Lecture 7 - Cloud Storage

AWS S3

Simple Storage Service

Simple Storage Service

- **Amazon Simple Storage Service** is storage for the Internet, designed to make web-scale computing easier for developers
- Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data
- It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure
- The service aims to maximize benefits of scale and to pass those benefits on to developers

- Amazon S3 is intentionally built with a minimal feature set that focuses on simplicity and robustness
- S3 Buckets are the fundamental container in Amazon S3 for data storage
- Buckets are region specific

- You can Store data in **Buckets**
  - Store an infinite amount of data in a bucket
  - Upload as many objects as you like into a bucket
  - Each object can contain up to 5 TB of data
  - Each object is stored and retrieved using a unique developer-assigned key

- S3 offers a range of storage classes designed for different use cases
  - **STANDARD** for general-purpose storage of frequently accessed data
  - **STANDARD_IA** (infrequent access) for long-lived, but less frequently accessed data
  - **GLACIER** for long-term archive

- As of 2022 the pricing structure for Standard S3 (US East) is broken down into 3 tiers
- For the first 50 TB - $0.023 per GB / per Month
- For the next 450 TB - $0.022 per GB / per Month
- Anything over 500 TB - $0.021 per GB / Month
- So for 10TBs worth of data we are talking about ~$235 USD a month, which sounds like a lot but it really isn't

S3 Buckets

S3 Buckets

- A bucket is a unique container for objects stored in S3
- Bucket names are unique globally and not just to your account
- Every object is contained in a bucket
- Buckets serve several purposes:
  - They organize the Amazon S3 namespace at the highest level
  - They identify the account responsible for storage and data transfer charges

S3 Objects & Keys

S3 Objects

- Objects are the fundamental entities stored in S3
- Objects consist of object data and metadata
- The data portion is opaque to Amazon S3
- The metadata is a set of name-value pairs that describe the object

S3 Keys

- A key is the unique id for an object in a bucket
- Every object in a bucket has exactly one key
- the combination of a bucket, key, and version ID uniquely identify each object

S3 API

S3 API

- The Amazon S3 architecture is designed to be programming language-neutral
- S3 uses the **REST** interface standard
  - Uses standard HTTP verbs for manipulating buckets and objects
- The REST API uses the standard HTTP headers and status codes, so that standard browsers and toolkits work as expected

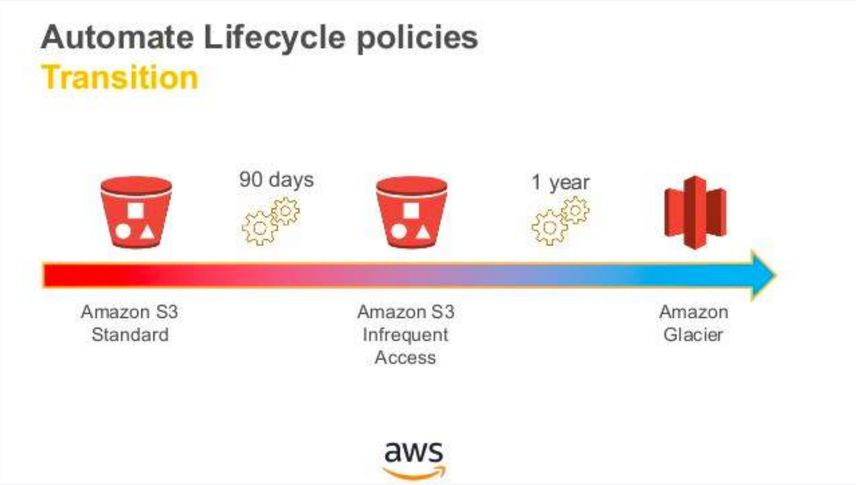
S3 Bucket Encryption

S3 Bucket Encryption

- S3 default encryption provides a way to set the default encryption behaviour for an S3 bucket
- You can set default encryption on a bucket so that all objects are encrypted
- You have the following options
  - Use Server-Side Encryption
    - You request Amazon S3 to encrypt your object before saving it on disks in its data centres and decrypt it when you download the objects
  - Use Client-Side Encryption
    - You can encrypt data client-side and upload the encrypted data to Amazon S3
- Server-side encryption uses either of the following ciphers:
  - Amazon S3-managed keys (SSE-S3)
  - AWS KMS-managed keys (SSE-KMS)

S3 Lifecycle Management

S3 Lifecycle Management



- To manage your objects so that they are stored cost effectively throughout their lifecycle, configure their lifecycle
- A lifecycle configuration is a set of rules that define actions that Amazon S3 applies to a group of objects
- **Transition actions**
  - Define when objects transition to another storage class
  - Example, you might choose to transition objects to the STANDARD_IA storage class 30 days after you created them
- **Expiration actions**
  - Define when objects expire
  - S3 deletes expired objects on your behalf

Amazon Glacier

Amazon Glacier

- Amazon Glacier is an extremely low-cost storage service that provides durable storage with security features for data backup
- Customers can store their data cost effectively for months, years, or even decades
- Getting Started
  - Glacier provides a console on the web in which you can use to create and delete vaults
  - All other interactions with Amazon Glacier require that you use the AWS Command Line Interface (AWS CLI) or write code

- S3 supports lifecycle configuration on an S3 bucket, which enables you to transition objects to the S3 GLACIER
- Although the objects are stored in Glacier, they remain S3 objects that you manage in S3
- A vault is a container for storing archives
  - https://**<region>**/**<account-id>**/vaults/**<vaultname>**
  - **Region** - glacier.us-west-2
  - **Account ID** - AWS account ID that owns the vault
  - **Vault Name** - identifies a specific vault in the vaults collection

- An AWS account can create vaults in any supported AWS Region
- An account must use unique vault names
  - An AWS account can create same-named vaults in different Regions
- Store an unlimited number of archives in a vault
- Amazon Glacier supports various vault operations. Note that vault operations are Region specific

- Jobs can perform a select query on an archive, retrieve an archive, or get an inventory of a vault
  - https://**<region>**/**<account-id>**/vaults/**<vault-name>**/jobs/**<job-id>**
- You initiate a job providing a SQL query and list of Amazon Glacier archive objects
- Glacier Select runs the query in place and writes the output results to S3
- Retrieving an archive and vault inventory are asynchronous operations
- Amazon Glacier maintains information such as job type, description, creation date, completion date, and job status

- Jobs take time to complete, Amazon Glacier supports a notification mechanism to notify you when a job is complete
- You can configure a vault to send notification to an SNS topic when jobs complete
- You can specify one SNS topic per vault in the notification configuration

Static Website Hosting

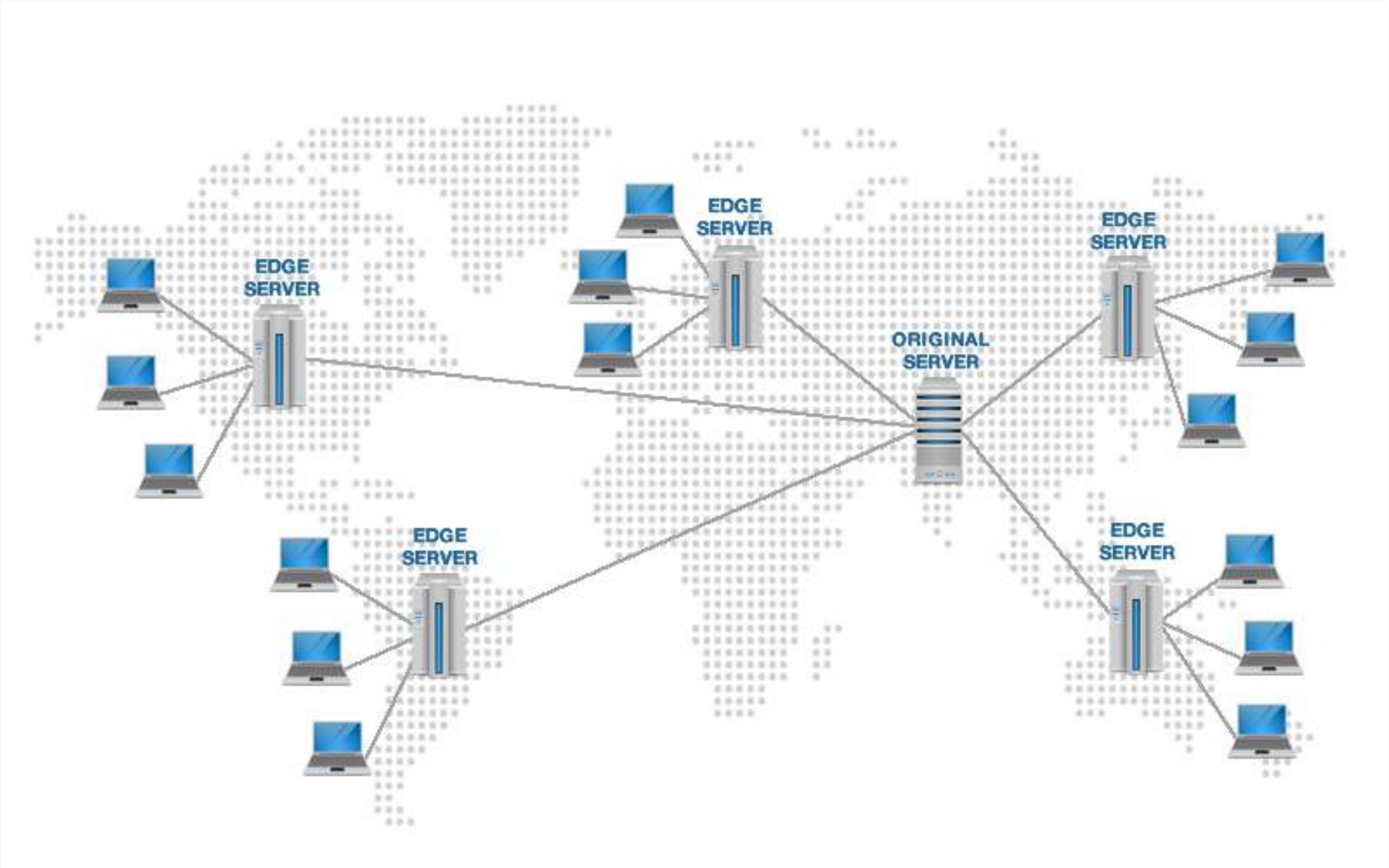
Static Website Hosting

- You can host a static website on S3
- To host a static website
  - Configure a bucket for website hosting
  - Upload your static website content
  - Bucket must have public read access
- Website endpoint will be in the format
  - *<bucket_name>*.**s3-website-***<region>***.amazonaws.com**

CloudFront

CloudFront

- **Amazon CloudFront** is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency, high transfer speeds, all within a developer-friendly environment
- CDN or Content Delivery Network is a set of geographically distributed network of proxy servers
- This means that there are servers physically located all throughout the world because distance means speed
- When an asset is cached through a CDN the first land is slow because it’s a cache miss, aka the asset isn’t in the current cache
- From each subsequent fetch, anyone that’s physically located near that server will be served with a copy located on that geographically close server which is much faster


EDGE SERVER
EDGE SERVER
EDGE SERVER
ORIGINAL SERVER
EDGE SERVER
EDGE SERVER

- Once you have an S3 bucket set up with a static website you can then use it with Cloudfront for a proper release with a named url

- A few big drawbacks of S3 is proper SSL support and the deterministic URLs

- **Cloudfront** resolves those shortfalls by allowing you to set up SSL, origin rules, differing behaviours as well as being behind a **CDN** for faster delivery to your customers

- It’s also near infinitely scalable, as long as you have the money

Azure’s Implementation

Azure Blobs

Azure Blobs

- Similar to AWS S3 by hosting files (blobs) in Buckets
- Can be accessed via an HTTPs url, Azure Storage REST API, Azure PowerShell, or a language specific library
- Maximum object size is 4.7TB
- So for 10TBs worth of data we are talking about ~$262.50 USD a month vs ~$235 for AWS

Google’s Implementation

Google Storage

Google Storage

- Google Storage is also very similar to Amazon S3
- As with the other two platforms there is a maximum size of 5TB for individual objects
- There is an update limit of once per second on each object; that means rapid writes to a single object won't scale
- Objects are addressable via HTTP(S) URLs and Google has its own CLI, called gsutil, as well as APIs, and web graphical user interface (GUI) access

- Google Cloud Storage comes in four classes characterized by availability, access time and cost:

Storage Class	Name for APIs and gsutil	Minimum storage duration	Typical monthly availability
Standard Storage	STANDARD	None	• >99.99% in multi-regions and dual-regions • 99.99% in regions
Nearline Storage	NEARLINE	30 days	• 99.95% in multi-regions and dual-regions • 99.9% in regions
Coldline Storage	COLDLINE	90 days	• 99.95% in multi-regions and dual-regions • 99.9% in regions
Archive Storage	ARCHIVE	365 days	• 99.95% in multi-regions and dual-regions • 99.9% in regions

- So for 10TBs worth of data we are talking about ~$210 USD a month vs ~$235 for AWS
- That’s comparable but Google services are typically more expensive than AWS

So why are we learning about cloud storage?

SLAs

A **Service-Level Agreement** (**SLA**) is a commitment between a service provider and a client

SLAs

- SLA is a **contract** between you and your customers
- This contract states that you guarantee a certain amount of "**uptime**" - meaning the availability of your company's sites or services using Uptime
- It also can mean having to **meet certain other metrics** that are worked out with the customer beforehand
- Failing to meet your SLA could result in **liabilities for damage**, **loss of work**, and **cash penalties**!

How do we attempt to meet our SLA’s?

The 9s of uptime

The 9s of uptime

- A very common approach is to measure uptime in percentage of availability per year
- If your site is up 90% of the time it means that your site could be down for up to 2h 30ms every day (https://uptime.is/90)
- That sounds HORRIBLE?! Who would want to use a service that's down constantly.
- Most people refer to it as how many 9s you can achieve

- Great Calculator: https://uptime.is

Number of 9's	Uptime %	Daily Allowed Outages	Weekly Allowed Outages	Monthly Allowed Outages	Quarterly Allowed Outages	Yearly Allowed Outages
1	90	2h 24m 0s	16h 28m 0s	3d 1h 2m 54s	9d 3h 8m 43s	36d 12h 34m 55s
2	99	14m 24s	1h 40m 48s	7h 18m 17s	21h 54m 52s	3d 15h 39m 29s
3	99.9	1m 26s	10m 4s	43m 49s	2h 11m 29s	8h 45m 56s
4	99.99	8s	1m	4m 22s	13m 8s	52m 35s
5	99.999	~0s	6s	26s	1m 18s	5m 15s

12 Factor App

12 Factor App

- **12 Factor** is a methodology and set of best practices for building large scale applications
- As the name implies there are 12 “rules” that you should know and strive to follow
- Like most development best practices sometimes you can’t blindly follow them because of some complication or business rule
- You can find it at https://12factor.net/

#	Factor	Description	Summary
I	***Codebase***	There should be exactly one codebase for a deployed service with the codebase being used for many deployments.	Use GIT
II	***Dependencies***	All dependencies should be declared, with no implicit reliance on system tools or libraries.	Use Library file like Gemfile, package.json, etc…
III	***Config***	Configuration that varies between deployments should be stored in the environment.	Store secrets outside of repo when you can Use ENV variables to relay secrets
IV	***Backing services***	All backing services are treated as attached resources and attached and detached by the execution environment.	Treat all external dependancies as replaceable
V	***Build, release, run***	The delivery pipeline should strictly consist of build, release, run.	Only have build, release, run. Nothing weird and artisanal.
VI	***Processes***	Applications should be deployed as one or more stateless processes with persisted data stored on a backing service.	Don't store files on the app server, use an external DB to store them or something like S3
VII	***Port binding***	Self-contained services should make themselves available to other services by specified ports.	Use ports and IPs to to intercommunicate, not some other special way
VIII	***Concurrency***	Concurrency is advocated by scaling individual processes.	Make sure that the way you program the app can be run in parallel
IX	***Disposability***	Fast startup and shutdown are advocated for a more robust and resilient system.	The server the app on can disappear at a moments notice, make sure you can survive that
X	***Dev/Prod parity***	All environments should be as similar as possible.	Dev, staging, production should be essentially the same behaving. No weird DEV only stuff
XI	***Logs***	Applications should produce logs as event streams and leave the execution environment to aggregate.	Log everything. Keep those logs on another service like S3
XII	***Admin Processes***	Any needed admin tasks should be kept in source control and packaged with the application.	Admin tasks must be programmed, be in the REPO, and be able to be executed by the app

- When we use an external Storage Service to host our files we no longer have to concern ourselves on where things are located inside our environments

- Instead we can have files be safely and securely handled while we **spin up** or **spin down** servers

- Using services like **AWS Elastic Beanstalk** or **AWS Elastic Container Service** we can scale our app in ways that were only dreamt of 20 years ago

- Besides a few incidents in the past few years, S3's uptime is 99.9999% which is unheard of

- Lastly, we didn't have to engineer the solution ourselves or maintain it especially for how cheap it is!

In Summary…

In Summary

- When needing to have files that are non ephemeral, always use a Cloud Storage service like S3, Azure Blobs etc…
- They are all pretty much the same in terms of features, but I'd suggest AWS S3
- For every 9 of uptime you guarantee, you need to almost square the cost associated
- 12 Factor is a set of rules to help you expand your companies applications

Questions?

Lecture 8 - Cloud Networking

DARPA / ARPANET

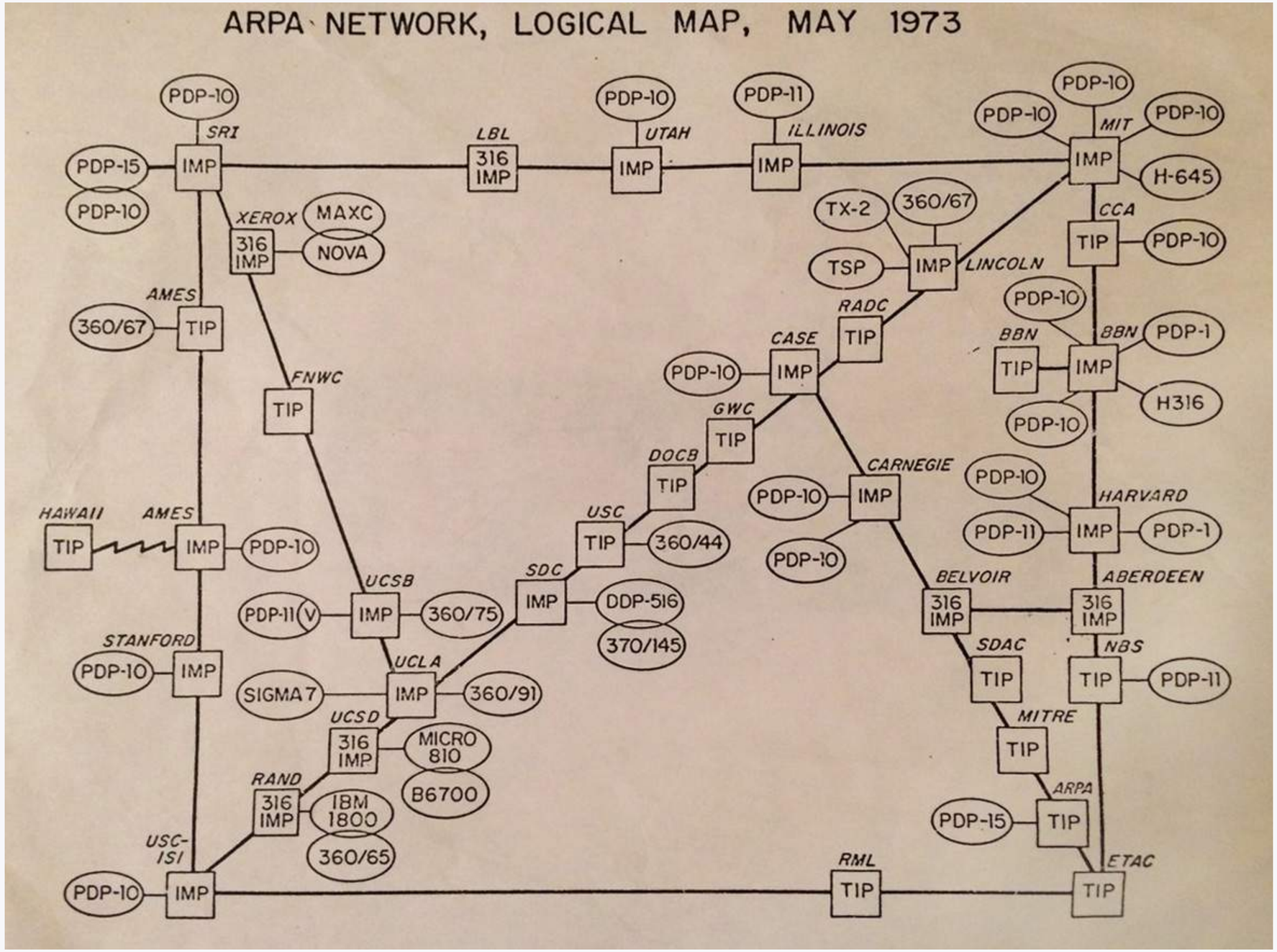
DARPA / ARPANET

- The internet all starts with **Defense Advanced Research Projects Agency** (**DARPA**)
- Communication between military installations prior to ~1969 were **Dedicated Circuits** based which was expensive
- Each Dedicated Circuit had to be run by a telephone company that was government approved and had to be actively maintained

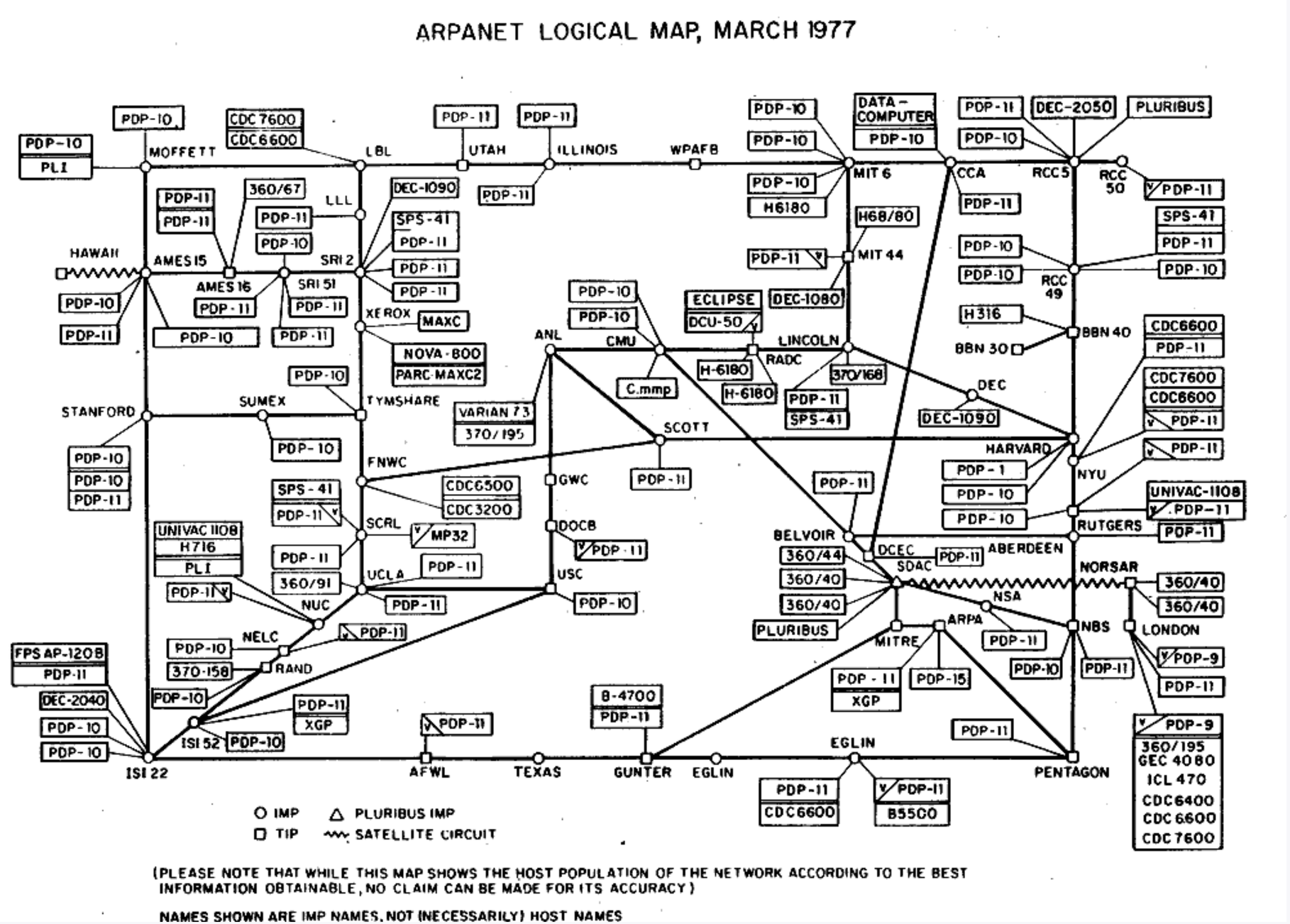
- In early ~1966 work began on this radical concept of packet based switching for a new project called **Advanced Research Projects Agency Network** (**ARPANET**)

- Instead of Dedicated Circuits to each installation there would be a way to send a message to a single installation and for it to be smart enough to route to the correct one

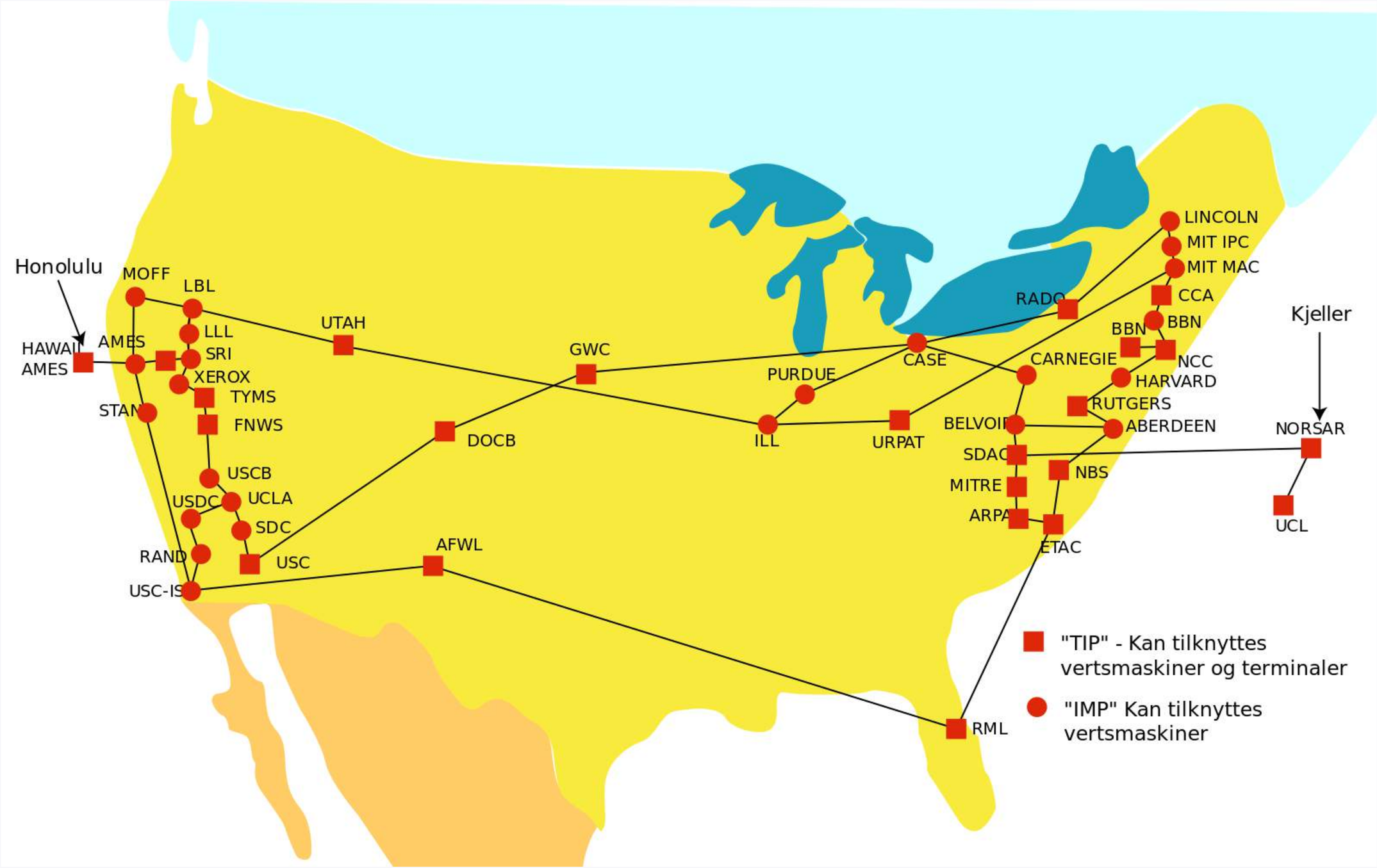
- In ~1969 the first connection was established with a 56Kbps connection



SRC: https://en.wikipedia.org/wiki/ARPANET



SRC: https://en.wikipedia.org/wiki/ARPANET



SRC: https://en.wikipedia.org/wiki/ARPANET

The University Connection

The University Connection

- In 1982, the **Internet protocol suite** (**TCP/IP**) was introduced as the standard networking software stack in the **ARPANET**

- In the early 1980s, the **National Science Foundation** (**NSF**) funded the establishment of national supercomputing centers at several universities, and provided network access and network interconnectivity with the NSFNET project in 1986

- The ARPANET project was formally decommissioned in 1990, after partnerships with the telecommunication industry paved the way of future commercialization of a new world-wide network, known as the Internet

TCP/IP

TCP/IP

- **TCP/IP** is the protocol developed in the 80's, which rose to prominence in 1983, that is still being used to this day

- **Transmission Control Protocol** (**TCP**) provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network

- **Internet Protocol** (**IP**) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries

- The story goes that two telecommunications engineers, **Vint Cerf** and **Bob Kahn**, designed the internet working protocol for sharing resources using packet switching among network nodes over breakfast at a conference

- TCP ensures that packets that are “lost” in the system get retransmitted, therefore the entire data stream reaches its destination

- Another protocol exists called **User Datagram Protocol** (**UDP**) which fires packets at the destination but doesn’t check if they’ve reached it

- It’s also much more lightweight because it doesn’t have the error checking of TCP

The First Routing Systems

The First Routing Systems

- During the early days of the ARPANET each host (computer) on the network had a list of addresses of all the other computers on the network
- This is a horrible way to do things, for every host you need to notify all the other hosts about the update
- That’s a lot of manual work (not many automated systems back then)

- A couple of engineers got together and tried to figure out a way to propagate updates throughout the network without having to manually tell each host

- A governing body arose to guide standards for computer hardware and software in the 80s called the **Internet Engineering Task Force** (**IETF**) which introduced a formal proposal for standards called **Request For Comments** (**RFC**)

- In 1988 one of the first RFC for Routing protocol was proposed called **RIPv1**

RIPv1

- **Routing Information Protocol** (**RIP**) is one of the oldest distance-vector routing protocols which employ the hop count as a routing metric

- **Distance-vector routing protocols** measure the distance by the number of routers a packet has to pass, one router counts as one hop, preferring the route with the least amount of hops

- When RIP is starting up, and every 30 seconds thereafter, a router with RIPv1 implementation broadcasts to 255.255.255.255 a request message through every RIPv1 enabled interface

- Neighbouring routers receiving the request message respond with a RIPv1 segment, containing their routing table

IPv4

IPv4

- **IPv4** is the forth version of the Internet Protocol
- The first version of the internet protocol developed for ARPANET only had a maximum of 255 records for hosts, this clearly isn't enough
- The second and third versions were prototypes that never really went anywhere
- The fourth version, one of the ones we use today, allows for a maximum of **4,294,967,296**

- IPv4 addresses are written in dot-decimal notation, which consists of four octets of the address expressed individually in decimal numbers and separated by periods
  - Ex: **192.168.1.1**
- So in theory the highest possible IP address is 255.255.255.255 but that address is reserved for what's called the **broadcast address**
- There is also the inverse of a broadcast called a network address which is the lowest possible address in a subnet (192.168.0.0)
- The lowest possible IP address is 1.1.1.1

- While **4,294,967,296** possible addresses SEEMS like a lot we are quickly running out
- Between home internet connections, cell phones, servers, IP hoarders, routers, etc… we have reached a boiling point of IPs
- What are we going to do?

Running Out Of IPs

Running Out Of IPs

- This isn't the first time we've "Ran out of IPs"
- Back in the old days EVERY computer on the network got a "Public" IP address
- **Classless Inter-Domain Routing** (**CIDR**) was introduced in 1993 to help solve this problem
- CIDR was designed to permit repartitioning of any address space so that smaller or larger blocks of addresses could be allocated to users
- What that means is it separates the larger network into smaller sub networks within

- This is where the “Public” vs “Private” IP address scheme came from
- **Every major entry point** (think router) would be given one or more public IP address(es)
- Behind this would be a private network with many more private IP addresses
- These private IP addresses would be given to computers or hosts

- The entry point will keep a record of which Private IP address requested what Public IP address and over which port they tried to communicate

- It would then assign a returning port for the remote service to attempt to communicate back to

- For example Host A would hit example.com on port 80, and the router will tell it to return communicate on port 42019

- If the returning service sends a communication back it will know to whom it was talking to by that returning port

Subnetting

Subnetting

- A Subnet mask is a 32-bit number that masks an IP address, and divides the IP address into network address and host address

- Subnet Mask is made by setting network bits to all "1"s and setting host bits to all “0"s

- Within a given network, two host addresses are reserved for special purpose, and cannot be assigned to hosts. The "0" address is assigned a network address and "255" is assigned to a broadcast address, and they cannot be assigned to hosts.

SRC: https://www.iplocation.ne

- A Subnet mask is a 32-bit number that masks an IP address, and divides the IP address into network address and host address

- Subnet Mask is made by setting network bits to all "1"s and setting host bits to all “0"s

- Within a given network, two host addresses are reserved for special purpose, and cannot be assigned to hosts. The "0" address is assigned a network address and "255" is assigned to a broadcast address, and they cannot be assigned to hosts.

SRC: https://www.iplocation.ne

Name	CIDR block	Address range	Number of addresses	Classful description
Class A Subnet	10.0.0.0/8	10.0.0.0 – 10.255.255.255	16,777,216	Single Class A.
Class B Subnet	172.16.0.0/12	172.16.0.0 – 172.31.255.255	1,048,576	Contiguous range of 16 Class B blocks.
Class C Subnet	192.168.0.0/16	192.168.0.0 – 192.168.255.255	65,536	Contiguous range of 256 Class C blocks.

SRC: https://www.iplocation.ne

Address block	Address range	Number of addresses	Scope	Description
0.0.0.0/8	0.0.0.0 – 0.255.255.255	16,777,216	Software	Current network (only valid as source address).
10.0.0.0/8	**10.0.0.0 – 10.255.255.255**	**16,777,216**	**Private network**	**Used for local communications within a private network.**
100.64.0.0/10	100.64.0.0 – 100.127.255.255	4,194,304	Private network	Shared address space for communications between a service provider and its subscribers when using a carrier-grade NAT.
127.0.0.0/8	**127.0.0.0 – 127.255.255.255**	**16,777,216**	**Host**	**Used for loopback addresses to the local host.**
169.254.0.0/16	169.254.0.0– 169.254.255.255	65,536	Subnet	Used for link-local addresses between two hosts on a single link when no IP address is otherwise specified, such as would have normally been retrieved
172.16.0.0/12	**172.16.0.0 – 172.31.255.255**	**1,048,576**	**Private network**	**Used for local communications within a private network.**
192.0.0.0/24	192.0.0.0 – 192.0.0.255	256	Private network	IETF Protocol Assignments.
192.0.2.0/24	**192.0.2.0 – 192.0.2.255**	**256**	**Documentation**	**Assigned as TEST-NET-1, documentation and examples.**
192.88.99.0/24	192.88.99.0 – 192.88.99.255	256	Internet	Reserved. Formerly used for IPv6 to IPv4 relay (included IPv6 address block 2002::/16).
192.168.0.0/16	**192.168.0.0 – 192.168.255.255**	**65,536**	**Private network**	**Used for local communications within a private network.**
198.18.0.0/15	198.18.0.0–198.19.255.255	131,072	Private network	Used for benchmark testing of inter-network communications between two separate subnets.
198.51.100.0/24	**198.51.100.0 – 198.51.100.255**	**256**	**Documentation**	**Assigned as TEST-NET-2, documentation and examples.**
203.0.113.0/24	203.0.113.0 – 203.0.113.255	256	Documentation	Assigned as TEST-NET-3, documentation and examples.
224.0.0.0/4	**224.0.0.0 – 239.255.255.255**	**268,435,456**	**Internet**	**In use for IP multicast. (Former Class D network).**
240.0.0.0/4	240.0.0.0 – 255.255.255.254	268,435,456	Internet	Reserved for future use. (Former Class E network).
255.255.255.255/32	**255.255.255.255**	**1**	**Subnet**	**Reserved for the "limited broadcast" destination address**

SRC: https://en.wikipedia.org/wiki/IPv4

More Advanced Routing

More Advanced Routing

- As networks grew and **Internet Service Providers** (**ISP**) got bigger they needed to get more complicated with their **peering agreements**

- A peering agreement is the written contract that 2 ISPs can send network traffic over each others network

- This could be for each terabyte of data or it could be carte blanche for all traffic

- ISPs came up with a protocol called the **Border Gateway Protocol** (**BGP**) in 1994
- We are currently on BGP4
- Each ISP using BGP is required to obtain an **Autonomous System Number** (**AS Number**) which is basically its ID
- I won’t go into detail on BGP because that’s an entire course

IPv6

IPv6

- **IPv6** is the next evolution of IPv4
- You may ask, what happened to **IPv5**?
- The answer is it was a prototype protocol that was going to be used with video and audio over the internet until some engineer figured out how to do it with TCP/IPv4 so they abandoned it
- IPv6 is **128 bits long** instead of the 32 bits of IPv4
- That means it has approximately **3.4×10^{38} IP addresses** available for use

- IPv6 addresses are incredibly long and hard to remember
- For example:
  - Loop back (localhost) is **0000:0000:0000:0000:0000:0000:0000:0001** or abbreviated to **::1**
  - Another address can be **2001:0db8:0000:0000:0000:ff00:0042:8329** which can be abbreviated to **2001:db8:0:0:0:ff00:42:8329** by removing all leading zeros OR **2001:db8::ff00:42:8329** after omitting consecutive sections of zeros.
- As you can tell it’s incredibly hard to remember AND it’s complicated in the number scheme

- I won’t cover much of IPv6 because to be honest it’s not widely used
- It’s a bad thing that we haven’t whole-heartedly switched over, but old habits die hard and people are terrified about the number scheme
- Some day it will become the default standard on the internet
- Until then we run what’s called a **Dual Stack** mode, meaning our internet interfaces on servers have both an **IPv4** and a **IPv6** address so they can be routed to by either

DHCP

DHCP

- **Dynamic Host Configuration Protocol** (**DHCP**) is a network management protocol that assigns IP addresses from a pool to requesting devices

- It's the thing on your router that hands out IP address so you don't have to manually input them into each device

- While not explicit required on all networks, the amount of work needed to manually do its job is just not worth it so people typically use them

VPC

VPC

- A **Virtual Private Cloud** (**VPC**) is essentially your own private subnet inside of a Cloud Provider
- It has a DHCP type service that hands out internal IP addresses
- It allows intercommunication between servers and services within the VPC without having to expose external ports

Static vs Dynamic IP

Static vs Dynamic IP

- Typically when an IP address is handed to you (public or private) it has a finite life before it's released back to the pool of IP addresses

- We call these IP addresses **Dynamic IPs** because you can't guarantee which you'll have a week from now (for the most part)

- For things like laptops or phones this is absolutely fine, but for things that are serving a service that is unacceptable

- For things that many people need to reach the IP address needs to be a fixed constant
- You can purchase a lease to an IP address from a Cloud Service Provider for anywhere from free to a small monthly fee based on the amount of traffic flows to it
- From there we associate that **Static IP** address to the service or server and then assign a DNS record for ease of use

- For AWS this is called an **Elastic IP Address**
  - AWS allows you to dynamically disassociate and associate the Elastic IP at will from their console
- For Azure you can at the time of creation request a Static IP address for that VM
  - When I first used Azure that was the only time you could associate a Static IP
  - It appears now you can at will associate and disassociate Static IPs

Ports

Ports

- In networking a port is a communication endpoint
- In software it’s a local construct that identifies a process or two of network service
- It’s a 16-bit unsigned integer that ranges from **0** to **65535**
- The **Internet Assigned Numbers Authority** (IANA) is responsible for the global coordination of the DNS Root, IP addressing, and other Internet protocol resources. This includes the registration of commonly used port numbers for well-known Internet services
- These well known ports range from **0 - 1023**

Port number	Assignment
20	**File Transfer Protocol** (**FTP**) Data Transfer
21	**File Transfer Protocol** (**FTP**) Command Control
22	**Secure Shell** (**SSH**) Secure Login
23	**Telnet** remote login service, unencrypted text messages
25	**Simple Mail Transfer Protocol** (**SMTP**) E-mail routing
53	**Domain Name System** (**DNS**) service
67, 68	**Dynamic Host Configuration Protocol** (**DHCP**)
80	**Hypertext Transfer Protocol** (**HTTP**) used in the World Wide Web
110	**Post Office Protocol** (**POP3**)
119	**Network News Transfer Protocol** (**NNTP**)
123	**Network Time Protocol** (**NTP**)
143	**Internet Message Access Protocol** (**IMAP**) Management of digital mail
161	**Simple Network Management Protocol** (**SNMP**)
194	**Internet Relay Chat** (**IRC**)
443	**HTTP Secure** (**HTTPS**) HTTP over TLS/SSL

SRC: https://en.wikipedia.org/wiki/Port_(computer_networking)#Common_port_numbers

DNS

The **Domain Name System** (**DNS**) is a *hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network*

DNS

- **Domain Name System** (**DNS**) is essentially IP Addresses with an easy to remember name
- When you type a URL into your browser it has to go through a few steps in order to determine if you can find the address
- This evolved during the early stages of ARPANET when both US Military personal and University Faculty got sick of memorizing locations so they built a naming system to get around it

- Let’s use the following domain as an example: http://www.google.ca.

1. http:// is the protocol
2. www is the subdomain
3. google is the domain
4. ca is the top level domain
5. . at the end is root

- When your computer goes to look up http://www.google.ca. it has to look it up in reverse order starting with the period at the end

    1. Root Name Server Lookup
    2. Top Level Domain Lookup
    3. Domain Lookup
    4. Sub-Domain Lookup

- Your computer has no idea where to find anything on the internet; how would it?
- Each layer of the DNS query depends on the layer before it.

http://	www	google	ca	.
Protocol	Sub-Domain	Domain Name	Top Level Domain	Root Domain

Root Name Server

- Have you ever typed a period at the end of a URL?
- It’s assumed to be there so software tends to hide it. Only when you are dealing in DNS record management do you care about it
- Your computer has no idea where to find any site ending with ca

- The manufacturer of your operation system (Apple, Microsoft, Google, FLOSS) has a list of **13 Root name servers** and **their IP addresses**

- These Root name servers are operated by various people including **businesses**, **Universities**, **Government Facilities**, and the **US Army**

- These Root name servers know the IP address of the Top Level Domain (TLD) name servers like .com, .ca, .org etc…

Letter	IPv4 address	IPv6 address	Operator
A	198.41.0.4	2001:503:ba3e::2:30	Verisign
B	199.9.14.201	2001:500:200::b	USC-ISI
C	192.33.4.12	2001:500:2::c	Cogent Communications
D	199.7.91.13	2001:500:2d::d	University of Maryland
E	192.203.230.10	2001:500:a8::e	NASA Ames Research Center
F	192.5.5.241	2001:500:2f::f	Internet Systems Consortium
G	192.112.36.4	2001:500:12::d0d	Defense Information Systems Agency
H	198.97.190.53	2001:500:1::53	U.S. Army Research Lab
I	192.36.148.17	2001:7fe::53	Netnod
J	192.58.128.30	2001:503:c27::2:30	Verisign
K	193.0.14.129	2001:7fd::1	RIPE NCC
L	199.7.83.42	2001:500:9f::42	ICANN
M	202.12.27.33	2001:dc3::35	WIDE Project

Top Level Domain Name Server

- **Top Level Domain** (**TLD**) contain the DNS records for all domains underneath it
- TLDs are the last part of a domain
  - For example: .com, .ca, .org, .biz
- Initially there were 7 domains but since then we have grown to hundreds

Name	Entity	Administrator
.com	commercial	Verisign
.org	organization	Public Interest Registry
.net	network	Verisign
.int	international organizations	Internet Assigned Numbers Authority
.edu	education	Educause (via Verisign)
.gov	U.S. national and state government agencies	General Services Administration (via Verisign)
.mil	U.S. military	United States Department of Defense

Your Domain

- When you register for your domain the **registar** typically hosts it for you
- This doesn't have to be the case; you can **host your own DNS servers** if you would like to
- Hosting your own is preferred if you have: a lot of automation, weird setups like split views, or a ton of records
- Despite what people say, **text based entry is always faster than GUI**

- You need to remember that DNS is just a text file on a server
- It’s easy to write your own but very archaic; it was developed in the 80s mind you
- This file is called a Zone File

DNS Zone File

A **DNS zone file** is a text file that describes a DNS zone

```
1  $TTL 300
2  @        1440    IN      SOA     dns1.staging.example.com.       root.example.com. (
3                           58                  ;serial
4                           360                 ;refresh
5                           180                 ;retry
6                           300                 ;expire
7                           360 )               ;minimum negative caching
8           IN      NS      dns1.staging.example.com.
9           IN      NS      dns2.staging.example.com.
10
11 @        IN      A       52.20.29.204
12
13 dns1     IN      A       52.72.127.230
14 dns2     IN      A       52.72.141.164
15
16
17 www      IN      A       173.254.42.132
18
19 ftp      IN      CNAME   www.example.com.
20
```

DNS Zone Files

- Zone files describe the hosts within your zone
- When you need to add a new subdomain you would put it in your zone file
- Zone Files have many different types of records but we will only be talking about 6
- Note that for changes to take effect your serial number must be at least 1 higher than the last

- **A Record**
  - Returns a 32-bit IPv4 address, most commonly used to map hostnames to an IP address of the host
  - Example: www IN A 192.167.0.1
- **AAAA Record**
  - Returns a 128-bit IPv6 address, most commonly used to map hostnames to an IP address of the host.
  - Example: wwww IN A 2001:0db8:85a3:0000:0000:8a2e:0370:7334.
- **CNAME Record**
  - A Canonical Name record (abbreviated as CNAME record) is a type of resource record in the Domain Name System (DNS) used to specify that a domain name is an alias for another domain
  - www IN CNAME web-srv1.example.com.

- **@ Record**
  - The Root record for your domain.
  - Where it goes if you don’t put any subdomain
- **MX Record**
  - Where the mail server is hosted for your domain
- **SPF Record**
  - Identifies which mail servers are permitted to send email on behalf of your domain
  - This is a failed anti-spam technique

Emailing

Emailing

- Emails are tricky because of Spam
- Spam ruins all
- Emails need to have valid DNS records (forwards and backwards) and they need to have valid SPF signatures and TXT records and so on and so forth
- It’s not worth your effort
- Use AWS SES or a similar service to send emails

HTTP Status Codes

HTTP Status Codes

- HTTP has **response codes** for almost any action
- These codes are mostly standardized
- When building or working with an API it’s in everyones best interest to follow the standardized code

Code	Meaning	What it really means
200 OK	Standard response for successful HTTP requests. The actual response will depend on the request method used.	Got the page ok
201 Created	The request has been fulfilled, resulting in the creation of a new resource	Successfully Created
301 Moved Permanently	This and all future requests should be directed to the given URI.	Moved
302 Moved Temporarily / 303	The response to the request can be found under another URI using the GET method. When received in response to a POST (or PUT/DELETE), the client should presume that the server has received the data and should issue a new GET request to the given URI	Moved
400 Bad Request	The server cannot or will not process the request due to an apparent client error (e.g., malformed request syntax, size too large, invalid request message framing, or deceptive request routing).	Bad parameters / failed validation
401 Unauthorized	Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.	Not authorized
403 Forbidden	The request was valid, but the server is refusing action. The user might not have the necessary permissions for a resource, or may need an account of some sort.	Not authenticated or blocked
404 Not Found	The requested resource could not be found but may be available in the future. Subsequent requests by the client are permissible.	Not Found
418 I'm a teapot	The RFC specifies this code should be returned by teapots requested to brew coffee.	Delicious
500 Internal Server Error	A generic error message, given when an unexpected condition was encountered and no more specific message is suitable.	Your code crashed
502 Bad Gateway	The server was acting as a gateway or proxy and received an invalid response from the upstream server.	Your application went down hard

Security Groups / ACLs

Security Groups

- **Security Groups** act as a set of firewall rules that block all ports and only allow a few predetermined ones to the servers / services attached to this security group

- **Network access control** lists (**ACLs**) Act as a firewall rules, similar to a security group, but at a subnet (VPC) level

- You can associate more than one Security Group with a server / service

Questions?

Database Management

Databases

Databases

- In this lecture we will be talking about databases, their importance, and 5 of the most popular ones used today
  - **MySQL**
  - **SQLite**
  - **MongoDB**
  - **Redis**
  - **PostgreSQL**

- This is mostly meant to be an introduction to databases and their basic management but it’s not all encompassing
- I implore you to do some additional research on databases since they are such an important part of software projects

MySQL

MySQL

- **MySQL** is an open-source **relational database management system** that was initially launched in May of 1995

- Was created by **Michael Widenius** and **David Axmark** and named after a portmanteau of Michael's daughter "**My**" and SQL

- Was primarily sponsored by Michael and Davids company **MySQL AB** until **Sun Microsystems** acquired them in January 2008

- **Sun Microsystems** was acquired by **Oracle** in 2010 which let a change in how the MySQL project was handled
- Michael left the project and the company in February of 2009 to start a new company **Monty Program AB** and **fork MySQL** into **MariaDB**
- MariaDB is named after Michael youngest daughter Maria

MySQL Users

MySQL Users

- In most SQL Languages databases are typically isolated to a single user to prevent a single application being broken into compromising the whole server
- Every database should have a User act as administrator for that database only
- Make sure to use long, secure, pseudo random passwords for these users

MySQL Permissions

MySQL Permissions

- In MySQL privileges are given out at different levels of operation
  - Administrative Privilege (Global) which enables a user(s) to manage operation of MySQL and all it’s databases
  - Database Privilege (Database Specific) enables a user(s) to manage one or more specific databases and all objects within
  - Object Privilege (Object Specific) which enables a user(s) to manage a specific set of objects, or all objects within a specific database

- As an **Administrator** your goal is always give least permissions to any one single user
- What ever is the least permissions they can have to do their job, that’s what we will be giving
- That being said, as the Administrator you have absolute authority within the MySQL Database to view all objects
- With great power comes great responsibility

MySQL Common Tasks

MySQL Common Tasks

- After you've installed an MySQL you need to do some basic security because it's wide open for attack
  - **$ mysql_secure_installation**
- To log into mysql as the root admin
  - **$ sudo mysql -u root**
- See all the Users in the system
  - **> SELECT * FROM mysql.user;**

- Create A database
  - **> CREATE DATABASE your_database_here;**
- Create a user with a password on this MySQL instance
  - **> CREATE USER 'username'@'localhost' IDENTIFIED BY 'TopsyKretPasswords';**
- Give that user absolute permissions
  - **> GRANT ALL PRIVILEGES ON your_database_here.* TO 'username'@'localhost';**

- Login as the newly created user
  - **$ sudo mysql -u username -p**
- Show all database in system (that you have permissions to see)
  - **> SHOW DATABASES;**
- Use A specific database
  - **> USE your_database_here;**
- Show tables of a specific database
  - **> SHOW TABLES;**

MySQL Default Storage Engines

MySQL Default Storage Engines

- MySQL has a few default storage engines that ship with most versions of the Application
- A database engine (or "storage engine") is the underlying software component that a DBMS uses to create, read, update and delete (CRUD) data from a database
- The two most commonly talked about MySQL Storage Engines are **MyISAM** and **InnoDB**
- Up until version 5.5 of MySQL, the default engine used if one wasn’t specified with all newly created databases would be MyISAM

- **MyISAM** had a few well know problems like an **absence of transactions** or **lacks foreign key support**

- **MyISAM** is known to not recover gracefully from hard faults and **tends to corrupt** from them

- **InnoDB** is a newer engine for MySQL and tries to correct some of these shortcomings as well as improved speed in well formed databases

- If you use MySQL make sure to always use the InnoDB engine

MySQL Best Practices

MySQL Best Practices

- Always make sure to run **mysql_secure_installation** for even just the most basic security measures like giving the root MySQL a password

- Never expose your MySQL port (**3306**) to the wide internet, you will be attacked constantly with people trying to break in

- Never ever ever use PhpMyAdmin in production, that's a one way ticket to getting everything stolen from your application

MariaDB

MariaDB

- As mentioned before **MariaDB** is a fork of MySQL by one of the original authors of MySQL, **Michael Widenius**
- A lot of conjecture is said about which is the better version of MySQL but to me it all comes down to Licensing and rights
- MariaDB is distributed under the GPL License and is fully compatible with MySQL
- MySQL is owned by Oracle which has a history of licensing shenanigans

SQLite

SQLite

- **SQLite** is a database engine that isn't a standalone app but instead a software library developers can use to embed into their apps
- It's probably the most common database deployed worldwide due to how prolific it's being used by major applications
  - Such as all web browsers, some functions on common operating systems, mobile phones, and many embedded devices

- **D. Richard Hipp** designed SQLite in the spring of 2000 while working for General Dynamics on contract with the **United States Navy**

- The design goals of SQLite were to allow the program to be operated without installing a database management system or requiring a database administrator

- It supports most of the **SQL-92** standard but has a few major drawbacks like being non-networked and locks tables for writing, giving it slow transactional speed

SQLite - Interesting Facts

- SQLite **doesn't** use **GIT SCM**, instead the D. Richard Hipp wrote his own SCM based on SQLite called **Fossil**

- SQLite uses automated regression testing prior to each release. Over 2 million tests are run as part of a release's verification

- SQLite claims to have 100% test coverage

MongoDB

MongoDB

- **MongoDB** describes it's self as a source-available cross-platform document-oriented database program
- Originally developed as a component of a platform as a product service in 2009
- MongoDB stores data in non associated JSON objects instead of rows and columns like relational databases

MongoDB Users

MongoDB Users

- MongoDB supports non admin user authentication but not by default
- You have to enable Access Controls intentionally and create an Administrator user prior to making any subordinates
- In MongoDB databases are called collections and users can be registered as access to only a single Collection

- To add a user you need to open your MongoDB shell
  - **$ mongo**
- Switch to Admin Database
  - **$ use admin**
- Create a user for the Admin Database
  - **$ db.createUser({ user: "admin", pwd: "TopsyKretPasswords", roles: [{ role: "userAdminAnyDatabase", db: "admin" }] })**

- Authenticate the newly created Admin User
  - **$ db.auth("admin", “TopsyKretPasswords”)**
- To create a user for a specific database (Collection)
  - **$ use database_1**
  - **$ db.createUser({ user: "user_1", pwd: "your_password", roles: [{ role: "dbOwner", db: "database_1" }] })**

MongoDB Security

MongoDB Security

- Most people leave a majority of settings on default, even in the professional world
- By default MongoDB is open to the whole wild world with no username or password and isn't bound to localhost or any kind of IP Range
- Due to this, thousands and thousands of companies have had their MongoDB databases scraped clean by "**Script Kiddies**"

- When setting up a MongoDB instance the first thing you need to do is edit the main configuration file located at **/etc/mongod.conf** and add the following settings

```
security:
      authorization: enabled
```

```
bind_ip=127.0.0.1
```

Redis

Redis

- **Redis** (**Remote Dictionary Server**) is an in-memory key-value database store
- The Redis project began when **Salvatore Sanfilippo** the original developer of Redis, was trying to improve the scalability of his Italian startup, developing a real-time web log analyzer
- It’s technically a NoSQL database since it’s just a simple key-value store

- Redis keeps the entire database in memory and commits to disk every so many transactions or after so long since last commit
- This allows it to be relatively safe in case of a fault from dataloss but that isn’t the intent of the product
- Redis is hugely used as an in memory cache and temporary datastore for things like backend processors and queues
- Only recently Redis started supporting user ACLs

PostgreSQL

PostgreSQL

- **PostgreSQL**, also known as Postgres, is a free and open-source **relational database management system** (**RDBMS**) emphasizing **extensibility** and **SQL compliance**

- Started as the successor to the **Ingres** project at the University of Berkeley ~1986 by former Ingres team lead **Michael Stonebraker**

- PostgreSQL is also known for being extremely reliable with a ton of useful features

- PostgreSQL supports a large amount of datatypes but mainly it support JSON document storage natively (JSONB format)

- You can search directly into the JSON documents as well as keep those fast relational lookups

- Since it can both act as traditional RDBMS as well as have the same NoSQL features as say MongoDB without the insecurity it kinda makes MongoDB redundant

PostgreSQL Users

PostgreSQL Users

- Much like MySQL, with PostgreSQL you can create users for individual databases
- Users need to be the **Owner** of a database to be able to read / write to it
- These commands can be execute by the system user “postgres” or else can be executed using the psql command

PostgreSQL Common Tasks

PostgreSQL Common Tasks

- To create a local user to use with PostgreSQL
  - **$ sudo -u postgres bash -c "psql -c \"CREATE USER psql_username WITH PASSWORD 'TopsyKretsPassword';\""**
- To create a database locally
  - **$ sudo -u postgres bash -c "psql -c \"CREATE DATABASE your_database;\""**
- To make the User the Owner of the Database
  - **$ sudo -u postgres bash -c "psql -c \"ALTER DATABASE your_database OWNER TO psql_username;\""**

- To create a remote user to use with PostgreSQL
  - **$ PGPORT=5432 PGPASSWORD=admin_password psql -h 127.0.0.1 -U admin template1 -c "CREATE USER username WITH PASSWORD 'TopsyKretsPassword';"**
- To create a database remotely
  - **$ PGPORT=5432 PGPASSWORD=admin_password psql -h 127.0.0.1 -U admin template1 -c "CREATE DATABASE your_database;"**
- To make the User the Owner of the Database remotely
  - **$ PGPORT=5432 PGPASSWORD=admin_password psql -h 127.0.0.1 -U admin template1 -c "ALTER DATABASE your_database OWNER TO username;"**

PostgreSQL Security

PostgreSQL Security

- A lot of what applies to MySQL also applies here
- Never expose your PSQL port to the wide internet, always keep that only accessible from inside your network
- Some people think port obfuscation is a way to properly secure something but in all actually it just makes it harder to remember for your staff and is fooled by 1 **nmap** scan
  - **$ nmap -T4 -A -v 10.13.37.8**

Cloud Database Services

AWS

AWS

- **Amazon Web Services** has a few options, besides hosting your own, that will cover all the database types we talked about today
  - **Amazon Relational Database Service** (**RDS**) which can host PSQL compatible database, PSQL native, or MySQL
  - **Amazon ElasticCache** can host managed Redis instances
  - **MongoDB Atlas** is a weird hybrid of managed services for MongoDB on AWS

Azure

Azure

- **Azure** has a similar options to AWS in terms of managed services
  - **Azure Database for PostgreSQL**
  - **Azure Database for MySQL**
  - **Azure Cache for Redis**
  - **Azure Cosmos DB** for Managed MongoDB

Google Cloud

Google Cloud

- **Google Cloud** has the following options
  - **Cloud SQL** for fully managed MySQL and PostgreSQL
  - **Memorystore** for fully managed Redis instances
  - **MongoDB Atlas** for fully managed MongoDB instances

Backups

Backups

- Sooner or later you will need to restore from a backup due to **migrating** to another server or managed service, or an error has occurred and **data was loss**, or someone **infiltrated** your application

- The problem arises of when to do them since backing up your database slows it down and may have to lock tables or rows to back them up properly

- So the conundrum becomes how many times a day do we trigger it?

- Through my experience I've come up with these following rules
  - **Non Production Databases**: every few hours
    - You may need to roll them back due to testing and they'll be infrequently accessed
  - **Production Databases**: at LEAST once a day, preferably twice and both on off hours
    - Middle of the night for the first, and the second backup taken near lunchtime of the heaviest users (noon-ish)

Backup Retention

- Now we have backups, where do we store them and how long do we keep them?
- Ideally you'd keep them forever but that's an ever expanding cost and they could be silently corrupting due to disk rot
- I'd suggest keeping backups for **90 days** at least
- I'd also suggest keeping them in some kind of **offsite storage** like a **private S3 bucket** (where I keep mine)

Restoring Backups

Restoring Backups

- Restoring is a critical operation to know since the only time you really need it, is when you really need it
- First thing to note, you should make sure that all apps touching the database are either turned off or disallowed to commit during restoration process
- Second thing you need to do is work in pairs during a restore procedure

- Now why should you work in pairs?
- Since when you are restoring there's a good chance that you will be under some pressure from a failure you have a higher chance to make mistakes
- When doing any kind of major work on a production type system you **ALWAYS write your command out in a text document**, **have your team mate read it allowed back to you**
- If neither of you find an issue during review, you can go ahead an execute it but if there's any "**sniff**" stop and dig deeper

How to restore a Backup

- PSQL Restore
  - **$ psql --host=127.0.0.1 --dbname=your_database --username=username --password --file=restore_file.sql**
- MySQL Restore
  - **$ mysql --host=127.0.0.1 --user=username --password your_database < restore_file.sql**

How to restore a Backup - Image

```
PSQL Restore
    psql --host=127.0.0.1 --dbname=your_database --username=username --password --file=restore_file.sql

MySQL Restore
    mysql --host=127.0.0.1 --user=username --password your_database < restore_file.sql
```

Tools

Tools

- Sometimes you need to go into the database and manually tweak something
  - Never ever edit the schema manually, always allow the developer to deploy changes via ORMs!!
- The tools I use professionally are either
  - For PSQL on **MacOS** there is none better than **Postico**
  - For everything else on all other platforms **DBeaver**

In Summary…

Name	Database Type	Relational vs Non-relational	Persistence	Pros	Cons	Best Usecase
PostgreSQL	Object-relational database management system	Relational	Yes	- ACID Compliant - Safe and stable - SQL Compliant - Atomic Transactions	- “Slower” than others - More difficult to manage	- My Goto database of choice - Never failed me once
MySQL	Open-source relational database management system	Relational	Yes	- Fast - Well Known - Been around for a long time	- Fails SQL Compliance - Tends to corrupt - Non-Atomic Transactions	- Legacy projects - I’d advise using it in production
SQLite	Relational database management system	Relational	Yes	- Fast - Isn’t client - Server - Can Run on clients computer	- Table locking doesn’t allow for multiple simultaneous transactions - Corrupts Easily	- Great for development environment - Do NOT USE IN PRODUCTION, EVER
Redis	Open-source in-memory database	Non-Relational	Semi	- Extremely fast key value store - Great for storing temporary data - Persists data to disk	- Data is persisted to disk every so many transactions - Can have dataloss	- Great for job queues - Great for storing sessions - Great for light workloads
Mongo	Document-oriented database	Non-Relational	Yes	- Great for storing binary files - Great for storing entire objects	- Super insecure out of the box - Non-Atomic transactions	- I’d use this to store documents but use a traditional RDMS for everything else

In Summary

- Use PSQL if you can
- Use MariaDB if you have to use a MySQL compliant database
- Backup all the time, when appropriate
- Always write restore commands to a text file and have a coworker read them to your out loud before ever running them, this will save your bacon

Questions?

Lecture 10 - Deployment

Deployment

- Since this Program is primarily a software development program I'm going to break into more of the DevOps approach here
- We are going to be talking about deployment not strictly in the sense of servers but in general as it is our industry
- We're moving rapidly towards treating our Servers and infrastructure as code, like any other, so we need to treat it as such

- Even network switches can now be programmed via software tools such as:
  - **Really Awesome New Cisco config Differ** (**RANCID**) for network switch configs
  - **Project OpenDaylight** implementing **software-defined networking** (**SDN**) for centralized network management

Linux Threads vs Processes

Linux Threads vs Processes

- When a program is launched it communicates with the **Operating System** (**OS**) for resource allocation
- Some of the things it asks for are **A register**, **A Program Counter**, and some **Memory for the Stack**

- Modern Operating Systems use a form of memory management called **Address space layout randomization** (**ASLR**) which creates a block of virtual memory that may or may not be associated to real memory

- This isolates each process into it’s own block of “**memory**” so no other processes cannot manipulate each other

- This also allows the operating to move memory onto the “**swap space**” to free up RAM for another process to use

- This independence of processes is valuable because the operating system tries its best to isolate processes so that a problem with one process doesn’t corrupt or cause havoc with another process

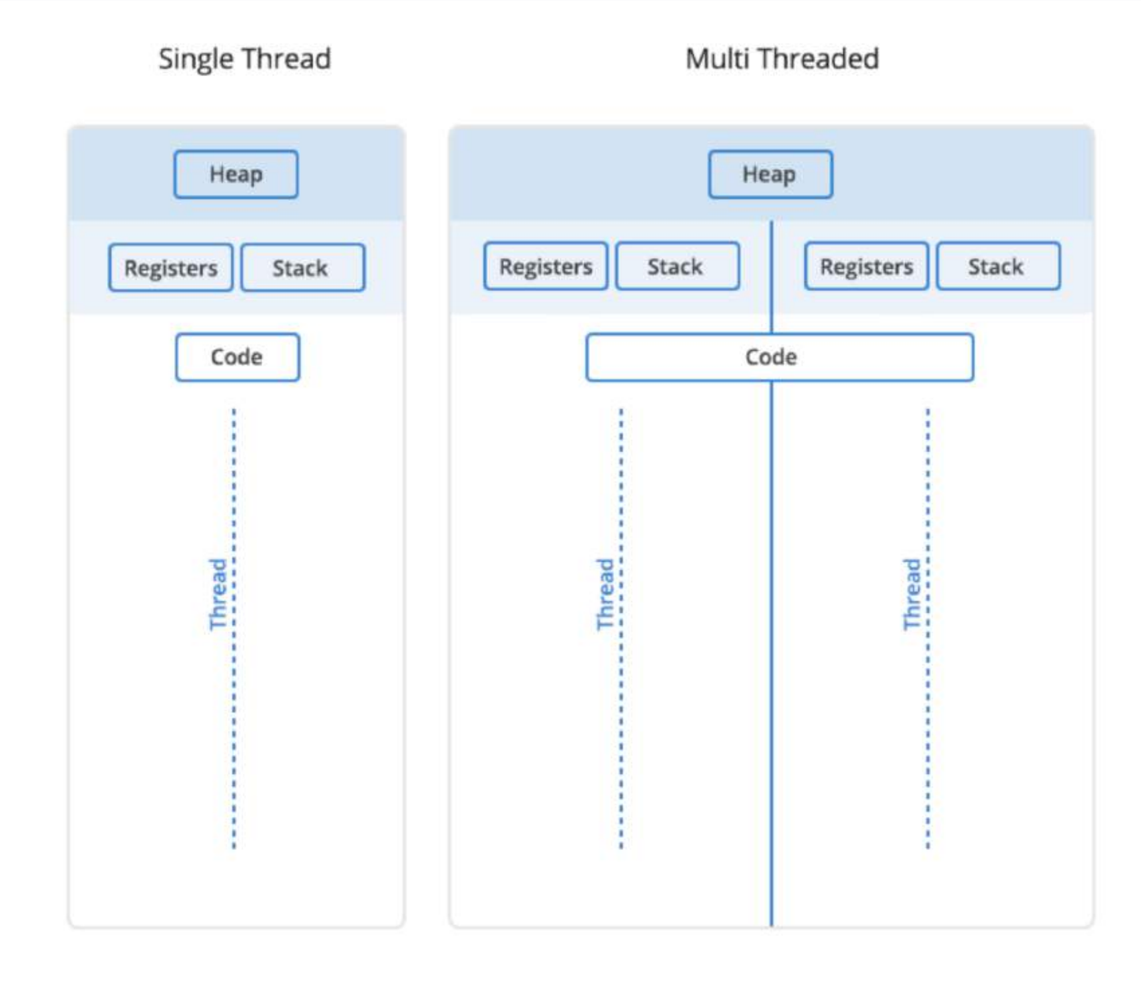
- Each process being spun up is a “**expensive**” task in terms of computer resources

- **Intercommunication of processes** is slow and cumbersome

Linux Threads

Linux Threads

- A **thread** is the unit of execution within a process.
- A **process** can have anywhere from just one thread to many threads
- When a process starts, it is **assigned memory and resources**. Each thread in the process shares that memory and resources.
- Since each thread shares the same memory, inter thread communication is easy



- Each **thread** shares the same heap so it **saves on some overhead**
- If a thread gets stuck though it **could cause the entire process to seize** and crash
- Most modern app servers are highly threaded

Processes vs. Threads — Advantages and Disadvantages

PROCESS	THREAD
Processes are heavyweight operations	Threads are lighter weight operations
Each process has its own memory space	Threads use the memory of the process they belong to
Inter-process communication is slow as processes have different memory addresses	Inter-thread communication can be faster than inter-process communication because threads of the same process share memory with the process they belong to
Context switching between processes is more expensive	Context switching between threads of the same process is less expensive
Processes don't share memory with other processes	Threads share memory with other threads of the same process

Linux Signals

Linux Signals

- **Signals** are a limited form of **inter-process communication** (**IPC**), typically used in Unix, Unix-like, and other POSIX-compliant operating systems

- A **signal** is an **asynchronous notification** sent to a process or to a specific thread within the same process in order to notify it of an event that occurred

- When a signal is sent, **the operating system interrupts the target process**' normal flow of execution to deliver the signal

- What it means in layman’s terms is that the OS can send signals to a process to notify it to do something
- This is commonly used to notify a process to **SIGSTOP**, be **SIGKILL**, or to **SIGHUP**
- In a **UNIX** environment the **SIGKILL** signal will terminal the process at the OS level, forcefully

- You can find a list here but the most important are:
  - **SIGSTOP** - Ask the application to stop
  - **SIGKILL** - Kill the application from the OS level
  - **SIGHUP** - Send a “hangup”, like hanging up the phone
  - **SIGUSR1** and **SIGUSR2** - Application defined signals

Rolling Restarts

Rolling Restarts

- The reason we're learning unix signals is that they're very important to modern applications
- As mentioned before modern app servers are **highly threaded**
- When we go to restart them we actually send them a **SIGHUP** or **SIGUSR1** signal which causes the process to do a **rolling-restart**

- Let’s take **PUMA** (Rails app server) as an example
  - This isn’t specific to PUMA since a lot of modern services behave this way
- When PUMA starts it has a main process and say 10 threads underneath it
- Each thread loads a copy of the application code into it

- When the main process is given a **SIGHUP** (**signal hangup**) it will start with the last thread and ask it to restart and load in new application code

- As each thread goes offline it will wait for it to properly start before restart the next thread

- That way the surviving threads will be able to handle incoming requests presenting 0 downtime to the customer

```
  1  [|||                                                                                                  2.0%]     Tasks: 44, 98 thr; 1 running
  2  [|||                                                                                                  2.0%]     Load average: 0.10 0.11 0.05
  Mem[||||||
```

Orchestration

Orchestration

- **Orchestration** is the **automated configuration**, **coordination**, and **management** of computer systems and software

- The main difference between a workflow "**automation**" and an "**orchestration**" (in the context of cloud computing) is that workflows are processed and completed as processes within a single domain for automation purposes, whereas orchestration includes a workflow and provides a directed action towards larger goals and objectives

- To sum it all up it means that Orchestration is a tool to allow you to automate large tasks to achieve a goal

- For example it can automate a whole deployment of an application and all servers it depends on without user (Human) interaction

- Tech companies are pushing towards this because it reduces the risk of human error and is repeatable

- The way it works is that there is a Service / Server that is statically created and never allowed to go down
- From this Server it will trigger the creation of VMs / Containers / Services etc… and keep record of where they are
- When the Server is created it calls back home to the Orchestration server and updates its record with all of its IP, Hardware Configuration, OS, etc… so we can call upon it to do actions
- It will also register these servers in DNS and / or Load Balancers so other servers can send data to them

- For example there is something called a **GitlabRunner** which is a **CI/CD** service gitlab provides

- You would install this runner on a VM separate from the Gitlab VM and have it constantly communicate back to the main Gitlab server

- When a trigger is met, someone pushed commits to the staging branch, then it will execute the **gitlab-ci.yml** file in that project

- It will then:
  - Create a VM
  - Install docker on it
  - Run the build stage
  - Report the results
  - Then destroy the VM shortly after to save money

- This allows for many jobs to be processed in parallel while not paying for idle servers waiting to execute something

- Netflix for example will spin up thousands of Application Servers to handle increased traffic in the evening but then kill them in the middle of the night to save thousands of dollars each month

- It allows them to meet demand and satisfy their customers but not have a lot of overhead costs with idle servers

Automating Deployments

Automating Deployments

- Automation applies to all types of Development / IT regardless of what you think
  - Mobile: BuildBuddy, BitRise, FastLane, etc…
    - Can build iOS and Android projects
    - Run tests for you, and if they fail stop the build
    - Take screenshots and post the newest onto the app store page for you
    - Deploy to app stores for you

- For Web Projects: Jenkins, GitlabCI, Github Actions, TravisCI, etc…
  - Can build your projects
    - Run your tests
    - Lint your code (code smells)
    - Package your code into Docker Containers
    - Deploy said containers
    - Notify people that a version went live

- We call these tools **Continuous Integration** and **Continuous Delivery** or Deployment (**CI/CD**)

- **Continuous Integration**: The practice of merging in working code onto a shared machine and executing a planed sequence against it

- **Continuous Delivery**: is a development approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time

- These tools help you automate workflow saving you hours of time and a ton of heartache

CI/CD

CI/CD

- One of the old problems in the past is software would be released once a quarter to once every few years
- In-between when the software is being developed and being launched there's a whole lot that has changed
- During development a developer would continuously update the application as they develop it (duh) but they may have introduced a breaking change further down the line which will cause the whole project to implode during a fresh install

- This may sound like a rare problem but it’s actually the opposite, extremely common
- So the way we avoid it is to continuously “rebuild” your project from the last released state and from “fresh” to make sure these changes don’t reach our customers
- A CI/CD tool **listens** in **Git on a branch or a tag** and will execute its instructions when a condition is met, like a **merge request** (**MR**) or a **pull request** (**PR**)

- These actions are normally dictated in a YML file located in the root of the project
- Typically a pipeline is a set of actions done in a sequence
- These actions could be first checking the codebase for syntax errors, then building the application, then the next phase could be testing the built application, then finally deploy the application

- If any part of that chain fails then it will cancel the pipeline completely to make sure no bad code goes out

- Since each build will be separate and from fresh you can see how the application will behave when you go live

- And since you can trust that the pipeline is working correctly, your code is tested, no obvious errors found you can then Go Live to the world multiple times a day without any human interaction

Merge branch 'version_1.1.4' into 'staging'

```
Version 1.1.4

See merge request !303
```

12 jobs for `staging` in 6 minutes and 49 seconds (queued for 2 seconds)

latest

a33e53b4

12 related merge requests: !271 Staging, !255 Staging, !249 Staging, !243 Staging, !240 Staging, !233 Staging, !230 Staging, !223 Staging, !220 Staging, !216 Staging, !213 Staging, !197 Staging

Pipeline Jobs 12 Failed Jobs 2

Audit	Build	Lint	Test	Deploy
api_audit	api_build	api_lint	api_test	api_deploy_stag...
frontend_audit_...	frontend_build	frontend_lint	frontend_test	frontend_deplo...
frontend_audit_...	frontend_build_...			

removed mailer from scheduler

11 jobs for `version_1.1.4` in 4 minutes and 50 seconds (queued for 1 second)

62bf82a1

2 related merge requests: !304 Version 1.1.4, !303 Version 1.1.4

Pipeline Jobs 11 Failed Jobs 3

Audit	Build	Lint	Test
api_audit	api_build	api_lint	api_test
frontend_audit_...	frontend_build	frontend_lint	frontend_test
frontend_audit_...	frontend_build_...		

Updating GEM and docker. Getting everything to play nice

3 jobs for `version_1.0.0` in 6 minutes and 46 seconds (queued for 2 seconds)

`0aae64b6`

No related merge requests found.

Pipeline Jobs 3 Failed Jobs 2

Security_check	Static_code_analysis	Test
ruby_advisory_d...	rubocop	rspec

Rubocop hurts

3 jobs for `version_1.0.0` in 13 minutes and 50 seconds (queued for 3 seconds)

`11bdfc8f`

No related merge requests found.

Pipeline Jobs 3 Failed Jobs 2

Audit

rspec

rubocop

ruby_advisory_d...

When Not to Automate a Deployment

When Not to Automate a Deployment

- Sometimes business rules dictate that you can't deploy every few hours, few days, or few weeks
- In a lot of ways we really can't argue those business rules; sometimes it's just out of our hands
- I personally feel like production deployments should be a matter of intention and not an accident, for that reason I believe that for production only the final step in deploying software requires human input (by clicking a play button)

- There are two hard things in computer science:
  - 1. cache invalidation,
  - 2. naming things,
  - 3. and off-by-one errors.
- That was said by Jeff Atwood, cofounder of Stack Exchange

Idempotence

Idempotence

- **Idempotence** is the property of certain operations in mathematics and **computer science** whereby they can be **applied multiple times without changing the result beyond the initial application**

- What this means is that if an operation is performed multiple times, it will only have an effect the first time but none on sequential runs

- On/Off buttons of a train's destination sign control panel. Pressing the On button (green) is an idempotent operation, since it has the same effect whether done once or multiple times. Likewise, pressing Off is idempotent.

src: https://en.wikipedia.org/wiki/Idempotence#:~:text=Idempotence%20(UK%3A%20%2F%CB%8C%C9%AA,result%20beyond%20the%20initial%20application.

Orchestration Services

WORKED FINE IN DEV
OPS PROBLEM
NOW
quickmeme.com

Puppet

Puppet

- **Puppet** is an **open-core** software configuration management tool
- What that means is puppet will configure servers and services on those servers using idempotence aware configuration files
- This is an alternative to using Docker but is typically used to set up servers with the base Docker package so you can deploy docker images to that server
- This is typically used by **System Administrators** (**OPS**) and can be used by **Developers** (**Devs**)

Chef

Chef

- **Chef** is a configuration management tool, similar to Puppet
- It also configures servers and services on those servers using **idempotence** aware configuration files using what's called a Recipe
- Is one of the industry's most notable **Infrastructure as Code** (**IAC**) tools
- What that means is it aims to treat all servers and services as code instead of hardware

Ansible

Ansible

- **Ansible** is an open-source **software provisioning**, **configuration management**, and **application-deployment tool**

- Originally developed by Ansible Inc founded in 2013 but acquired by Red Hat in October 2015

- Unlike Chef and Puppet, Ansible runs in a simple **agent less architecture** meaning you don't need to install / configure a listening agent software on the controlled hosts

- Ansible connects to the controlled host using **SSH** or **Windows Remote Management** to do it's tasks or **Playbooks** as they call it

- For the duration of an orchestration task, a process running the module communicates with the controlling machine with a JSON-based protocol via its standard input and output

- The configuration files for ansible are held as simple ASCII files written in **YAML** or **INI** format

- A pretty standard way to lay out an ansible management node is:
  - **ansible/**
  - **ansible/hosts.yml**
    - This is a file that has all the target host groups and associated addresses (IP or DNS)
  - **ansible/playbooks/**
  - **ansible/playbooks/deb_fail2ban.yml**

Ansible Inventory

- Hosts are grouped under the default group of **all** with subgroups underneath
- A host can be in multiple subgroups, which then becomes a child of the **all** group
- Each subgroup can then have subgroup children as well, ad nauseam
- When we execute commands we will be requiring the hosts file and telling it which group to execute against

```yaml
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

Ansible Commands

- Let’s take the following command as an example:
  - **$ ansible -i ./hosts.yml all -m ping**
- **-i** specifies the inventory hosts.yml file to load
- **all** is the target group, which is all
- **-m** is the module name, which this is a predefined one in ansible called ping

Ansible Playbook Commands

- Let's take a look at another command for an example:
  - **$ ansible-playbook -i ./hosts.yml playbooks/deb_patching.yml --ask-become-pass**
- **-i** specifies the inventory hosts.yml file to load
- **playbooks/deb_patching.yml** is the playbook we wish to run, which already has the target hosts defined inside
- **--ask-become-pass** is instructing ansible to ask for the password to elevate privileges through sudo on the target hosts

Ansible Note

- One thing to note is that since this is uses basic ASCII files you need to have all your files relative to where you are excepting

- What I mean is that you have an **ansible folder** and inside you have all the files you need to execute your playbooks

- One exception is that you can have your **hosts.yml** file at **/etc/ansible/hosts** and have it autoload in your commands, but I prefer manually specifying

Cargo Culting

Cargo Culting

- **Cargo Culting** is when a software development organization attempts to emulate more successful development houses, by slavishly following a software development process without understanding the reasoning behind it

- The term cargo cult as an idiom, originally referred to aboriginal religions that grew up in the South Pacific after World War II

- The practices of these groups centred on building elaborate mock-ups of airplanes and military landing strips in the hope of summoning the god-like airplanes that had brought marvellous cargo during the war

- During this lecture we learned all about interesting technologies and ways to approach reliable deploying and automation

- They all sound great on paper but until you have a business reason to implement the approach, it will just cost you time and money with added complexity that could easily overwhelm you

- All things should be approached skeptical until the need actually arises

    - For example: if you were working with a large organization or a large team and you need to constrain other developers from making ignorant changes to the overall project infrastructure

In Summary…

In Summary

- There are a great many powerful tools that allow you to automagically scale a server from 1 to 1000 servers in minutes, but they can be complicated to set up the correct infrastructure to support this
- Depending on your actual needs, this may be overkill and using a simpler but still effective tool (ansible) would probably be your best bet
- Threads, processes, and signals are an intrinsic part of Unix and understanding them can let you roll out applications with 0 downtime

Links

- Ansible Links:
  - https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html
  - https://docs.ansible.com/ansible/latest/cli/ansible.html
- Other links
  - https://www.chef.io/
  - https://puppet.com/

Questions?

Lecture 11 - Autoscaling

Scaling and Autoscaling

- Today we are going to be talking about scaling, scaling problems, and possible tools and solutions to solve the problem
- We are going to be talking in the context of you, a developer, launching a new and fun startup **blockbin.io**
- This will give you the reason and rational why we are scaling the way we are

Phase 1 - Initial Launch and a few users

The beginning of your startup

blockbin.io

- Web Service, Application Service, and Database on a single Server

What’s going to Fail Next?

Too many requests

Why did it fail?

- Each request takes up a thread slot for the duration of the request
  - More on this later
- Each request hits the database to retrieve data for the user
  - It takes work and memory to get data from the database
- Each request takes up **CPU time** and **memory**
  - Not much mind you, but it all adds up
- If you exceed memory (and swap) the server **SEGFAULTS** (hard crash) and goes offline
  - If server is offline, who can serve the customers requests?

Phase 2 - 10, 000 Users

How do we handle the load?

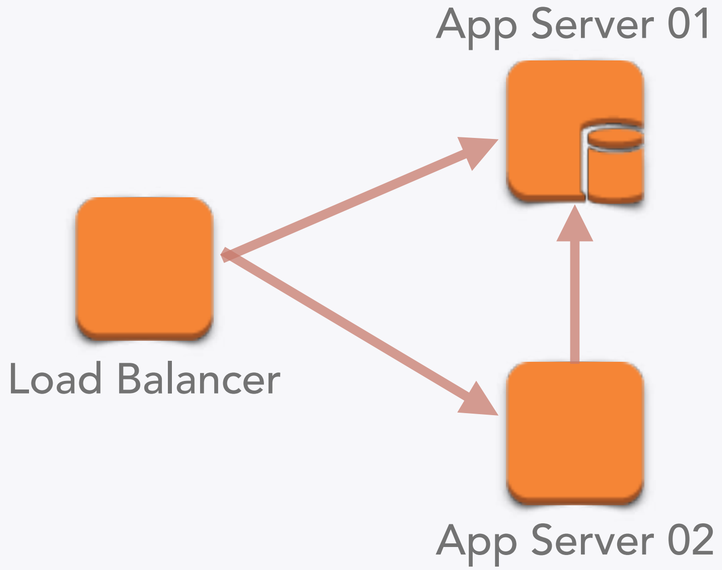
Load Balancers

Load Balancers

- **Load balancers** are a piece of hardware or software that sits in front of your application servers to handle incoming traffic and route accordingly

- Every few hundred milliseconds the load balancer will send a **http check** against the server to see if it responds

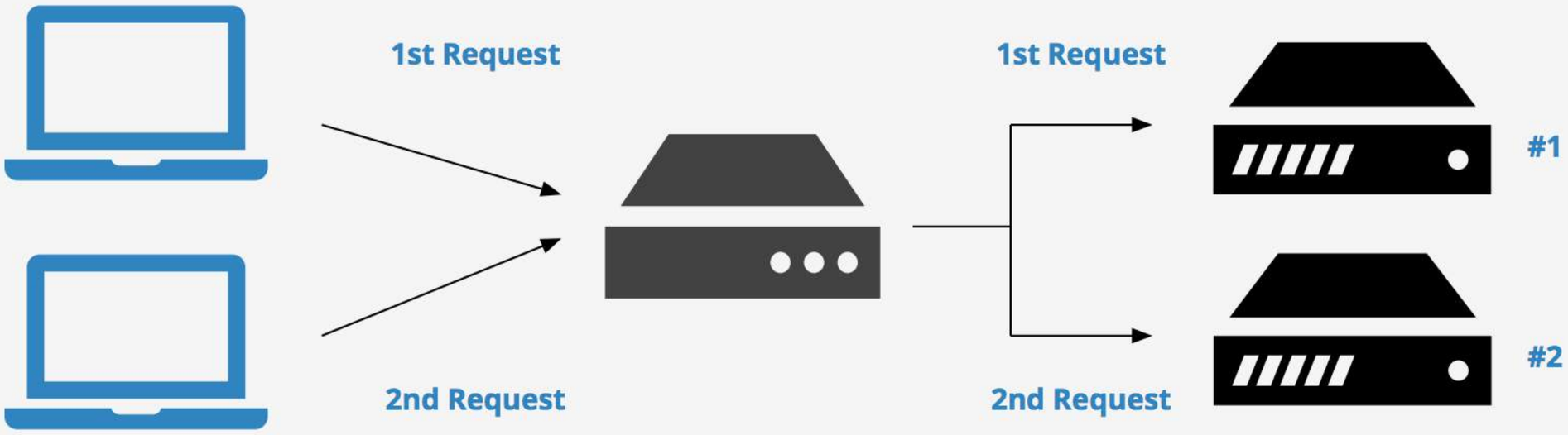
- If it **doesn't respond in a certain amount of time** it will mark the server as dead and remove it from the pool of servers to balance against



- If it comes back online then it gets re-added to the **distribution pool**
- There are different strategies to load balancers such as:
  - **Round Robin**
  - **Least Connection**
  - **Preferred host**

Round Robin Load Balancer

- For now we are going to talk about **Round Robin load balancing**
- The basic idea behind it is that each server, that's online, will get a **request sequentially** to how the requests come in
- For example there are 2 application servers in a load balancer distribution pool:
  - Client 1 sends request 1 so app server 1 handles the request
  - Client 2 sends request 2 so app server 2 handles the request
  - Client 3 sends request 3 so app server 1 handles the request



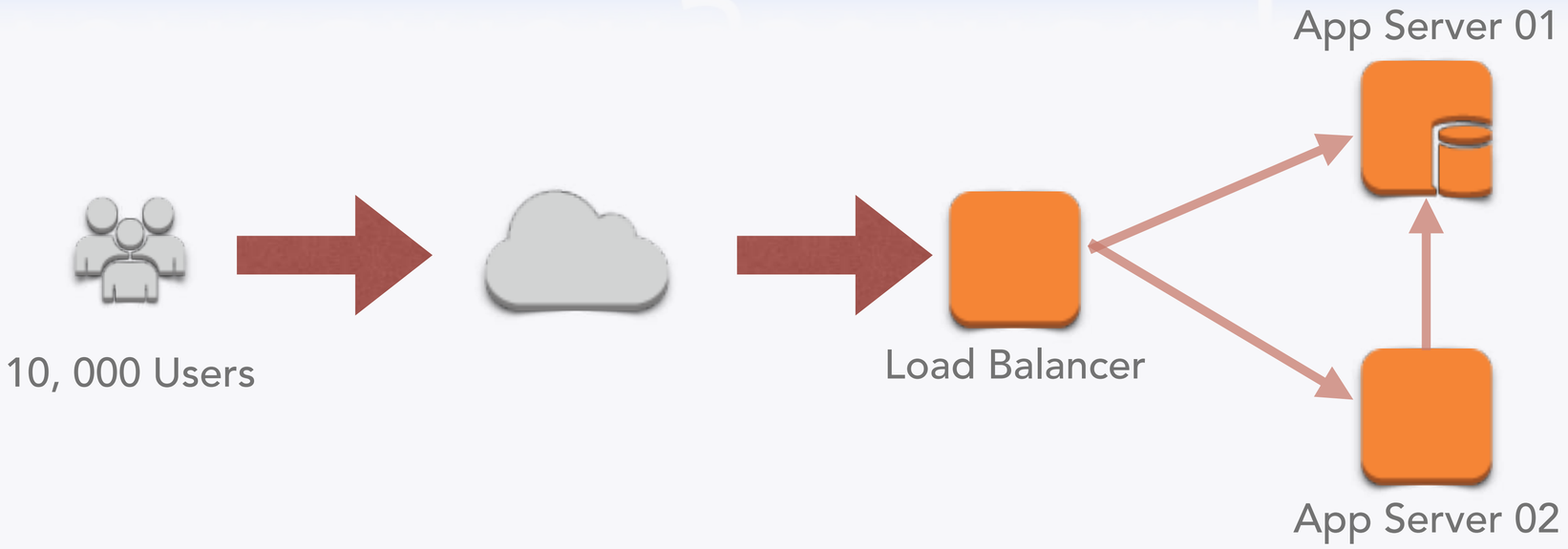
Load Balancing

https://www.keycdn.com/support/load-balancing

Some Architecture Concerns

- Since files exist only on the server that got the request you can error out if you attempt to manipulate the file from another server
  - All files should be stored on some kind of network device that all servers can see like S3 or a shared directory
- With more servers comes a greater dependance on using DNS to intercommunicate instead of hardcoding IP Addresses
  - Remember DNS can be changed external to your application so you don’t need to push an update to move something

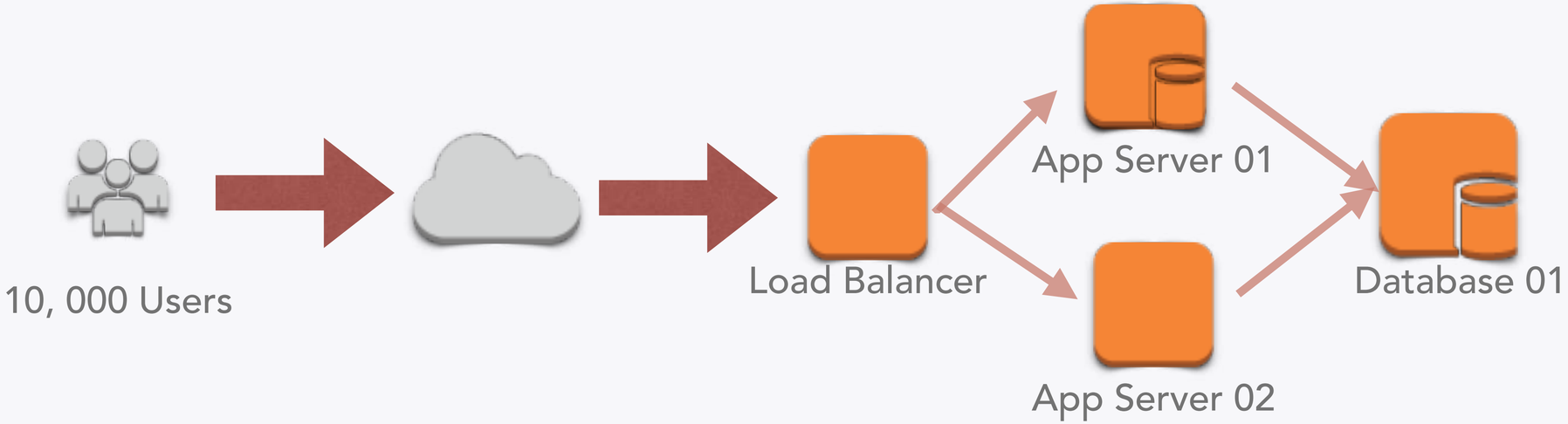
Split Work amongst multiple servers



Split the incoming requests to each server

What happens if you need to take down the app server with the database on it for maintenance?

Separate Database From App Servers



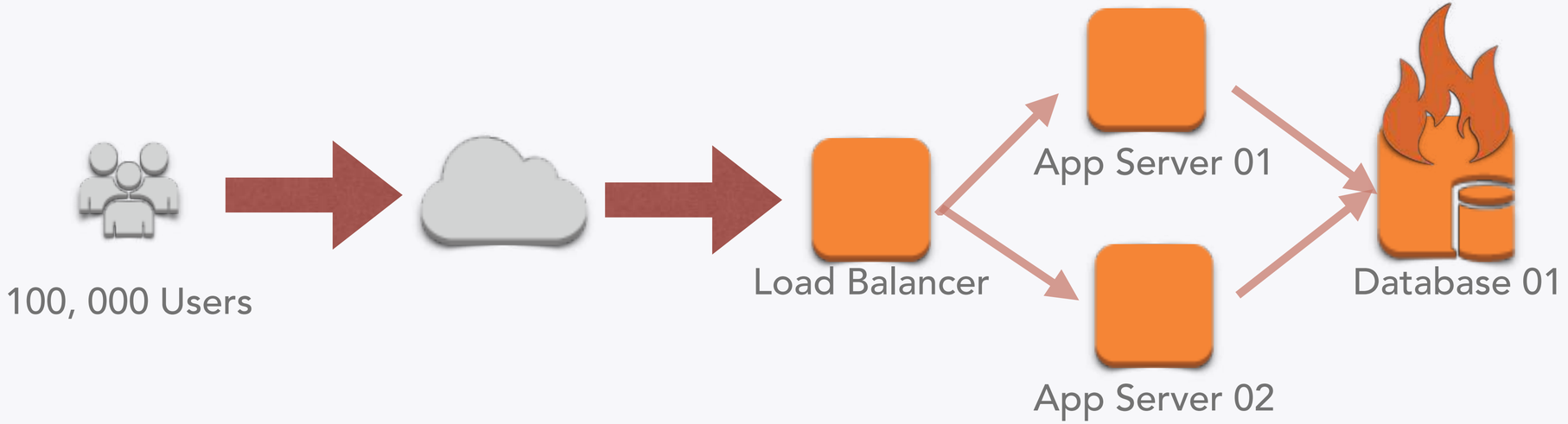
Why separate the database from the servers?

Separation of concerns

- Each server should have a single use or “**concern**”
- The more **concerns** on a single server the more likelihood that a **single failure can bring down your entire application**
- Each concern can be **scaled independently** instead for when the need arises

Phase 3 - 100, 000 Users

What’s going to Fail Next?



Too much traffic for the Database to handle

Common Causes of Database Breakdowns

- **Long Database Queries**
- **Improperly indexing** your tables
- **Too much indexing** of your tables
  - Sometimes too much is just as bad as not enough
  - Remember Indexes increase the amount of data that is written to the server at the expensive of faster lookup times in certain scenarios

Database 01

How do you solve it?

Clustering

What is Clustering?

- **Clustering** is, perhaps, the least understood technology in IT.

- The equation of "Cluster = Performance" is one of the **biggest misconceptions**. It's true that clustering is often critical in high performance applications. That said though, it's also one of the hardest goals to achieve unless an application has been specifically designed to take advantage of clustering.

- There are two major problems that clustering solves; Reliability and Scalability.

- At its most basic meaning, a "**Cluster**" is nothing more than **two or more computers working together** to accomplish a common task. Beyond that, a "cluster" can be anything the implementer designs it to be.

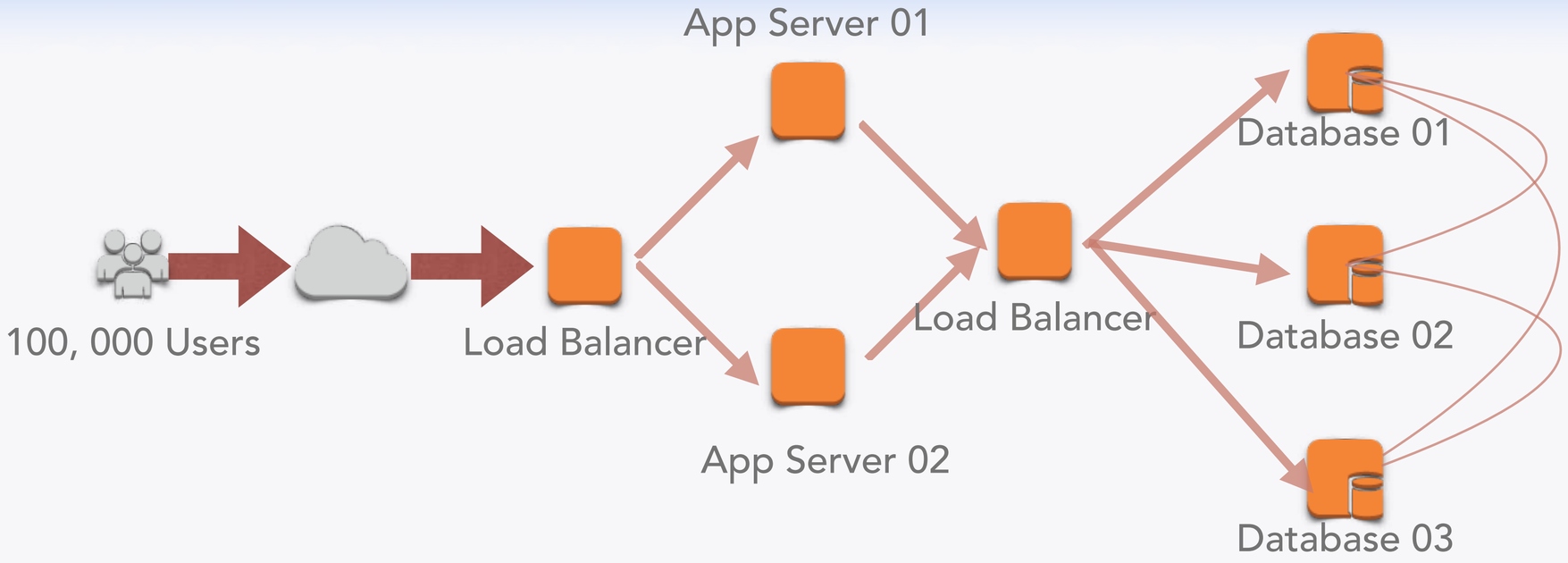
- In many cases, this means customizing a given application to (effectively) spread out the load among the cluster's nodes in some way best suited to that given application. When an application is not cluster-aware, it may involve clustering a component of the system below the application, such as a clustered Virtual Server on top of a DRBD partition using LVM.

- For these reasons, clusters are very rarely consistent with one another.

- Some terms you should know:
  - A "**node**" is simply a computer participating in a cluster.
  - In a cluster with shared storage, a "**fence**" is a method of disabling or disconnecting a defective node from the cluster and its shared resources.
  - In clusters with more than three nodes, "**quorum**" is achieved when more than half of the maximum nodes can communicate with each other. A cluster, or cluster segment, can only start up when it has quorum. Any nodes in a segment without quorum will be fenced.

- There are two main types of clusters:
  - **High Availability** (HA)
    - What we are talking about today
  - **High Performance** (HPC)
    - Another term for this is “**Super Computers**”

Let’s Cluster The Database



A cluster of 3, minimum

Database Clustering

Database Clustering

- **Database clustering** works by taking incoming database writes and copying them amongst multiple machines in case one fails

- The idea is that if the application is critical enough, say banking, they **can't allow a single server to take down all operations**

- One unique property of database clustering is that when you go to **write** data to the database, it will first distribute the **write** to the entire cluster, and when **all nodes confirm it's written it returns a success to the application**

Multi-Master Database Clustering

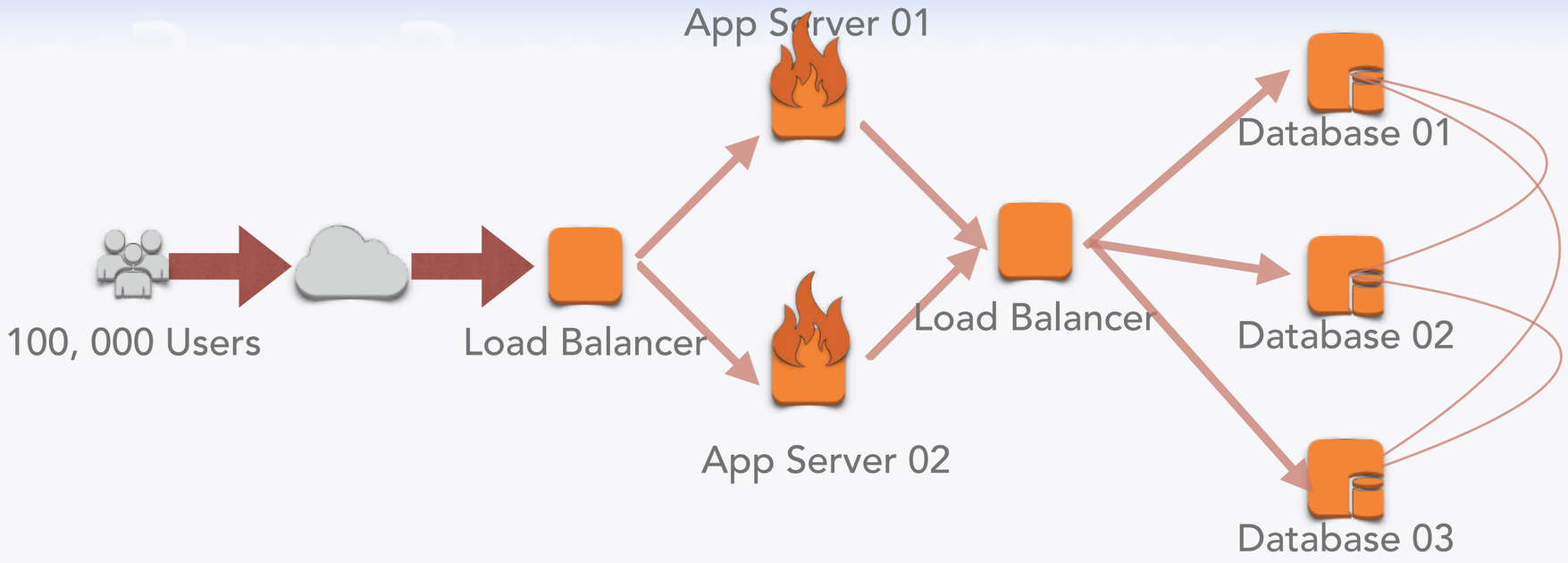
- Multi-master database clustering is when all **nodes** of a DB cluster can read and write to the cluster at any time
- This could be helpful if you wanted to distribute the load amongst many machines **BUT** it has a massive downside
- Since a write is only successful if all **Nodes** can successfully **write** it creates a delay when writing to the cluster
- If each **Node** is trying to **write** to the cluster at the same time they all have to wait for all of them to finish which causes delays which could cause the application to seize up waiting

Single master with hot spares

- A more efficient and, in my opinion, more sane way to achieve a database cluster is to have a **single master**
- This master will report to the load balancer it's available while the other ones won't
- If that master fails, then another spare / slave will **promote its self to master** and take its place
- This way there is virtually no downtime yet you don't have the drawbacks of multi-master

Phase 4 - 200, 000 Users

What’s going to Fail Next?

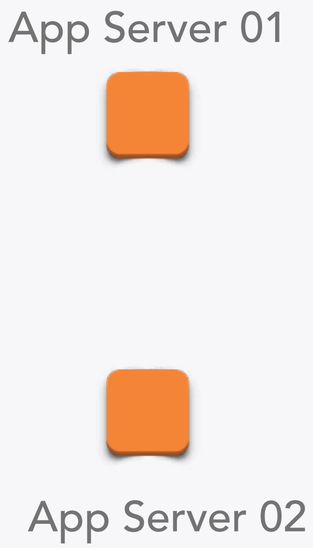


Application processing time is causing disruptions

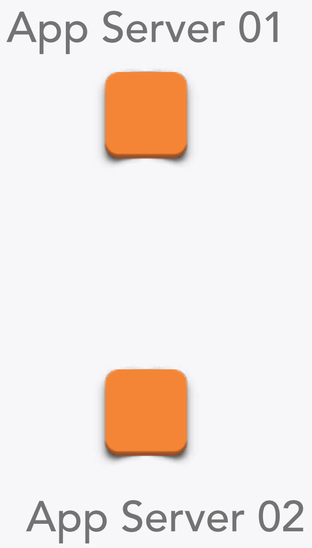
Application Threads

Application Threading

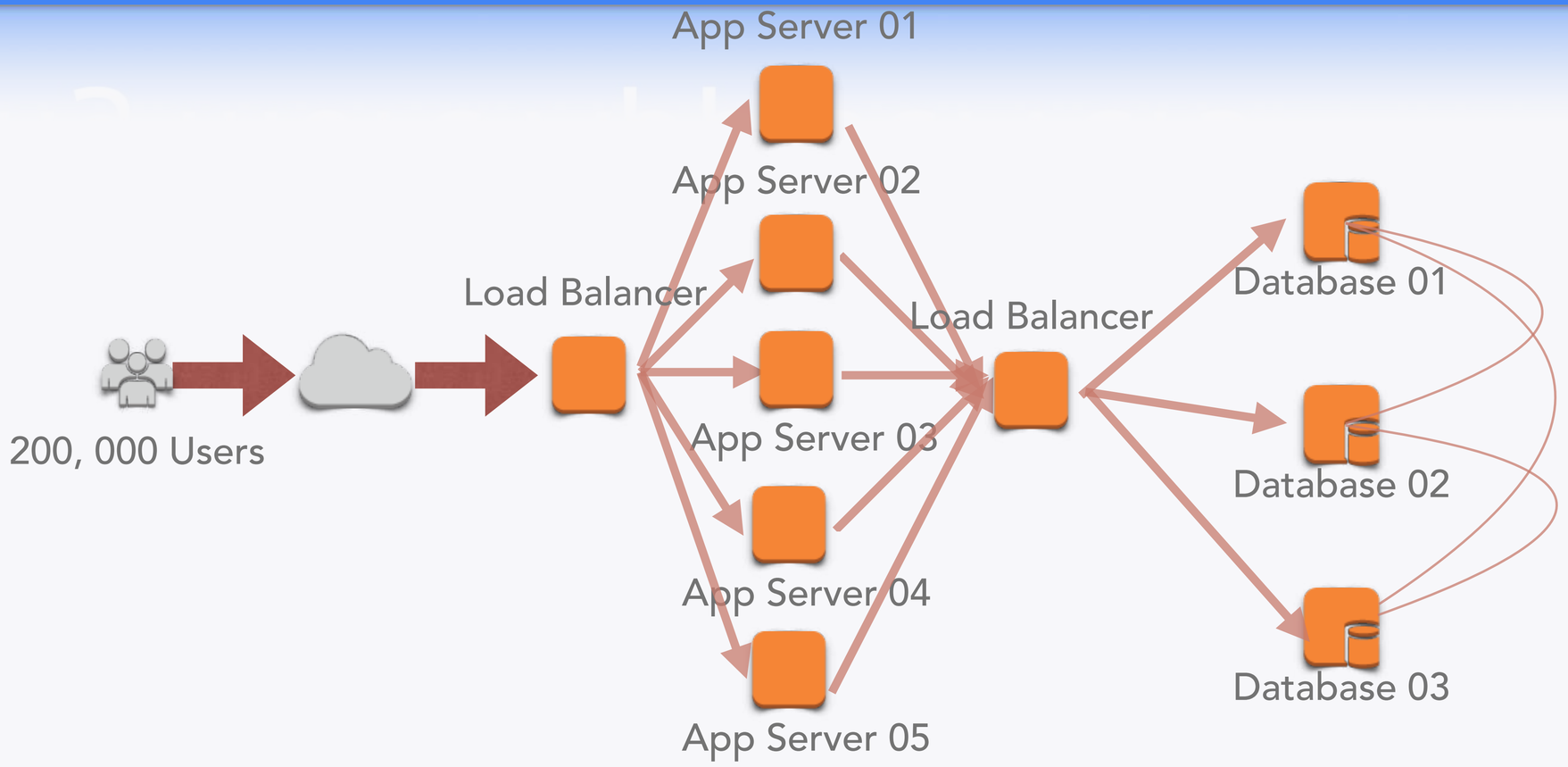
- An application can only handle 1 connection at a time
- These connections are typically short lived (100ms) but what happens if more than 1 are received at the same time?
- The second client would have to wait for their turn or else time out
  - Both of these make for bad user experience



- The way we typically overcome this hurdle is to have many copies of the application running in parallel
- A single process loads the app into memory, and under that process many threads are spawned listening for connections
- Typically we use 10 - 25 threads of an application in production, depending on how intensive the application is
- We can also increase server count which will then increase incoming connection count
  - 5 Servers * 10 threads = 50 simultaneous requests
  - 50 Servers * 25 threads = 1,250 simultaneous requests

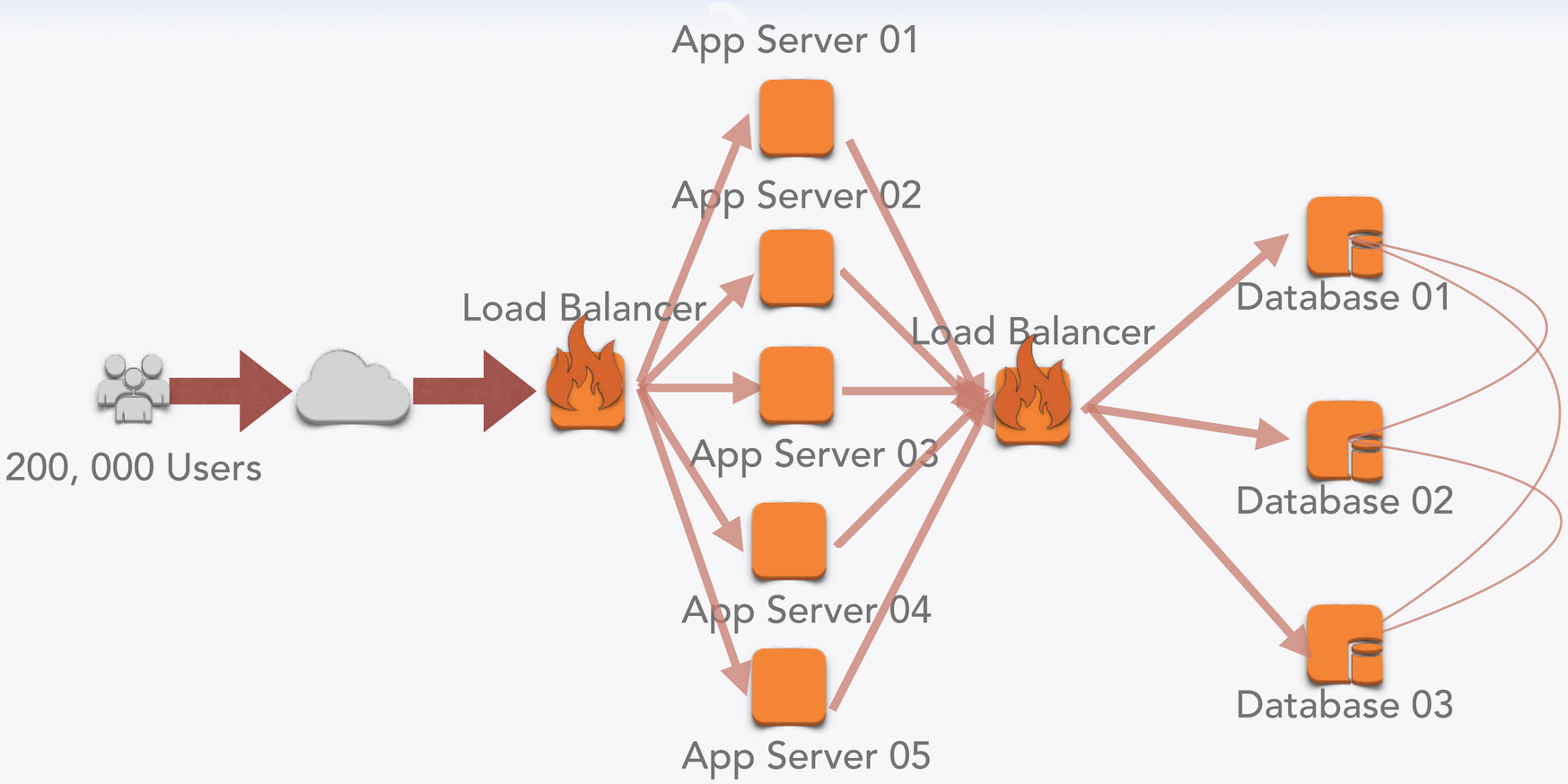


Adding more App Servers



Increasing the app cluster to handle requests

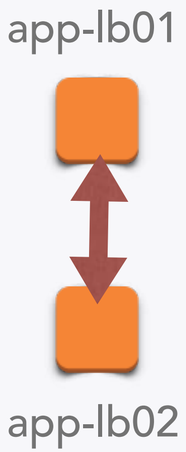
What’s most likely to fail next?



Virtual IP Addresses

Virtual IP Addresses

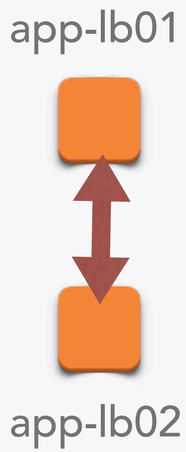
- There is a way to set up 2 servers to share an IP address
- An application called **KeepAliveD** allows you to configure two or more servers to work in a mini cluster
- This **KeepAliveD** cluster creates a IP address and on the master server broadcasts that to the network
- Once every few milliseconds all members intercommunicate to make sure they're still online



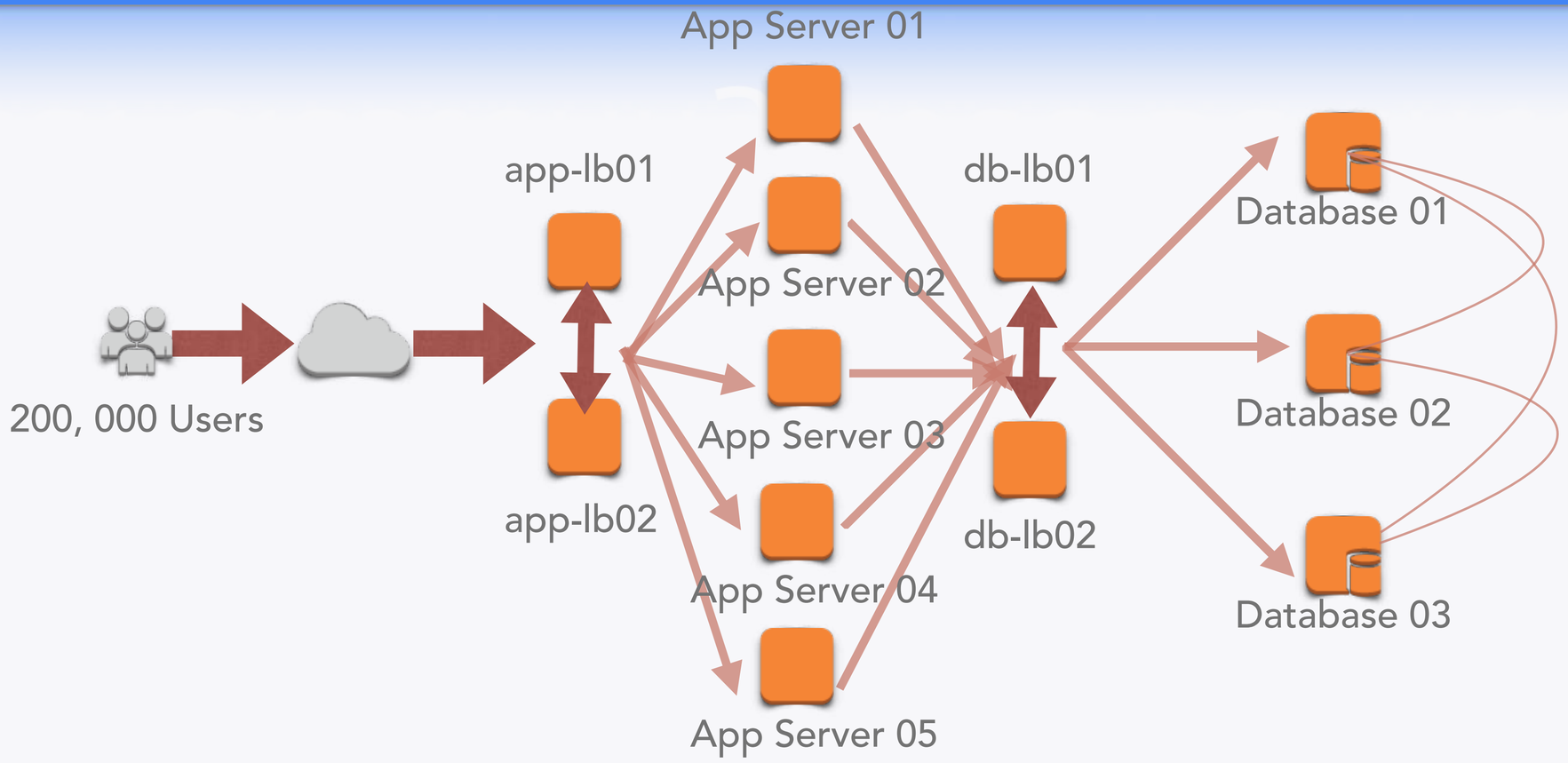
- If a "***slave***" member detects that the master is offline it will promote itself to Master and start broadcasting that the IP address has moved to the network

- This allows load balancers to have "**Hot**" backups waiting to take over if the primary / master one goes down

- This is one of the best strategies since load balancers sit in front of applications

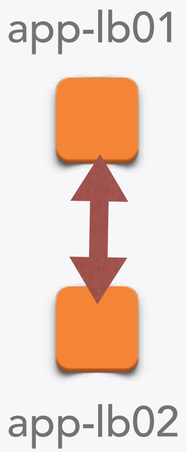


Virtual Clustering Load Balancers



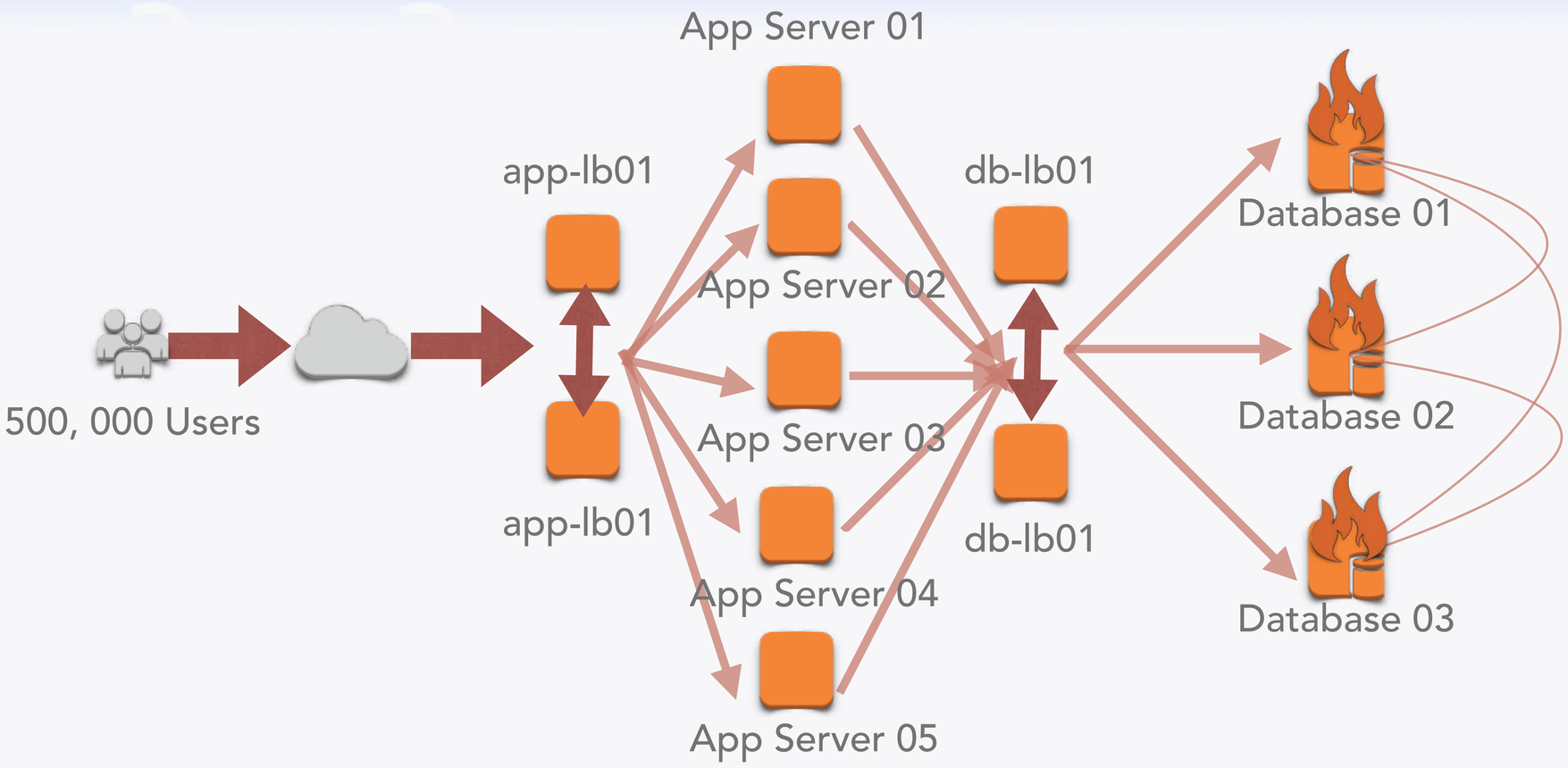
Known issues with load balancers

- Even though we have multiple load balancers working as hot backups it may not solve the scaling issue
- Sadly the only way to really scale the load balancers is to get bigger, better hardware or use some kind of unicasting IP method
- IPv6 allows a single IP address to be broadcasted so multiple Load Balancers in theory could work together and get traffic routed based on who gets it first but this can lead to **session sickness**



Phase 5 - 500, 000 Users +

What’s going to Fail Next?



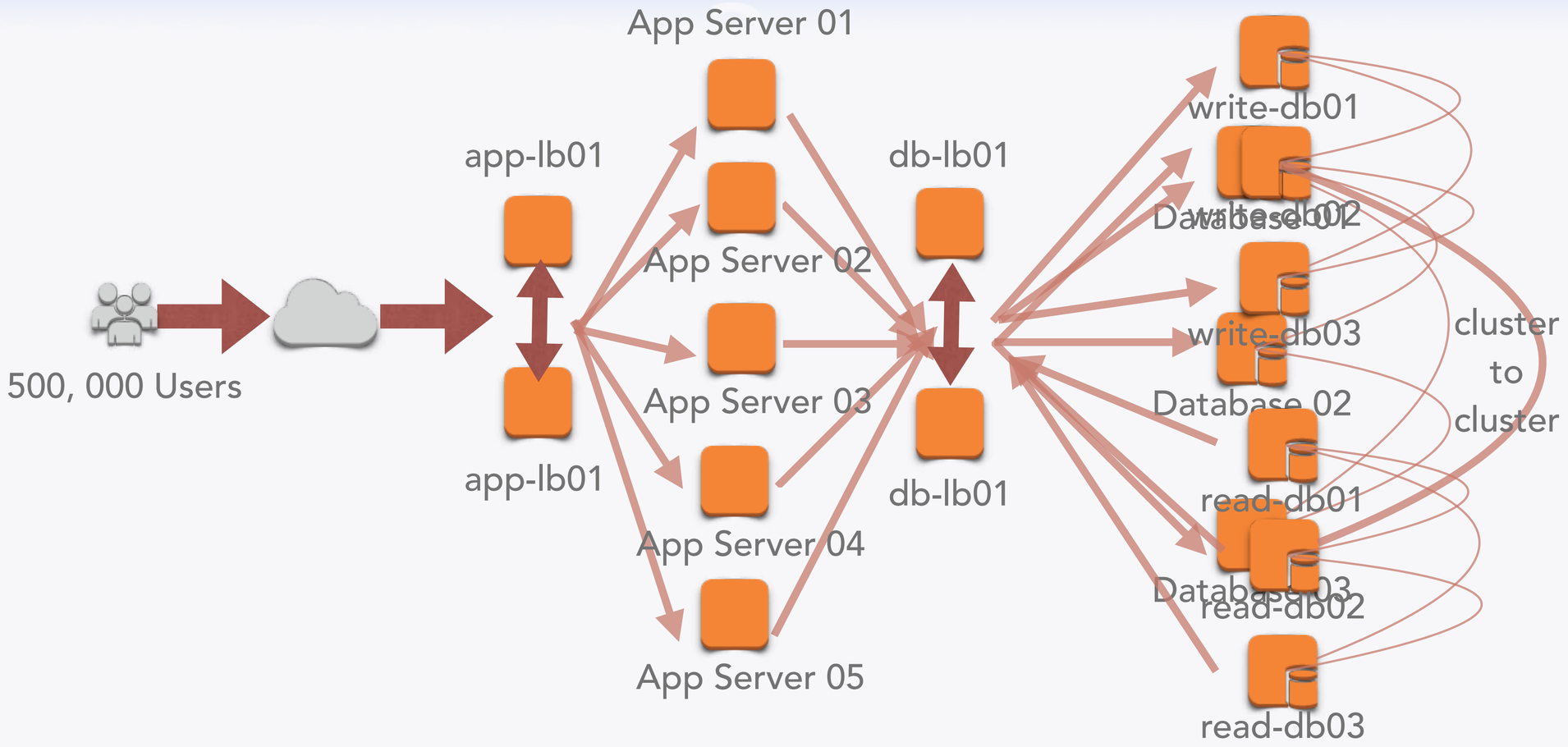
Database Sharding

Database Sharding

- At a certain size of application you will need to start "**sharding**" your database
- This means splitting database clusters into multiple clusters that server a single purpose
- One cluster will be a **write only** cluster. This will only handle incoming database writes.
- Once the write is committed to the cluster, the **write** cluster will replicate the data to the **read** clusters

- Since read clusters can be read from in any order it can scale much better than writes
- This allows many times more application servers to be served without taking down the entire application or waiting for writes to be committed
- That being said it does introduce some caveats….

Database Sharding



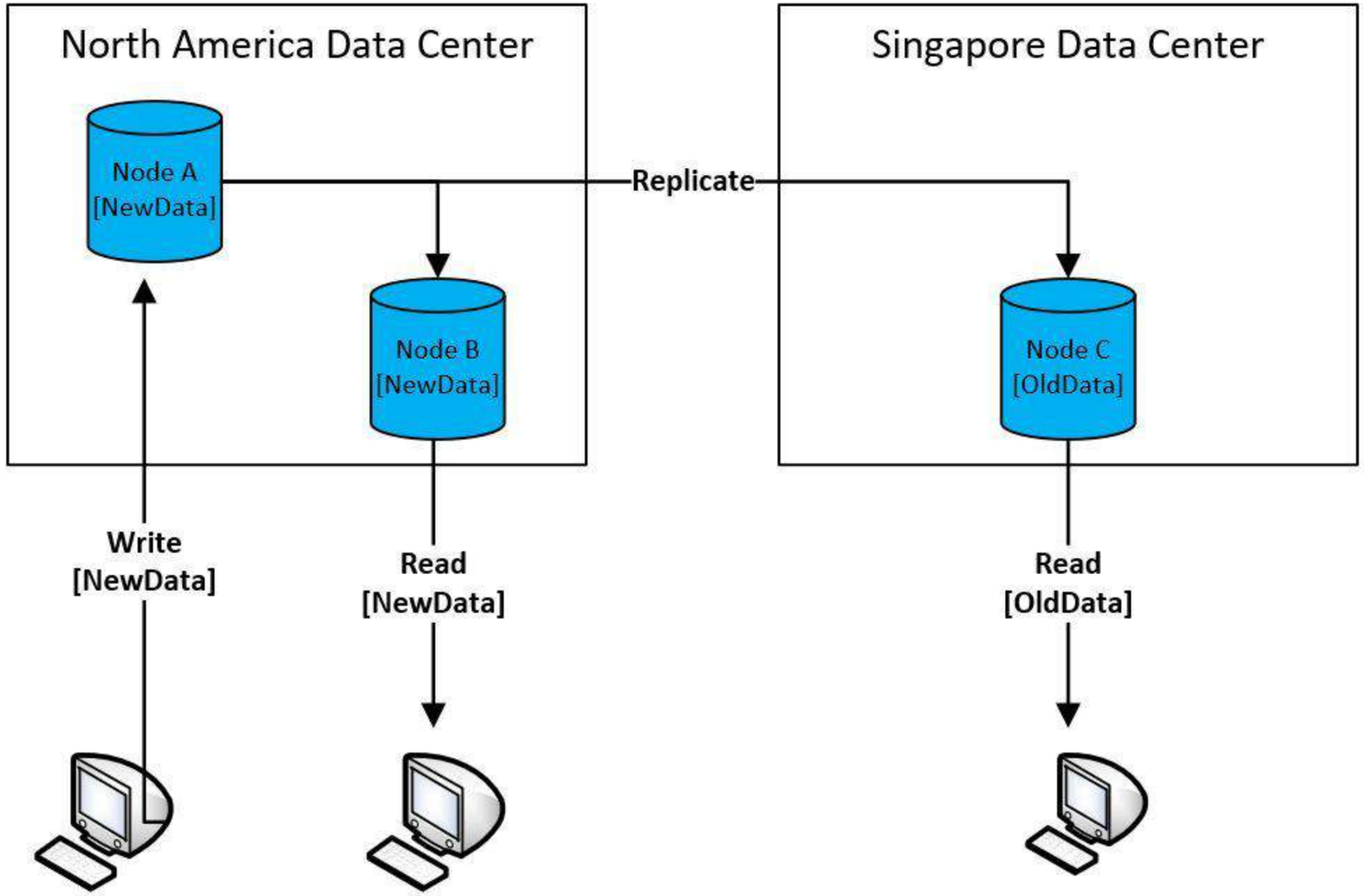
Eventual consistency

Eventual Consistency

- **Eventual consistency** is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

- **Eventual consistency**, also called **optimistic replication**, is widely deployed in distributed systems, and has origins in early mobile computing projects.

- For example, Netflix will update their database with new content and eventually that content will propagate to other regions of the world.



https://www.acodersjourney.com/eventual-consistency/

Microservices vs Monolithic Services

Monolithic Services

- A **Monolithic Service** or **Monolithic Application** is an application that contains many different parts but in a single codebase

- It is a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.

- Currently, every project you've done here at the college has been a **Monolithic Service**

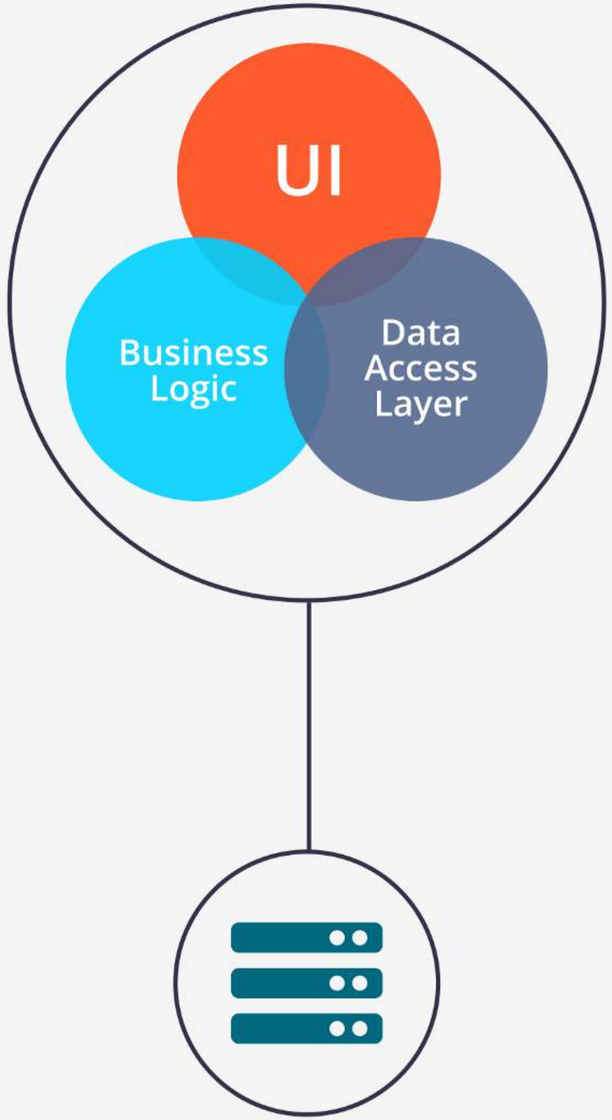
- These aren't inherently bad but they get messy when the application grows in size and scope

Microservices

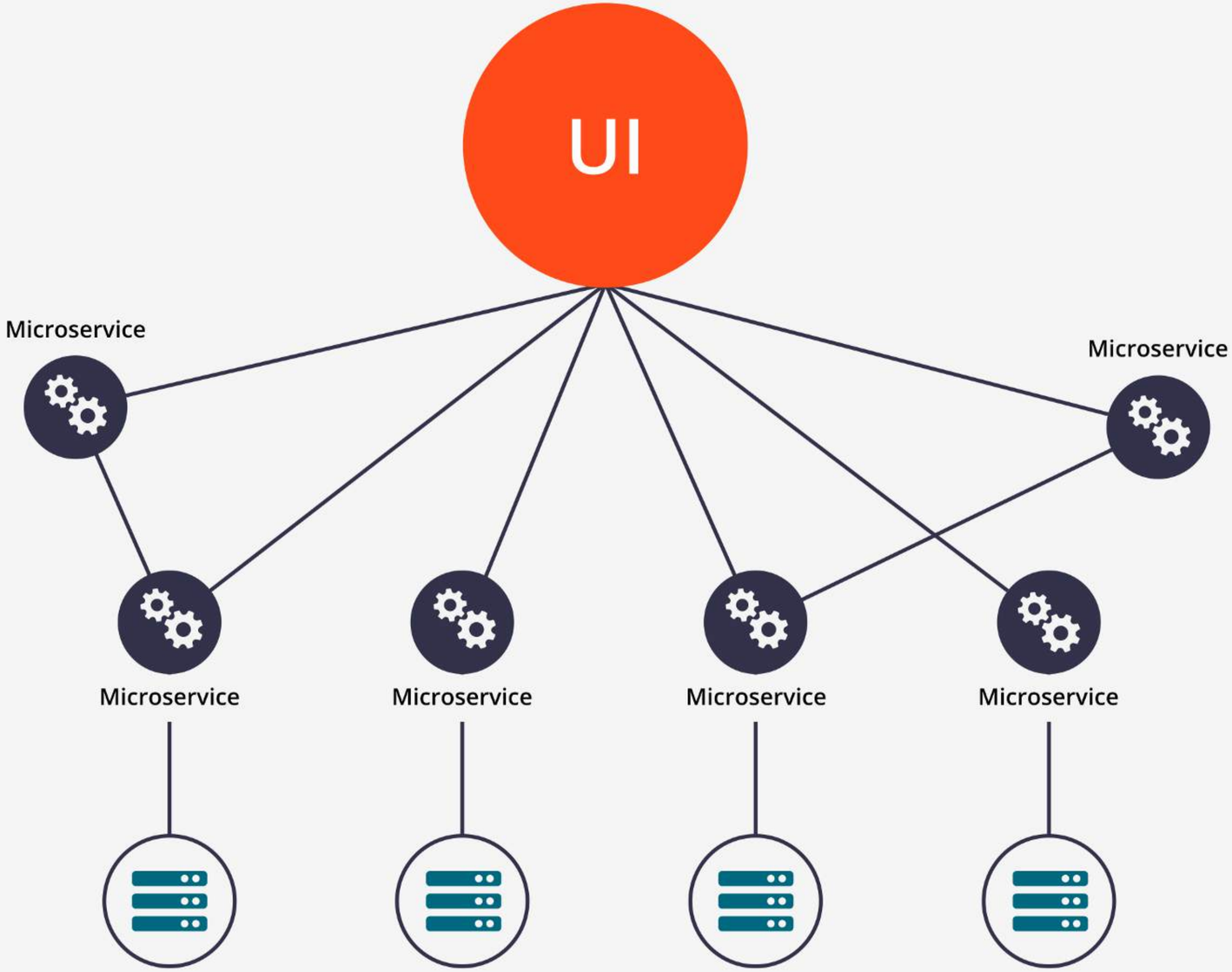
- In contrast **microservices** are many small services in a **Service-Oriented Architecture** that are loosely coupled but all perform distinct roles
- They are relatively light weight and should only **do a single thing, but do it well**
- For example you could have a service that handles authentication that all other **microservices consume** when someone wants to authenticate against them
- And another **microservice** will only process a certain type of special order

- This allows you to scale up or down different parts of your applications depending on need
- It also allows you to **refactor parts of your application** without having to touch everything as a whole
- It’s also **MUCH** harder to implement correctly and requires **MUCH** more adherence to standards and communication between teams

- I’d advise to only use **microservices** when you have a massive codebase and it’s a last resort
- It requires **A LOT MORE** resources such as staff, servers, time, and money
- One example of a success story is Netflix
  - They use to have one massive application that constantly crashed
  - They broke it into tons of microservices and now they are one of the most reliable applications I can think of



Monolithic Architecture



Microservice Architecture

https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4

IaaS, Infrastructure Services

Scale Application Servers Ad Libitum

- Since a programmer can automation tools to, well, automate their servers it allows us to deploy the application ad infinitum and ad libitum

- These tools can use a server template (much like the VM I gave you at the beginning of the semester) and will bootstrap from a blank slate to a fully working instance of your application, perfectly set up, within a few minutes

- Imagine being able to "**Spin Up**" and "**Spin Down**" servers and services as needed

Cut servers to cut costs

- At this point since the **servers** are created from code they become **disposable**
- If they are idle, they are **costing you money** without generating you any revenue
- During periods of **down time** (say middle of the night) we start just killing servers to reduce overhead
- Netflix ramps up to thousands of servers during peak hours (5pm - 2am) then cuts down to bare minimum to reduce overhead in the off time

AWS Services

- **Amazon Web Services**, for example, has a lot of services that can help your scaling issues **automagically**
  - **RDS** - Relational Database Server, PSQL database that's clustered and highly available for a price
  - **EC2** - Elastic Compute Cloud, Virtual machines from templates on demand
  - **Route 53** - Scalable and redundant DNS servers
  - **ELB** - Elastic Load Balancer, Load balancing service that's highly available and scalable

AWS Elastic Beanstalk

- **EBS** or ***E*lastic *B*ean*s*talk** is a service that amazon provides that will, under certain circumstances, automatically scale your application or reduce it

- You give your **EBS** instance a copy of your code and definition files on what needs to be installed, where, and how it's configured and it will spin up servers or down servers depending on:

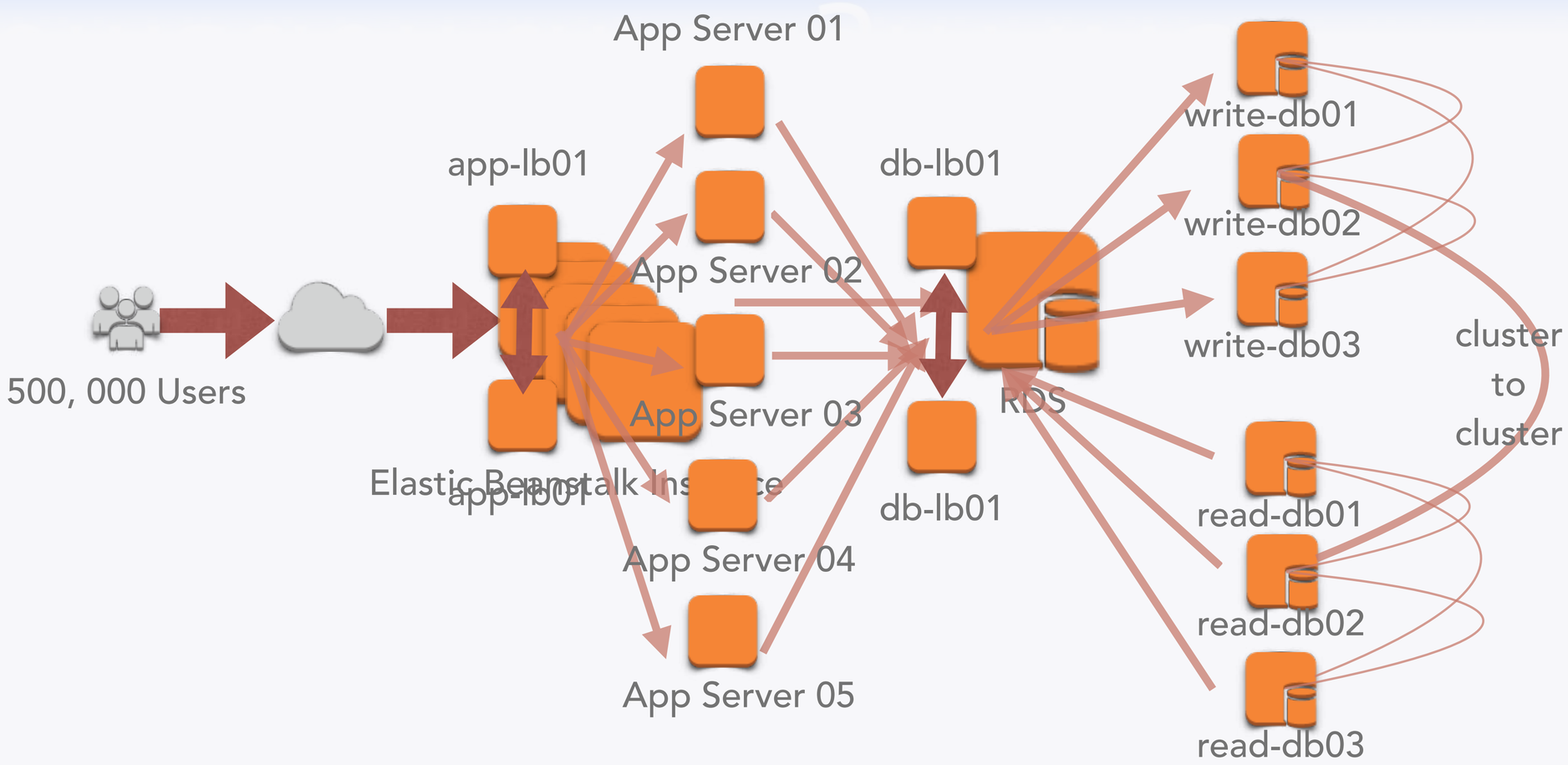
  - **Network Traffic**

  - **CPU or Disk Load**

  - **Latency of request** (how fast they are being processed)

- After a certain **threshold is met** it will proactively spin up the instances
- After the **threshold is no longer met** and some time (about an hour on average) it will start to kill servers until a minimum that you've specified are remaining
- Since the act of **spinning up a server** takes time it's better to lead into it
- Basically it's better to start the process of spinning up before the load becomes an issue. Trying to spin up servers after the fact that you're overwhelmed is a bad idea

Automation through AWS



Caching and CDNs

There are 2 hard problems in computer science:

1. Naming things

2. Cache Invalidation

3. Off by one errors

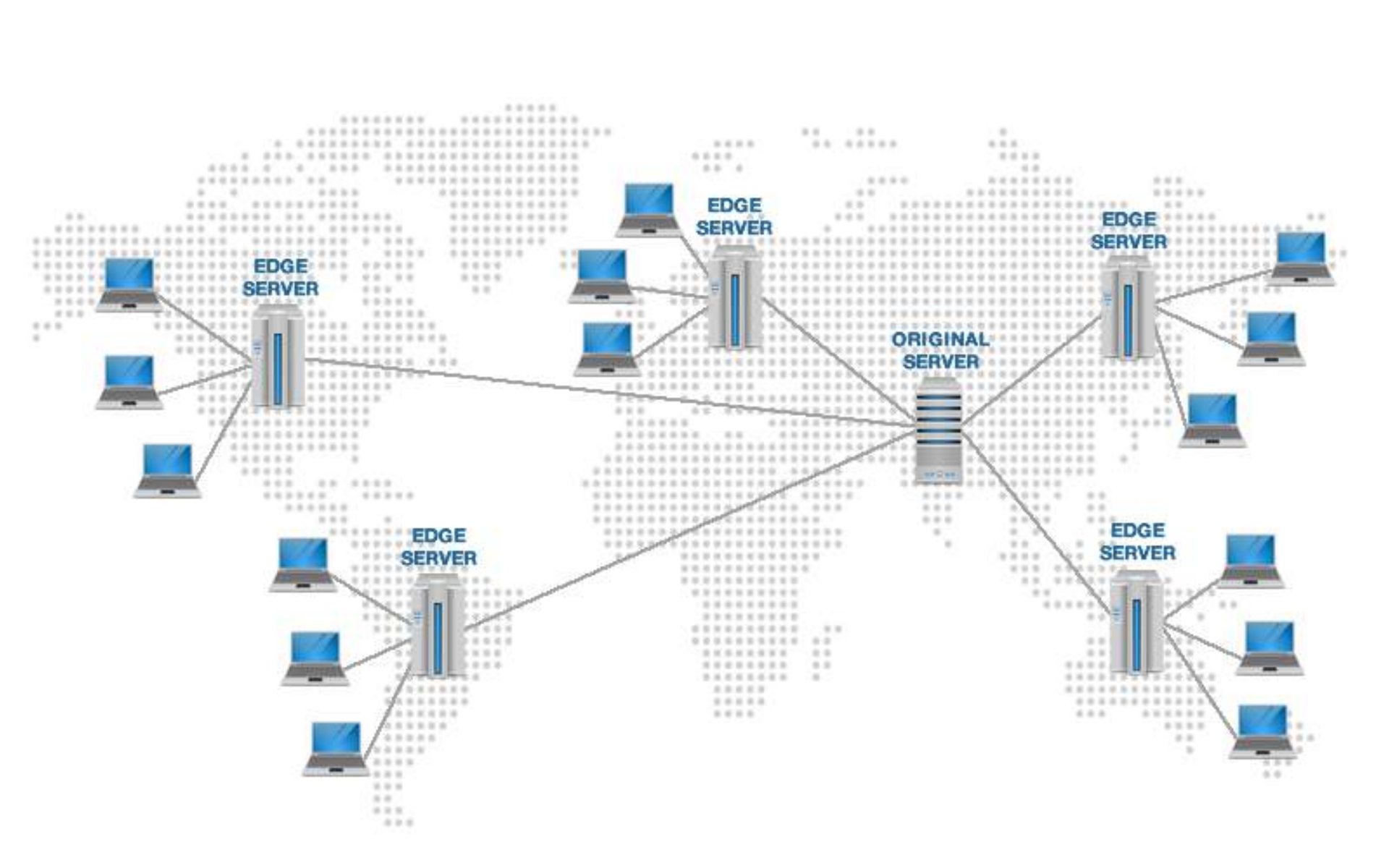
–Jeff Atwood

Website Caching

- A great way to improve performance is **caching** things that don't change
- For example, your CSS or Images don't change that often so **caching** them and serving them independent of your application will help speed up the process
- Even some applications can be mostly **cached**; for example Netflix product pages look the same to everyone in a country so why not precompute it, cache it, and serve it statically to save time and money
- We can use a service called **Varnish** that acts as a caching proxy in front of a lot of static things

Content Delivery Networks

- **CDN** or **Content Delivery Network** is a set of geographically distributed network of proxy servers
- That means that there are servers **physically located** all throughout the world because distance means speed
- When an asset is cached through a CDN the first land is slow because it's a **cache miss**, aka the asset isn't in the current cache
- From each subsequent fetch, anyone that's physically located near that server will be served with a copy located on that geographically close server which is much faster



https://www.themexpert.com/blog/cdn-service-make-joomla-website-load-faster

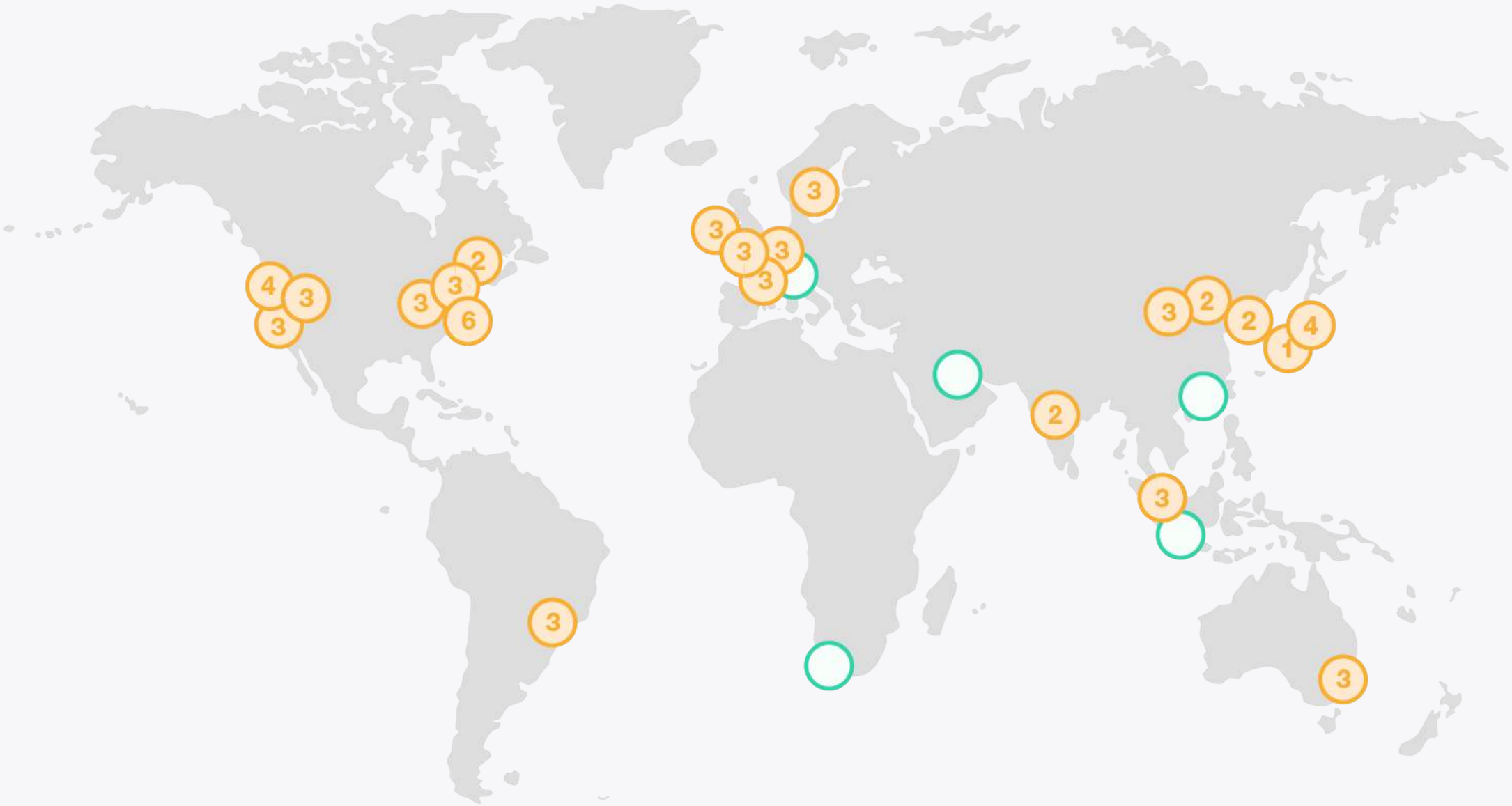
World Wide Distribution

Multiple Datacenters

- At some point you will need to expand your in demand app to **multiple datacenter across the world**
- Since **distance does play a factor in speed** (think how long it takes to load a UK site) you would ideally want some servers in a datacenter on each continent
- This gets **EXTREMELY COSTLY** but if it's generating profit it should outweigh the negatives

AWS Regions and Availability Zones

- In the AWS world this is called **Regions** and **Availability Zones** (**AZ**)
- An AWS **Region** is a specific set of datacenters
- An **Availability Zone** is a single datacenter in that region set
- Using **AWS Services** you can specify that you want it distributed amongst different **Regions** / **AZ**s automagically
- This will increase your resilience but at the end of the day it will cost you a ton of money



https://aws.amazon.com/about-aws/global-infrastructure/

Questions?

Lecture 12 - Serverless Servers

Lambdas

What Is AWS Lambda

- A compute service that lets you run code without provisioning or managing servers
- Executes your code only when needed and scales automatically, from a few requests per day to thousands per second
- Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources
- You can use AWS Lambda to run your code in response to events, such as changes to data in an Amazon S3 bucket or an Amazon DynamoDB table

When Should I Use AWS Lambda

- When using AWS Lambda, you are responsible only for your code
- Lambda manages the compute fleet that offers a balance of memory, CPU, network, and other resources
- This is in exchange for flexibility, which means you cannot log in to compute instances, or customize the operating system or language runtime

- These constraints enable AWS Lambda to perform operational and administrative activities on your behalf, including
  - provisioning capacity
  - monitoring fleet health
  - applying security patches
  - deploying your code
  - monitoring and logging your Lambda functions

Building Blocks of a Lambda-based Application

- **Lambda function**
  - The foundation, it is comprised of your custom code and any dependent libraries
- **Event source**
  - An AWS service, such as Amazon SNS, or a custom service, that triggers your function and executes its logic
- **Downstream resources**
  - An AWS service, such as DynamoDB tables or Amazon S3 buckets, that your Lambda function calls once it is triggered

- **Log streams**
  - Annotate your function code with custom logging statements that allow you to analyze the execution flow and performance of your Lambda function to ensure it's working properly
- **AWS SAM**
  - A model to define serverless applications. AWS SAM is natively supported by AWS CloudFormation and defines simplified syntax for expressing serverless resources

Lambda Functions

Working with Functions

- **Function Code**
  - Separate the Lambda handler (entry point) from your core logic
  - Take advantage of Execution Context reuse to improve the performance of your function
  - Use AWS Lambda Environment Variables to pass operational parameters to your function
- **Function Configuration**
  - Performance testing your Lambda function
  - Load test your Lambda function
  - Use most-restrictive permissions when setting IAM policies
- **Alarming and Metrics**
  - Use AWS Lambda Metrics and CloudWatch Alarms instead of within your function code

- **Stream Event Invokes**
  - Test with different batch and record sizes
  - Increase Kinesis stream processing throughput by adding shards
- **Async Invokes**
  - Create and use Dead Letter Queues to address and replay async function errors

Building Lambda Functions

- You upload your application code in the form of one or more Lambda functions to AWS Lambda, a compute service
- Typically, the lifecycle for an AWS Lambda-based application includes, general questions that come up in each of these lifecycle phases:
    - Authoring code for your Lambda function
        - What languages are supported?
        - Is there a programming model that I need to follow?
        - How do I package my code and dependencies for uploading to AWS Lambda?
    - Uploading code and creating Lambda functions
        - How do I upload my code package to AWS Lambda?
        - How do I tell AWS Lambda where to begin executing my code?
    - Monitoring and troubleshooting
        - For my Lambda function that is in production, what metrics are available?
        - If there are any failures, how do I get logs or troubleshoot issues?

Authoring Code

- You can author your Lambda function code in the languages that are supported by AWS Lambda
- There are tools for authoring code, such as the AWS Lambda console, Eclipse IDE, and Visual Studio IDE
  - But the available tools and options depend on:
    - Language you choose to write your Lambda function code
    - Libraries that you use in your code

- The Available languages include:
  - Node.js
  - Python
  - C#
  - Java
  - Go
  - PowerShell

Deploying Code

- Creating a Deployment Package – Organizing Code and Dependencies
  - You must first organize your code and dependencies in certain ways and create a deployment package
  - Instructions to create a deployment package vary depending on the language you choose to author the code
- Uploading a Deployment Package – Creating a Lambda Function
  - AWS Lambda provides the CreateFunction operation, which is what you use to create a Lambda function
  - You can use the AWS Lambda console, AWS CLI, and AWS SDKs to create a Lambda function

- Testing a Lambda Function
  - If your Lambda function is designed to process events of a specific type, you can use sample event data to test your Lambda function using one of the following methods:
    - Test your Lambda function in the console
    - Test your Lambda function using the AWS CLI
    - Test your Lambda function locally using the AWS SAM CLI

Monitoring And Troubleshooting

- AWS Lambda automatically monitors functions on your behalf, reporting metrics through Amazon CloudWatch
- These metrics include total requests, duration, and error rates
- You can monitor metrics for Lambda and view logs by using the Lambda console, the CloudWatch console, the AWS CLI, or the CloudWatch API
- The following procedures show you how to access metrics using these different methods
- Lambda logs all requests handled by your function and also automatically stores logs that your code generates in Amazon CloudWatch Logs

Program Model

- **Handler**
  - The function AWS Lambda calls to start execution of your Lambda function
  - You identify the handler when you create your Lambda function
  - When a Lambda function is invoked, AWS Lambda starts executing your code by calling the handler function
- The context object and how it interacts with Lambda at runtime
  - AWS Lambda also passes a context object to the handler function, as the second parameter
  - Via this context object your code can interact with AWS Lambda

- **Logging**
  - Your Lambda function can contain logging statements
  - AWS Lambda writes these logs to CloudWatch Logs
- **Exceptions**
  - Your Lambda function needs to communicate the result of the function execution to AWS Lambda
  - Depending on the language you author your Lambda function code, there are different ways to end a request successfully or to notify AWS Lambda an error occurred during execution

Configuring Lambdas

Basic Configuration

- **Memory**
  - Specify the amount of memory you want to allocate for your Lambda function
  - Lambda allocates CPU power proportional to the memory by using the same ratio as a general purpose Amazon EC2 instance type
- **Timeout**
  - To prevent your Lambda function from running indefinitely, you specify a timeout
  - When the specified timeout is reached, AWS Lambda terminates execution of your Lambda function

- **Execution role**
  - This is the role that AWS Lambda assumes when it executes the Lambda function on your behalf
- **Handler name**
  - The handler method is the entry point that executes your Lambda function code and any event source dependencies that you have included as part of your Lambda function

Environment Variables

- Environment variables for Lambda functions enable you to dynamically pass settings to your function code and libraries, without making changes to your code
- Environment variables are key-value pairs that you create and modify as part of your function configuration
- Lambda then makes these key value pairs available to your Lambda function code using standard APIs supported by the language
- You can use environment variables to:
    - Help libraries know what directory to install files in
    - Where to store outputs
    - Store connection and logging settings
    - Etc.

VPC Settings

- Typically, you create resources inside Amazon Virtual Private Cloud (Amazon VPC) so that they cannot be accessed over the public Internet
- Configuring a Lambda Function for Amazon VPC Access
  - You add VPC information to your Lambda function configuration using the VpcConfig parameter
  - When you add VPC configuration to a Lambda function, it can only access resources in that VPC

- Internet Access for Lambda Functions
  - Lambda uses the VPC info you provide to set up ENIs that allow your function to access VPC resources
  - Each ENI is assigned a private IP address from the IP address range within the Subnets you specify, but is not assigned any public IP addresses
- Guidelines for Setting Up VPC-Enabled Lambda Functions
  - Your Lambda function automatically scales based on the number of events it processes
  - If your VPC does not have sufficient ENIs or subnet IPs, your Lambda function will not scale as requests increase, and you will see an increase in invocation errors (EC2ThrottledException)

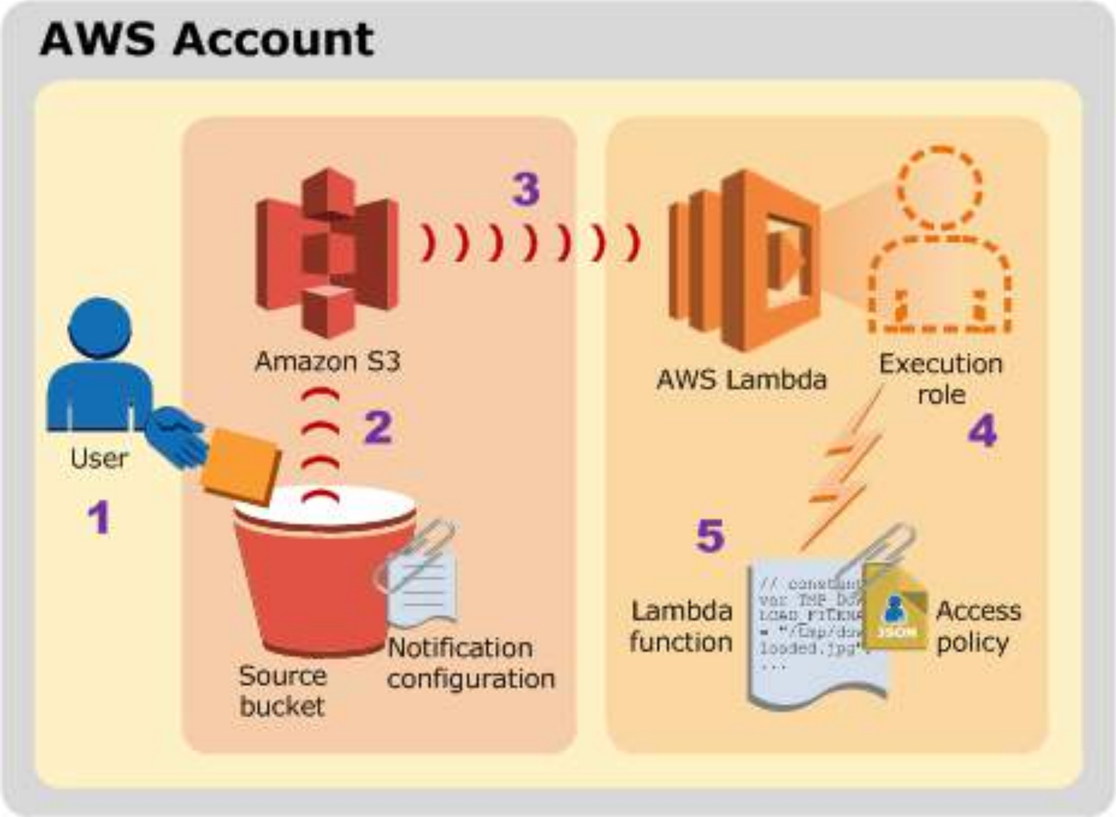
Invoking Functions

Event Source

- Lambda functions and event sources are the core components of Lambda
- An event source is the entity that publishes events, and a Lambda function is the custom code that processes the events
- Each event source mapping identifies the type of events to publish and the Lambda function to invoke when events occur
- You can also create custom applications to include AWS resource events and invoke a Lambda function

S3 Pushes Event

1. The user creates an object in a bucket
2. S3 detects the object created event
3. S3 invokes your Lambda function according to the event source mapping described in the bucket notification configuration
4. Lambda verifies the permissions policy attached to the Lambda function to ensure that S3 has the necessary permissions
5. Once Lambda verifies the attached permissions policy, it executes the Lambda function

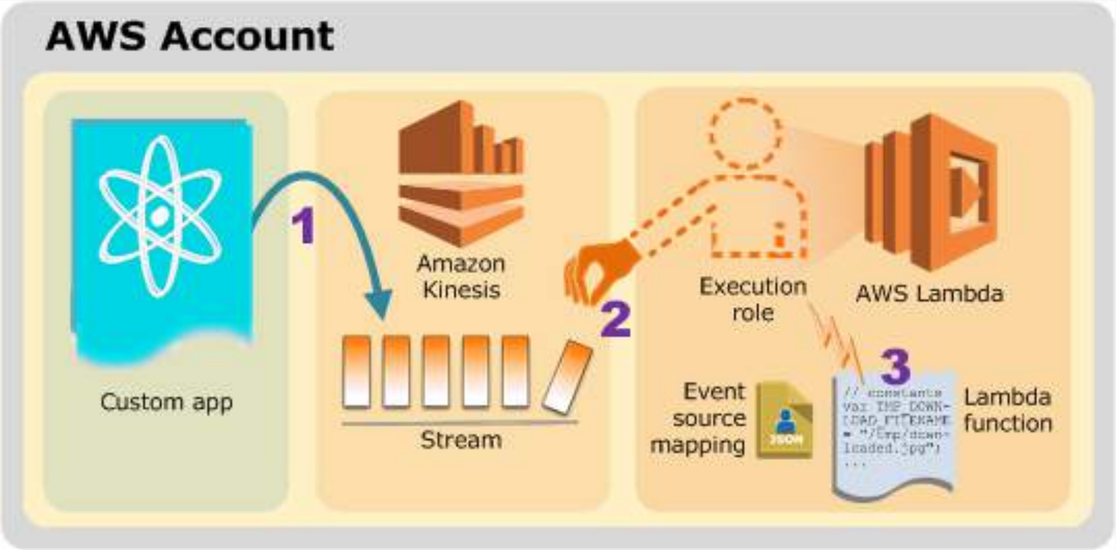


Kinesis Pull Events

1. The custom application writes records to an Amazon Kinesis stream

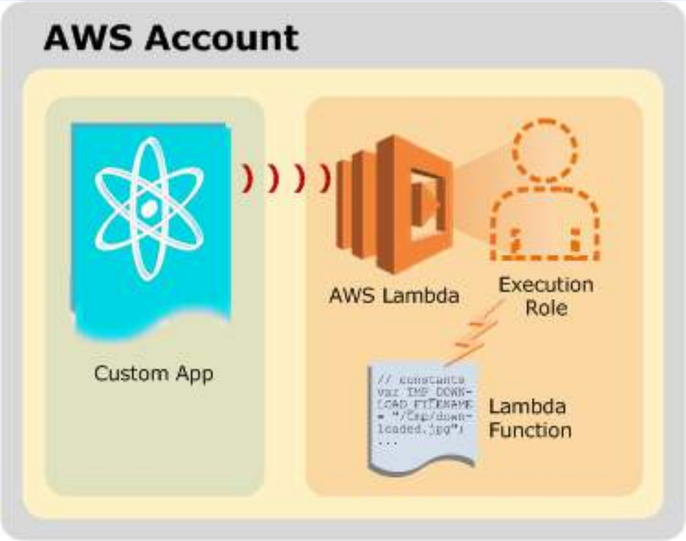
2. Lambda continuously polls the stream and invokes the Lambda function once the service detects new records on the stream

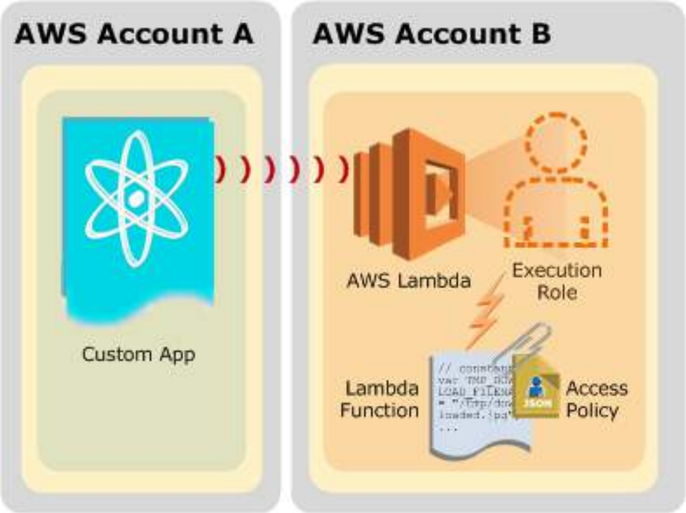
3. Assuming the attached permission policy, which allows AWS Lambda to poll the stream, is verified, AWS Lambda then executes the Lambda function



Custom App Pushes Event

- If you have custom applications that publish and process events, you can create a Lambda function to process these events
- there is no preconfiguration required—you don't have to set up an event source mapping
- Custom Application Publishes Events and Invokes a Lambda Function
  - Diagram A shows how a custom application in your account can invoke a Lambda function
  - Diagram B Shows the user application and Lambda function are owned by different AWS accounts
    - The AWS account that owns the Lambda function must have cross-account permissions in the permissions policy associated with the Lambda function





Invocation Types

- Lambda supports synchronous and asynchronous invocation of a function
- You can control the invocation type only when you invoke a Lambda function
- On-demand invocations:
  - Your custom application invokes a Lambda function
  - You manually invoke a Lambda function for testing purposes
- In both cases, you invoke your Lambda function using the Invoke operation, and you can specify the invocation type as synchronous or asynchronous
- When you use an AWS services as a trigger, the invocation type is predetermined for each service

Applications

Runtime Environments

- Lambda supports multiple languages through the use of runtimes
- You choose a runtime when you create a function, and you can change runtimes by updating your function's configuration
- When your function is invoked, Lambda attempts to re-use the execution environment from a previous invocation if one is available
    - This saves time preparing the execution environment, and allows you to save resources like database connections
- A runtime can support a single version of a language, multiple versions of a language, or multiple languages

Serverless Application Model

- The AWS Serverless Application Model (AWS SAM) is an open-source framework you can use to build serverless applications on AWS
- AWS SAM template specification
  - You use this specification to define your serverless application
  - It provides you with a simple and clean syntax to describe the functions, APIs, permissions, configurations, and events that make up a serverless application
  - You use an AWS SAM template file to operate on a single, deployable, versioned entity that's your serverless application

- AWS SAM command line interface
  - You use this tool to build serverless applications that are defined by AWS SAM templates
  - The CLI provides commands that enable you to
    - Verify that AWS SAM template files are written according to the specification
    - Invoke Lambda functions locally
    - Step-through debug Lambda functions
    - Package and deploy serverless applications to the AWS Cloud
    - Etc

Use Case: S3

- S3 can publish events to Lambda and invoke your function by passing the event data as a param
- **Non-stream based (async) model**
  - This is a model, where Amazon S3 monitors a bucket and invokes the Lambda function by passing the event data as a parameter
- **Asynchronous invocation**
  - AWS Lambda invokes a Lambda function using the Event invocation type

- **Event structure**
  - The event your Lambda function receives is for a single object and it provides information, such as the bucket name and object key
- Note that there are two types of permissions policies that you work with when you set up the end-to-end experience:
  - **Permissions for your Lambda function** – Regardless of what invokes a Lambda function, Lambda executes the function by assuming the IAM role (execution role) that you specify at the time you create the Lambda function
  - **Permissions for S3 to invoke your Lambda function** – S3 cannot invoke your Lambda function without your permission

Use Case: SQS

- You can use an AWS Lambda function to process messages in a standard Amazon Simple Queue Service (Amazon SQS) queue
- you can offload tasks from one component of your application by sending them to a queue and processing them asynchronously
- Lambda uses long polling to poll a queue until it becomes active
  - When messages are available, Lambda increases the rate at which it reads batches, and invokes your function until it reaches a concurrency limit

- When Lambda reads a message from the queue, it stays in the queue but becomes hidden until Lambda deletes it
- If your function returns an error, or doesn't finish processing before the queue's visibility timeout, it becomes visible again

Use Case: API Gateway

- Amazon API Gateway also adds a layer between your application users and your app logic that enables the following:
  - Ability to throttle individual users or requests
  - Protect against Distributed Denial of Service attacks
  - Provide a caching layer to cache response from your Lambda function

- Note the following about how the Amazon API Gateway and AWS Lambda integration works
    - **Push-event model** – This is a model, where Amazon API Gateway invokes the Lambda function by passing data in the request body as parameter to the Lambda function.
    - **Synchronous invocation** – The Amazon API Gateway can invoke the Lambda function and get a response back in real time by specifying RequestResponse as the invocation type
    - **Event structure** – The event your Lambda function receives is the body from the HTTPS request that Amazon API Gateway receives and your Lambda function is the custom code written to process the specific event type

Questions?