazon S3 and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might report "key does not exist."
- A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
- A process replaces an existing object and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might return the prior data.
- A process deletes an existing object and immediately attempts to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.”

Storage Pricing

Region:

| | Standard Storage | Reduced Redundancy Storage | Glacier Storage |
|----------------------|-------------------------|-----------------------------------|------------------------|
| First 1 TB / month | \$0.0300 per GB | \$0.0240 per GB | \$0.0100 per GB |
| Next 49 TB / month | \$0.0295 per GB | \$0.0236 per GB | \$0.0100 per GB |
| Next 450 TB / month | \$0.0290 per GB | \$0.0232 per GB | \$0.0100 per GB |
| Next 500 TB / month | \$0.0285 per GB | \$0.0228 per GB | \$0.0100 per GB |
| Next 4000 TB / month | \$0.0280 per GB | \$0.0224 per GB | \$0.0100 per GB |
| Over 5000 TB / month | \$0.0275 per GB | \$0.0220 per GB | \$0.0100 per GB |

Request Pricing

Region: US Standard

Pricing

| | |
|--------------------------------------|-----------------------------|
| PUT, COPY, POST, or LIST Requests | \$0.005 per 1,000 requests |
| Glacier Archive and Restore Requests | \$0.05 per 1,000 requests |
| Delete Requests | Free † |
| GET and all other Requests | \$0.004 per 10,000 requests |
| Glacier Data Restores | Free ‡ |

† No charge for delete requests of Standard or RRS objects. For objects that are archived to Glacier, there is a pro-rated charge of \$0.03 per gigabyte for objects deleted prior to 90 days. [Learn more.](#)

‡ Glacier is designed with the expectation that restores are infrequent and unusual, and data will be stored for extended periods of time. You can restore up to 5% of your average monthly Glacier storage (pro-rated daily) for free each month. If you choose to restore more than this amount of data in a month, you are charged a restore fee starting at \$0.01 per gigabyte. [Learn more.](#)

Data Transfer Pricing

The pricing below is based on data transferred "in" to and "out" of Amazon S3.

Region: US Standard

Pricing

Data Transfer IN To Amazon S3

| | |
|----------------------|----------------|
| All data transfer in | \$0.000 per GB |
|----------------------|----------------|

Data Transfer OUT From Amazon S3 To

| | |
|--|----------------|
| Amazon EC2 in the Northern Virginia Region | \$0.000 per GB |
|--|----------------|

| | |
|---|----------------|
| Another AWS Region or Amazon CloudFront | \$0.020 per GB |
|---|----------------|

Data Transfer OUT From Amazon S3 To Internet

| | |
|--------------------|----------------|
| First 1 GB / month | \$0.000 per GB |
|--------------------|----------------|

| | |
|---------------------|----------------|
| Up to 10 TB / month | \$0.120 per GB |
|---------------------|----------------|

| | |
|--------------------|----------------|
| Next 40 TB / month | \$0.090 per GB |
|--------------------|----------------|

| | |
|---------------------|----------------|
| Next 100 TB / month | \$0.070 per GB |
|---------------------|----------------|

S3 Weak Consistency Model

Direct quote from the Amazon developer API:

“Updates to a single key are **atomic**....”

“Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. If a PUT request is successful, your data is safely stored. However, information about the changes must replicate across Amazon S3, which can take some time, and so you might observe the following behaviors:

- A process writes a new object to Amazon S3 and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might report "key does not exist."
- A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
- A process replaces an existing object and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might return the prior data.
- A process deletes an existing object and immediately attempts to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.”

EBS is approx. 3x more expensive by volume
and 10x more expensive by IOPS than S3.

Cloud Computing 101: summary

- Utility Computing – Cloud Computing for rent
- Cloud Computing infrastructures
 - Virtualization (Virtual Machines, Containers, Orchestration)
 - Caching, Replication, Sharding
 - Eric Brewer' CAP theorem: can't have:
 - Consistency & Availability & Partition-tolerance
- Amazon Web Services tour:
 - Instance types, pricing models, & other terminology
 - Compute: **EC2**, **Lambda**
 - Storage: **S3**, **EBS**, **Glacier**
 - Data: **RDS**, **Kinesis**, **DynamoDB**, **Aurora**, **Redshift**, **Athena**