Geohashing and Quadtrees

Wednesday, 14 February 2024 5:21 PM

Geohashing

- geocoding method used to encode geographic coordinates such as latitude and longitude into short alphanumeric strings.

eg: San Francisco with coordinates 37.7564, -122.4016 can be represented in geohash as 9q8yy9mf.

How does Geohashing work?

Geohash is a hierarchical spatial index that uses Base-32 alphabet encoding, the first character in a geohash identifies the initial location as one of the 32 cells. This cell will also contain 32 cells. This means that to represent a point, the world is recursively divided into smaller and smaller cells with each additional bit until the desired precision is attained. The precision factor also determines the size of the cell.

Geohashing guarantees that points are spatially closer if their geohashes share a longer prefix which means the more characters in the string, the more precise the location. eg: geohashes 9q8yy9mf and 9q8yy9vx are spatially closer as they share the prefix.

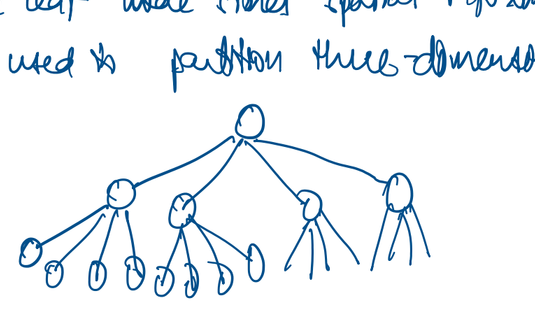
Geohashing can also be used to provide a degree of anonymity as we don't need to expose the exact location of user because depending on length of the geohash we just know they are somewhere within an area.

Use cases:
- simple way to represent and store a location in db.
- can be shared on social media as URLs since it is easier to share, and remember than latitudes and longitudes.
- can efficiently find the nearest neighbours of a point through a simple string comparisons and efficient searching of indexes

widely used and supported by popular db:
- MySQL, Redis, Amazon DynamoDB, Google Cloud Firestore

Quadtrees

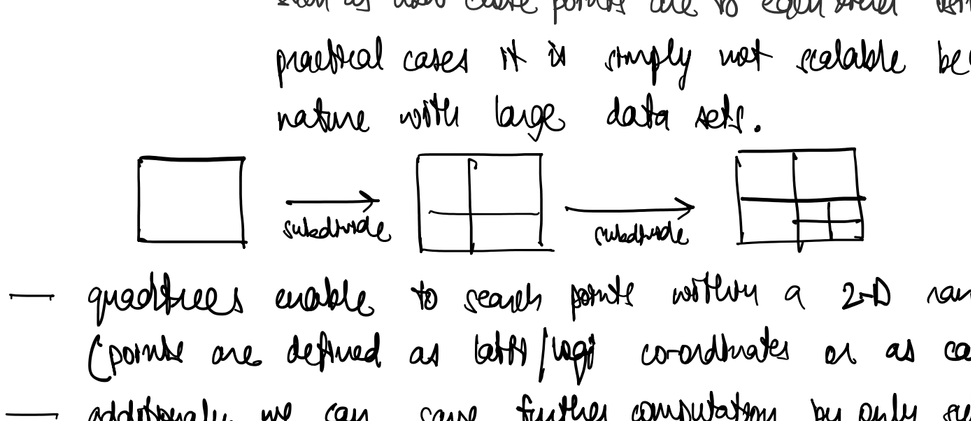
A tree data structure in which each internal node has exactly four children. Often used to partition a 2-D space by recursively subdividing it into four quadrants or regions. Each child or leaf node stores spatial information. Quadtrees are 2-D analog of Octrees which are used to partition three-dimensional space.



Types
- classified according to type of data they represent
  - areas, points, lines and curves
- common type quadtrees:
  - point quadtrees
  - point-region (PR) quadtrees
  - polygonal map (PM)
  - compressed
  - edge

Why do we need quadtrees?
- aren't latitudes and longitudes enough?

while in theory using latitude and longitude we can determine things such as how close points are to each other using euclidean distance, for practical cases it is simply not scalable because of its CPU-intensive nature with large data sets.



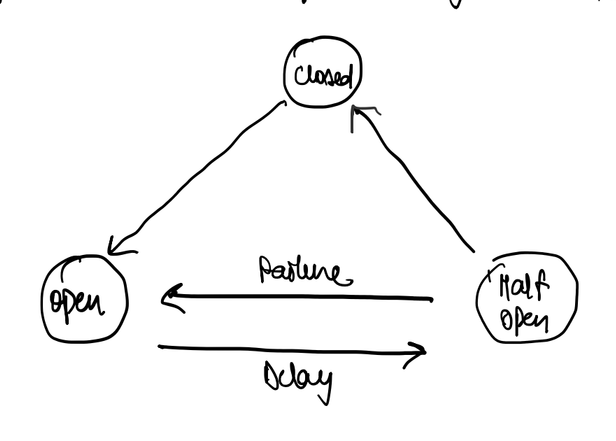
- quadtrees enable to search points within a 2-D range efficiently (points are defined as latitude/longitude coordinates or as cartesian (x,y) coordinates.
- additionally, we can save further computation by only subdividing a node after a certain threshold.

And with application of mapping algo such as the Hilbert curve, we can easily improve range query performance

Use cases:
- image representation, processing and compression
- spatial indexing and range queries
- location based services like google maps, Uber etc
- mesh generation and computer graphics
- sparse data storage

Circuit Breaker

- design pattern used to detect failures and encapsulates the logic of preventing a failure from constantly recurring during maintenance, temporary external system failure, or unexpected system difficulties.



Basic Idea

We wrap a protected function call in a circuit breaker object, which monitors for failures. Once the failures reach a certain threshold, the circuit breaker trips, and further calls to the circuit breaker return with an error, without the protected call being made at all.
- usually, we'll also want some kind of monitor alert if the circuit breaker trips.

Why do we need circuit breaking?
- common for software systems to make remote calls to software running in different processes probably on different machines across a network.
- Big difference: Remote calls can fail, or hang without a response until some timeout is reached. In-memory calls are opposite.
- what's worse — If we have many callers on an unresponsive supplier, then we can run out of circuit resources leading to cascading failures across multiple systems.

States

Circuit breaker states:
- Closed: when everything is normal, the circuit breakers remain closed, and all the request passes through to the services as normal.
  - If no. of failures increases beyond the threshold, the circuit breaker trips and goes into an open state.
- Open: In this state circuit breaker returns an error immediately without even invoking the process
  - circuit breakers move into half open state after a certain timeout period elapses.

    Usually, it will have a monitoring system where the timeout will be specified
- Half Open: circuit breaker allows a limited no. of requests from the service to pass through and invoke the operation.
  - If the requests are successfully, then the circuit breaker will go to the closed state
  - However, if the requests continue to fail, then it goes back to the open state.

Rate Limiting

- refers to preventing the frequency of an operation from exceeding a defined limit.
- In large-scale systems, rate limiting is commonly used to protect underlying services and resources.
- Rate limiting — defensive mechanism in distributed systems, so that shared resources can maintain availability.
- protects our APIs from unintended or malicious overuse by limiting the no. of requests that can reach our API in a given period of time

Why do we need?

Very important part of any large-scale systems:
- Avoid resource starvation as a result of Denial of service (DoS) attack.
- Rate limiting helps in controlling operational costs by putting a virtual cap on the auto-scaling of resources which if not monitored might lead to exponential bills.
- can be used as defense or mitigation against some common attacks
- For APIs that process massive amount of data, rate limiting can be used to control the flow of that data.

Algorithms

1) Leaky bucket

algorithm that provides a simple, intuitive approach to rate limiting via a queue.
- when registering a request, the system appends it to the end of queue.
- Processing happens FIFO order. If queue is full, then additional requests are discarded (or leaked)

ii) Token Bucket

uses a concept of bucket.
- when request comes in, a token from bucket must be taken and processed.
- request will be refused if no token is available in the bucket, and the requester will have to try again later
- As a result, the token bucket gets refreshed after certain time period.

(iii) Fixed window

uses a window size of n seconds to track the fixed window algorithm rate
- each incoming request increments the counter for the window.
- If discard the requests if the counter exceeds a threshold.

(iv) Sliding window

hybrid approach that combines the fixed window algorithm's low processing cost and the sliding log's improved boundary conditions.
- like the fixed window algorithm, we track a counter for each fixed window. Next we account for a weighted value of previous window's request rate based on the current timestamp to smooth out bursts of traffic

Rate Limiting in Distributed Systems

Becomes complicated when distributed systems are involved. The two broad problems that come with rate limiting in distributed systems are:

Inconsistencies
- when using a cluster of multiple nodes, we might need to enforce a global rate limit policy.
- because if each node were to track its rate limit, a consumer could exceed a global rate limit when sending requests to different nodes
- the greater the no. of nodes, the more likely the user will exceed the global limit

Simple way to solve this problem is to use sticky sessions in our load balancers so that each consumer gets sent to exactly one node but this causes a lack of fault tolerance and scaling problems.

Another approach might be to use a centralized data store like Redis but this will increase latency and cause race conditions.

Race conditions
- Issue happens when we use a naive "get-then-set" approach, in which we retrieve the current rate limit counter, increment it, and then push it back to datastore.
- problem: additional requests can come through in the time it takes to perform a full cycle of read-increment-store, each attempting to store the increment counter with an invalid (lower) count value.

This allows a consumer to send a very large no. of requests to bypass the rate limiting controls.

One way to avoid is to use some sort of distributed locking mechanism around the key, preventing any other processes from accessing or writing to the counter.

Though the lock will become a significant bottleneck and will not scale. A better approach might be to use a "set-then-get" approach, allowing us to quickly increment and then check counter values without letting the atomic operations get in the way.

Service Discovery

- detection of services within a computer network.
- Service Discovery Protocol (SDP) is a network standard that accomplishes the detection of networks by identifying resources.

Need?

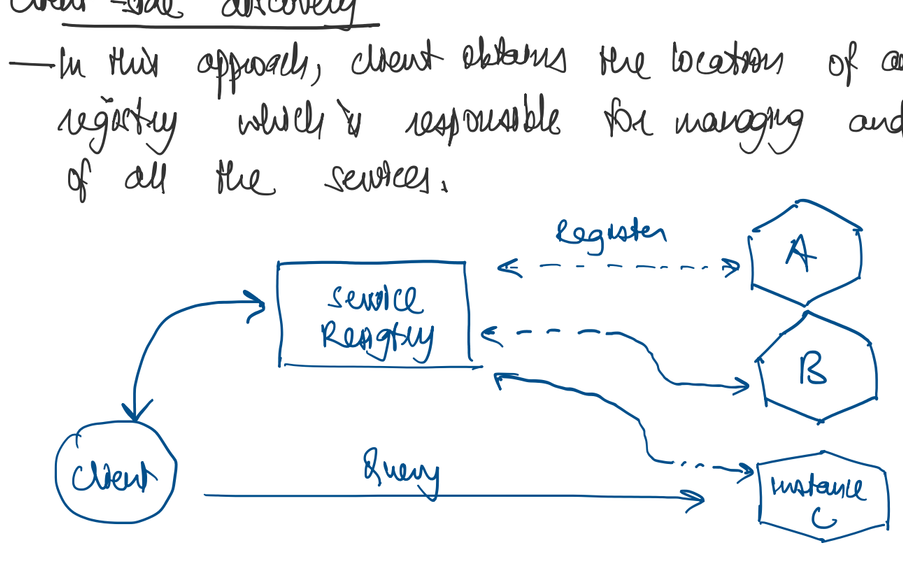
In a monolithic app, services invoke one another through language-level methods or procedure calls. However, modern microservices-based apps typically run in virtualized or containerized environments where the no. of instances of a service and their locations change dynamically.

Consecutively, we need a mechanism that enables the clients of services to make requests to a dynamically changing set of ephemeral service instances.

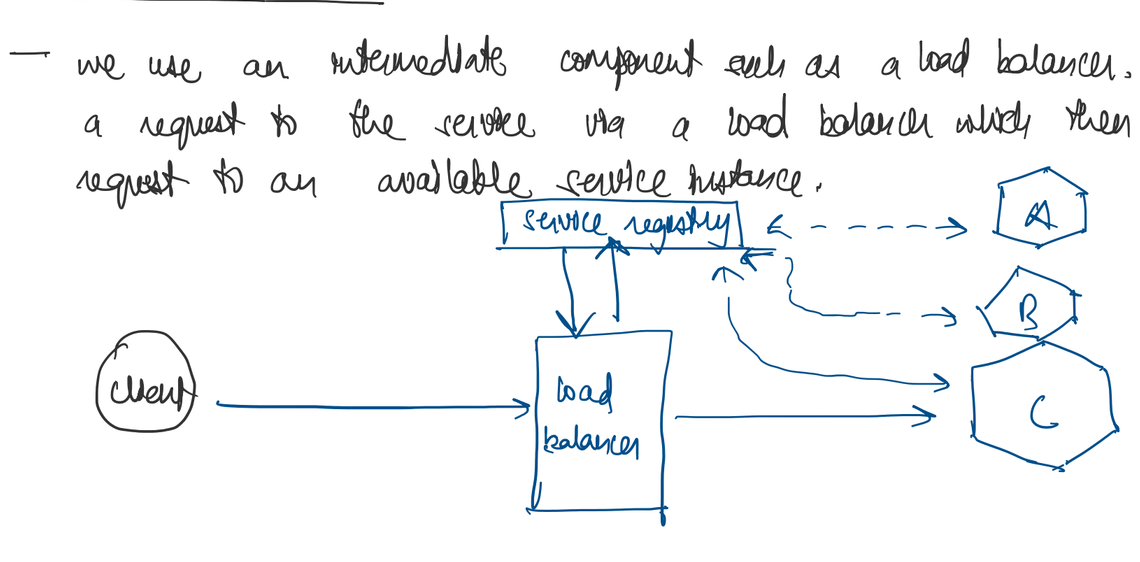
Implementations

Two main service discovery patterns:

Client-side discovery
- In this approach, client obtains the location of another service by querying a service registry which is responsible for managing and storing the network locations of all the services.



Server-side Discovery
- we use an intermediate component such as a load balancer. The client makes a request to the service via a load balancer which then forwards the request to an available service instance.



Service Registry

Basically a database containing the network locations of service instances to which the client can reach out.

Service registry must be highly available and up-to-date.

Service Registration
- need a way to obtain service information, often known as service registration.

Two possible approaches:
- Self Registration model
  - a service instance is responsible for registering itself in service registry.
  - In necessary, a service instance sends heartbeat requests to keep its registration alive
- Third-party registration
  - registry keeps a track of changes to running instances by polling the deployment environment or subscribing to events.
  - when it detects a newly available service instance, it records it in database.
  - Service-Registry also de-registers terminated service instances

Service Mesh
- service-to-service communication is essential in a distributed application but routing this communication, both within and across application clusters, becomes complex as no. of services grows.
- service mesh enables managed, observable, and secure communication between individual services.
- It works with a service discovery protocol to detect services. (Istio and envoy are some of most commonly used service mesh tech)

eg: etcd, Consul, Apache Thrift.

SLA, SLO, SLI

- mostly related to business and site reliability side of things but good to know.
- SLAs, SLOs, SLIs allow companies to define, track and monitor the promises made for a service to its users.
- Together, SLAs, SLOs, SLIs help teams generate more user trust in their services with an added emphasis on continuous improvement to incident management and response processes.

SLA
- An SLA, or service level agreement, is an agreement made between a company and its users of a given service.
- SLA defines the different promises that the company makes to users regarding specific metrics, such as service availability

SLAs are often written by a company's business or legal team.

SLO
- An SLO, or service level objective, is the promise that company makes to its users regarding a specific metric such as incident response or uptime.
- SLOs exist within an SLA as individual promises contained within the full user agreement.
- The SLO is the specific goal that service must meet in order to comply with the SLA.
- SLOs should always be simple, clearly defined, and easily measured to determine whether or not the objective is being fulfilled

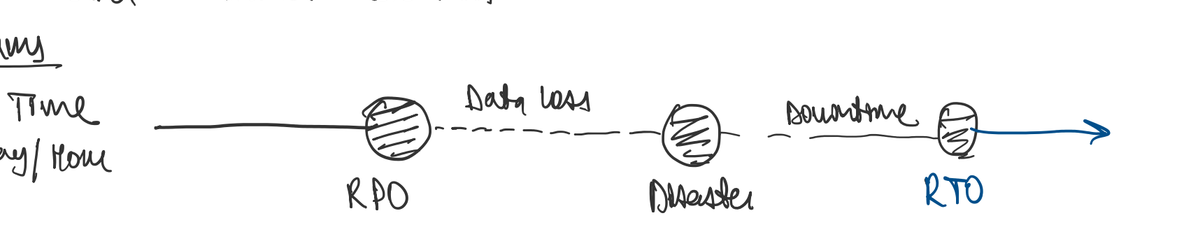
SLI
- An SLI, service level indicator, is a key metric used to determine whether or not the SLO is being met.
- It is the measured value of the metric described within the SLO.
- In order to remain in compliance with the SLA, the SLI's value must always meet or exceed the value determined by SLO.

Disaster Recovery (DR)

- process of regaining access and functionality of the infrastructure after events like a natural disaster, cyber attack, or even business disruptions.
- relies upon replication of data and computer processing in an off-premise location not affected by the disaster.
- Ideally, an organization can transfer its computing processing to that remote location as well in order to continue operations.

Benefits
- minimizes interruption and downtime
- limit damages
- fast restoration
- better customer retention

Terms



RTO — Recovery Time Objective
- maximum acceptable delay between the interruption of service and restoration of service.
- RTO determines what is considered an acceptable time window when service is unavailable

RPO — Recovery Point Objective
- maximum acceptable amount of time since the last data recovery point.
- determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

Strategies

variety of DR strategies can be part of a DR plan.
- Back-up

  simplest type of DR and involves storing data offsite or on a removable drive
- Cold Site

  an organization sets up basic infrastructure in a second site
- Hot Site

  hot site maintains up-to-date copies of data at all times.
  - Time-consuming to set up and more expensive than cold sites, but they dramatically reduce downtime.

SSL, TLS and mTLS

- important communication security protocols such as SSL, TLS, mTLS.

SSL
- secure socket layer, and refers to a protocol for encrypting and securing communications that take place on the internet.
- deprecated in favour of TLS (transport layer security)

Why is it called SSL certificate if it is deprecated?
- Most major certificate providers still refer to certificates as SSL, which is why the naming convention persists.

Why was SSL so important?
- originally, data on web was transmitted in plaintext that anyone could read if they intercepted the message.
- SSL was created to protect user privacy.
- By encrypting any data, SSL also stops certain kind of cyber attacks by preventing attackers from tampering with data in transit.

TLS
- widely adopted security protocol designed to facilitate privacy and data security for communications over the internet.
- Three main components to what the TLS protocol accomplishes:
  - Encryption — hides the data being transferred from third parties
  - Authentication — ensures that parties exchanging info are who they claim to be.
  - Integrity — verifies that the data has not been forged or tampered with.

mTLS
- mutual TLS a method for mutual authentication.
- mTLS ensures that the parties at each end of a network connection are who they claim to be by verifying that both have the correct private key.
- Information within their respective TLS certificates provides additional verification.

Benefits:
- ensures traffic is secure and trusted in both directions between client and server.
- mTLS is commonly used by microservices or distributed systems in a zero-trust security model to verify each other.

Virtual Machines (VMs) and Container

VM
- virtual environment that functions as a virtual computer system with its own CPU, memory, network interface and storage, created on a physical hardware system.
- software called hypervisor separates the machine's resources from the hardware and provisions them appropriately so they can be used by the VM.
- VMs are isolated from the rest of system, and multiple VMs can exist on a single piece of hardware, like a server.
- they can be moved between host servers depending on the demand or to use resources more efficiently.

What is hypervisor?
- sometimes called Virtual Machine Monitor (VMM), isolates the OS and resources from the virtual machines and enables the creation and management of those VMs.
- Hypervisor treats resources like CPU, memory and storage as a pool of resources that can be easily reallocated between existing guests or new VMs.

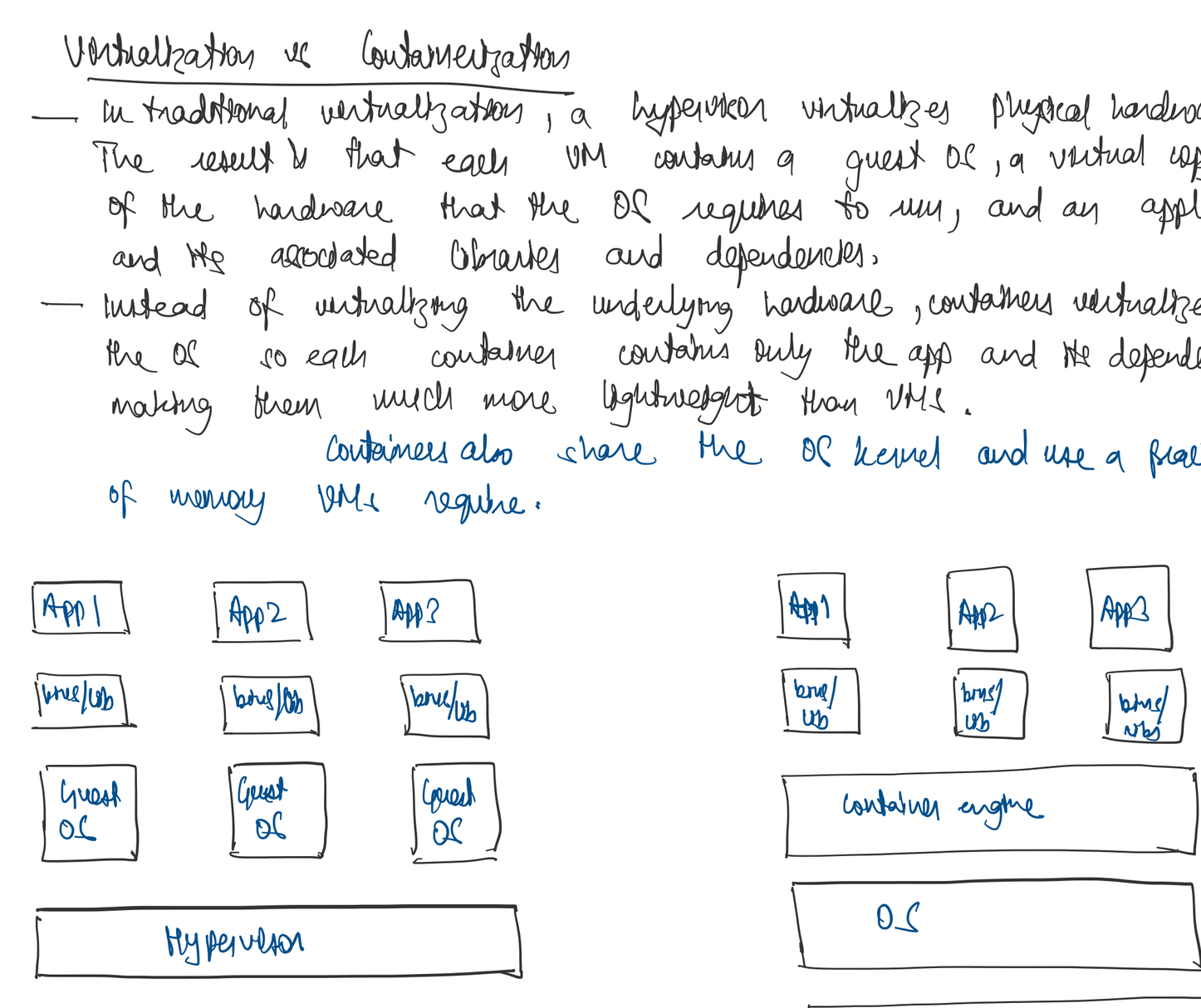
Why use VM?
- server consolidation is a top reason to use VMs.
- most OS and app deployment only use a small amount of the physical resources available.
- by virtualizing our servers, we can place many virtual servers onto each physical server to improve hardware utilization.
- VM provides an environment that is isolated from rest of system, so no interference with anything else running on host hardware
- Because VMs are isolated, they are a good option for testing new applications or setting up a production environment.
- we can also run a single-purpose VM to support a specific use case.

Containers
- standard unit of software that packages up code and all its dependencies such as specific versions of runtimes and libraries so that application runs quickly and reliably from one computing environment to another.
- containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.
- This decoupling allows container-based applications to be deployed easily and consistently, regardless of the target environment.

Need of containers
- Seperation of responsibility
  - provides clear separation of responsibility, as developers focus on application logic and dependencies, while operations team can focus on deployment and mgmt.
- workload portability
  - can run virtually anywhere, greatly easing development and deployment
- Application isolation
  - containers virtualize CPU, memory, storage and network resources at the OS level, providing developers with a view of the OS logically isolated from other applications
- Agile development
  - allows developers to move much more quickly by avoiding concerns about dependencies and environments
- Efficient operations
  - lightweight and allow us to use just the computing resources we need

Virtualization vs Containerization
- In traditional virtualization, a hypervisor virtualizes physical hardware. The result is that each VM contains a guest OS, a virtual copy of the hardware that the OS requires to run, and an application and its associated libraries and dependencies.
- Instead of virtualizing the underlying hardware, containers virtualize the OS so each container contains only the app and its dependencies making them much more lightweight than VMs.

Containers also share the OS kernel and use a fraction of memory VMs require.



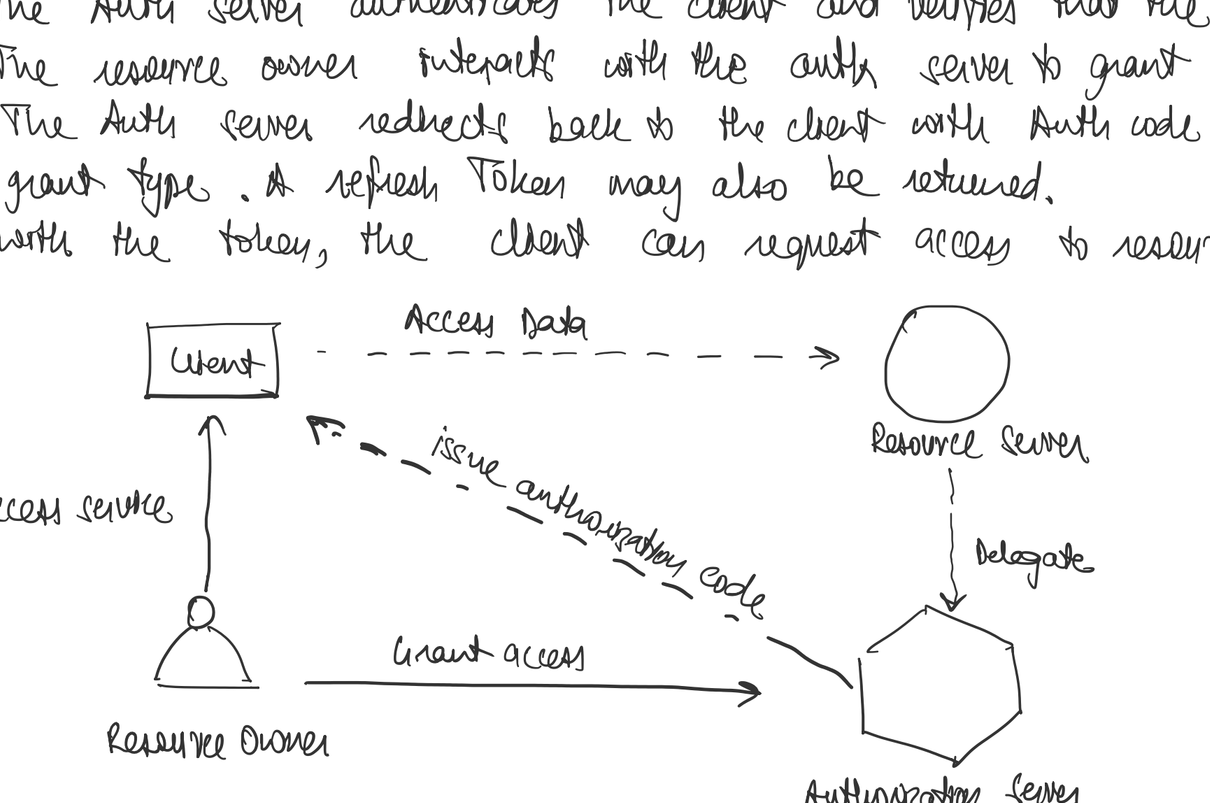
OAuth 2.0 and OpenID Connect (OIDC)

- stands for open authorization, is a standard design to provide consented access to resources on behalf of the user, without ever sharing the user's credentials.
- OAuth 2.0 is an authorization protocol and not an authentication protocol, designed primarily as a means of granting access to a set of resources, e.g. remote APIs or user's data.

Concept

OAuth 2.0 protocol defines following entities:
- Resource Owner: user or system that owns the protected resources can grant access to them
- Client: system that requires access to the protected resource
- Authorization Server: server receives requests from the client for Access Tokens and issues them upon successfull authentication and consent by resource owner.
- Resource Server: server that protects the user's resources and receives access requests from the client. It accepts and validates an Access Token from the client and returns the appropriate resource.
- Scopes: used to specify exactly the reason for which access to resources may be granted. Acceptable scope values, and which resources they relate to, are dependent on the Resource Server.
- Access Tokens: a piece of data that represents the authorization to access resources on behalf of end-user

Working of OAuth 2.0?
1. Client requests auth from the Auth Server, supplying the client id and secret as identification. It also provides the scopes and an endpoint URI to send the Access Token or Auth code.
2. The Auth server authenticates the client and verifies that the requested scopes are permitted.
3. The resource owner interacts with the auth server to grant access.
4. The Auth server redirects back to the client with Auth code or Access Token, depending on grant type. A refresh Token may also be returned.
5. With the token, the client can request access to resource from resource server



Disadv
- lacks built-in security features
- no standard implementation
- no common set of scopes

OpenID Connect
- thin layer that sits on top of OAuth 2.0 that adds login and profile information about person who is logged in.
- when auth server supports OIDC, it is sometimes called Identity Provider (IdP), since it provides information about the Resource Owner back to client.
- relatively new, resulting in lower adoption and industry implementation

Concepts

OIDC protocol defines the following entities:
- Relying Party: current app
- OpenID provider: essentially an intermediate service that provides a one-time code to relying party
- Token Endpoint: web server that accepts the one-Time code (OTC) and provides an access code that's valid for an hour

(main diff between OIDC and OAuth 2.0 is that the token is provided using Web Token (JWT))
- UserInfo Endpoint: relying party communicates with the endpoint, providing a secure token and receiving info about end-user.

* Both are easy to implement and are JSON based which is supported by most web and mobile app. However, OIDC specification is more strict than basic OAuth.

SAML vs OAuth 2.0 and OpenID Connect (OIDC)

- SAML uses XML to pass messages, while OAuth and OIDC use JSON.
- OAuth provides simpler experience, while SAML is geared towards enterprise security.
- OAuth and OIDC use restful communication.
- SAML drops a session cookie in a browser that allows user to access certain web pages (great for short-lived workloads)
- OIDC is developer-friendly and simpler to implement, which broadens the use cases for which it might be implemented.
- OIDC can be implemented from scratch pretty fast, via freely available libraries.
- SAML can be complex to install and maintain, which only enterprise-size companies can handle well.
- OpenID Connect is essentially a layer on top of OAuth framework.

  Therefore, it can offer a built-in-layer of permissions that asks a user to agree to what the service provider might access.
- Although SAML is also capable of allowing consent flow.

Advan
- Benefits of SSO:
  - ease of use as users only need to remember one set of credentials
  - " " access without having to go through a lengthy auth process.
  - enforced security and compliance to protect sensitive data.
  - simplify the mgmt with reduced IT support cost and admin time.

Disadv
- Single Password Vulnerability
- slower than traditional auth as every app has to request SSO provider for verification

eg: IdP → Okta, Google, Auth0, OneLogin etc

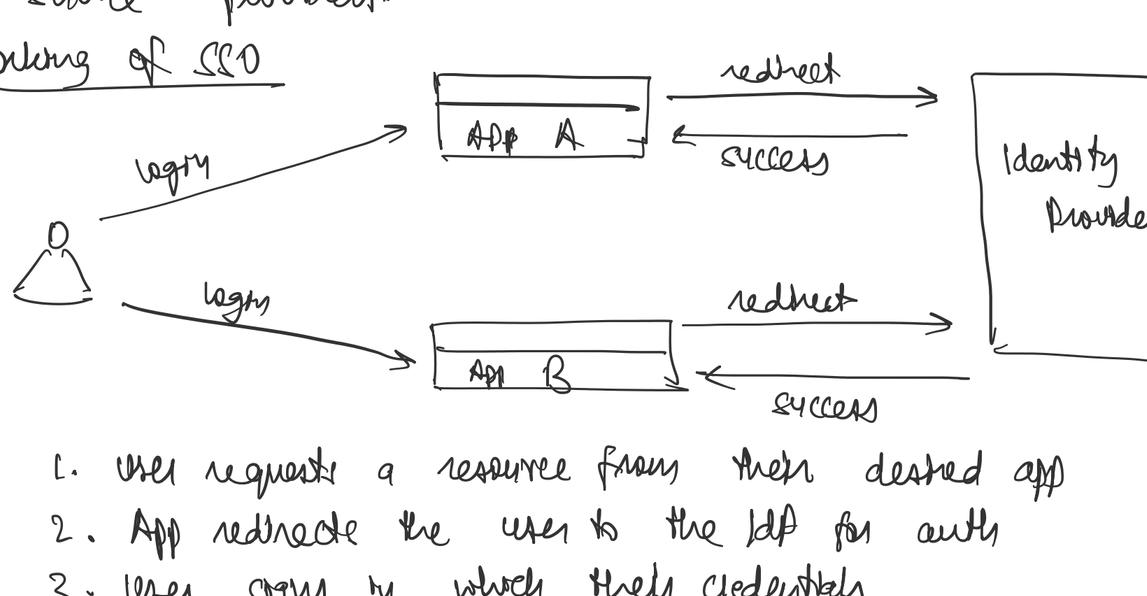
Single Sign-On (SSO)

- authentication process in which a user is provided access to multiple apps or websites by using only a single set of login credentials.
- This prevents need for user to log seperately onto the different applications.
- User credentials and other identifying info are stored and managed by a centralized system called Identity Provider (IdP).
- The Identity Provider is trusted system that provides access to other websites and apps.
- SSO based authentication systems are commonly used in enterprises environment where employee require access to multiple app of their organization.

Components
- Identity Provider (IdP)
  - User Identity info is stored and managed by a centralized system called IdP.
  - IdP can directly authenticate the user by validating a username and password or by validating an assertion about the user's identity as presented by a seperate identity provider.
  - IdP handles the mgmt of user identities in order to free the service provider from this responsibility.
- Service Provider
  - provides services to end-users.
  - rely on identity providers to assert the identity of user, and certain attributes about the user are managed by IdP.
  - service providers may also maintain a local account for the user along with attributes that are unique to their service.
- Identity Broker
  - acts as intermediary that connects multiple service providers with various different identity providers.
  - Using Identity Broker, we can perform single sign-on over any app without hassle of the protocol it follows.

SAML
- Security Assertion Markup Language is an open standard that allows clients to share security info about identity, authentication and permission across different systems.
- SAML is implemented with Extensible Markup Lang (XML) standard for sharing data.
- SAML specifically enables identity federation, making it possible for IdPs to seamlessly and securely pass authenticated identities and their attributes to service providers.

Working of SSO



1. User requests a resource from their desired app
2. App redirects the user to the IdP for auth
3. User signs in which their credentials.
4. IdP sends single sign-on response back to client app
5. App grants access to the user.