AWS notes

Aws IAM USER

* An IAM user is like a specific login for a person or application to access your AWS account
* It's how you manage who can get into your AWS environment

**Authentication**: verifies the identity of a user or service

**Authorization:** determines the permissions and actions they are allowed to perform

**IAM Users:** Created by a DevOps engineer or AWS admin, IAM users are given unique login credentials for accessing AWS resources.

**Policies:** These define permissions and are attached to users, groups, or roles to allow specific actions on AWS resources.

**Groups:** Used to organize users, groups enable the assignment of policies to multiple users at once, simplifying management.

**Roles are used to grant permissions to applications or services (not people)** that need to interact with AWS resources

They provide **temporary security credentials** to the entity assuming the role

**Use case 1:** If an application is running outside of AWS, like on a private cloud and needs to access a database in AWS, you would use a role to grant access

**EC2 (Elastic Cloud Compute)** is a service provided by AWS to run virtual servers in the cloud.

**Explanation**:

* **Elastic**: The ability to scale resources up or down as needed.
* **Cloud**: The infrastructure is provided over the internet.
* **Compute**: Refers to processing power (CPU, RAM, and storage).

**Benefits of EC2**

**Cost Efficiency**:

* Pay only for what you use (pay-as-you-go).
* Shut down servers when not needed to save money.

**Ease of Maintenance**:

* AWS handles updates, security, and uptime.
* No need to manually manage thousands of virtual servers.

**Types of EC2 Instances**

AWS offers different types of instances to match various needs.

1. **General Purpose**: Balanced for compute, memory, and storage.
   * Example: Running basic web applications.
2. **Compute Optimized**: More processing power.
   * Example: Machine learning models or gaming servers.
3. **Memory Optimized**: High memory for data-heavy applications.
   * Example: Big data analytics.
4. **Storage Optimized**: Fast storage performance.
   * Example: Data warehousing.
5. **Accelerated Compute**: Specialized hardware for intensive tasks.
   * Example: AI and graphics rendering

**Regions and Availability Zones**

**Regions**: Geographical locations where AWS has data centers.  
**Availability Zones**: Multiple isolated locations within a region.

**Inbound and Outbound Rules**

**Inbound Rules**

* Control incoming traffic to an instance.
* Define which IPs, protocols, and ports can send requests**.**

**Example Inbound Rules:**

* Allow SSH (TCP, Port 22) from your IP to connect to a server.
* Allow HTTP (TCP, Port 80) from all IPs (0.0.0.0/0) to serve a web page**.**

**Outbound Rules**

* Control outgoing traffic from an instance.
* Define what connections the instance can initiate to other resources or the internet**.**

**Example Outbound Rules:**

* Allow all traffic (default setting) for instances to access the internet or other services.
* Allow traffic on Port 443 (HTTPS) to access secure websites**.**

**Interview question**

**How would you design a VPC architecture for a two-tier application that needs to be highly available and scalable?**

Answer:

* Use multiple availability zones for high availability by placing instances in at least two zones.
* Use Auto Scaling Groups for scalability to handle increased load.
* Create a public subnet for the load balancer and a private subnet for the application servers**.**

**How can you restrict outbound internet access for resources in one subnet while allowing it for another?**

**AWS S3 Buckets**

**Simple Storage Service**

* AWS S3 is a fundamental storage service that is easy to use
* S3's characteristics

1. Scalability
2. Availability
3. Durability
4. Security
5. cost efficiency
6. performance

* S3 buckets and objects are globally accessible but are tied to an AWS region
* Bucket policies and permissions allow fine-grained control over data access.
* S3 is suitable for a variety of use cases from general file storage to hosting static websites

**1. Scalability**

* **Key Point**: S3 offers virtually unlimited storage capacity.
* One object should not be more than 5 TB
* **Practical Use**: Businesses can store from a few gigabytes to exabytes of data without worrying about upfront provisioning.

**2. Availability and Durability**

* **Durability of 11 nines (99.999999999%)**: Ensures that the probability of data loss is extremely low.
* **Availability**: 99.9% uptime for S3 Standard. Redundancy across multiple Availability Zones protects against regional failures.

**3. Security**

* **Encryption Options**: Supports server-side encryption with S3-managed keys (SSE-S3), AWS KMS-managed keys (SSE-KMS), and client-side encryption.
* **Access Management**: IAM policies, bucket policies, and ACLs provide fine-grained control over access.
* **Logging and Monitoring**: Enable detailed logs using Amazon S3 Server Access Logging or AWS CloudTrail.

**4. Cost Efficiency**

* **Storage Classes**:
  + S3 Standard: Frequently accessed data.
  + S3 Intelligent-Tiering: Optimizes costs automatically for unknown or changing access patterns.
  + S3 Glacier & Glacier Deep Archive: Ideal for long-term data archiving at low costs.
* **Pricing Considerations**: Costs are influenced by storage, data transfer, and API request usage.

**5. Performance**

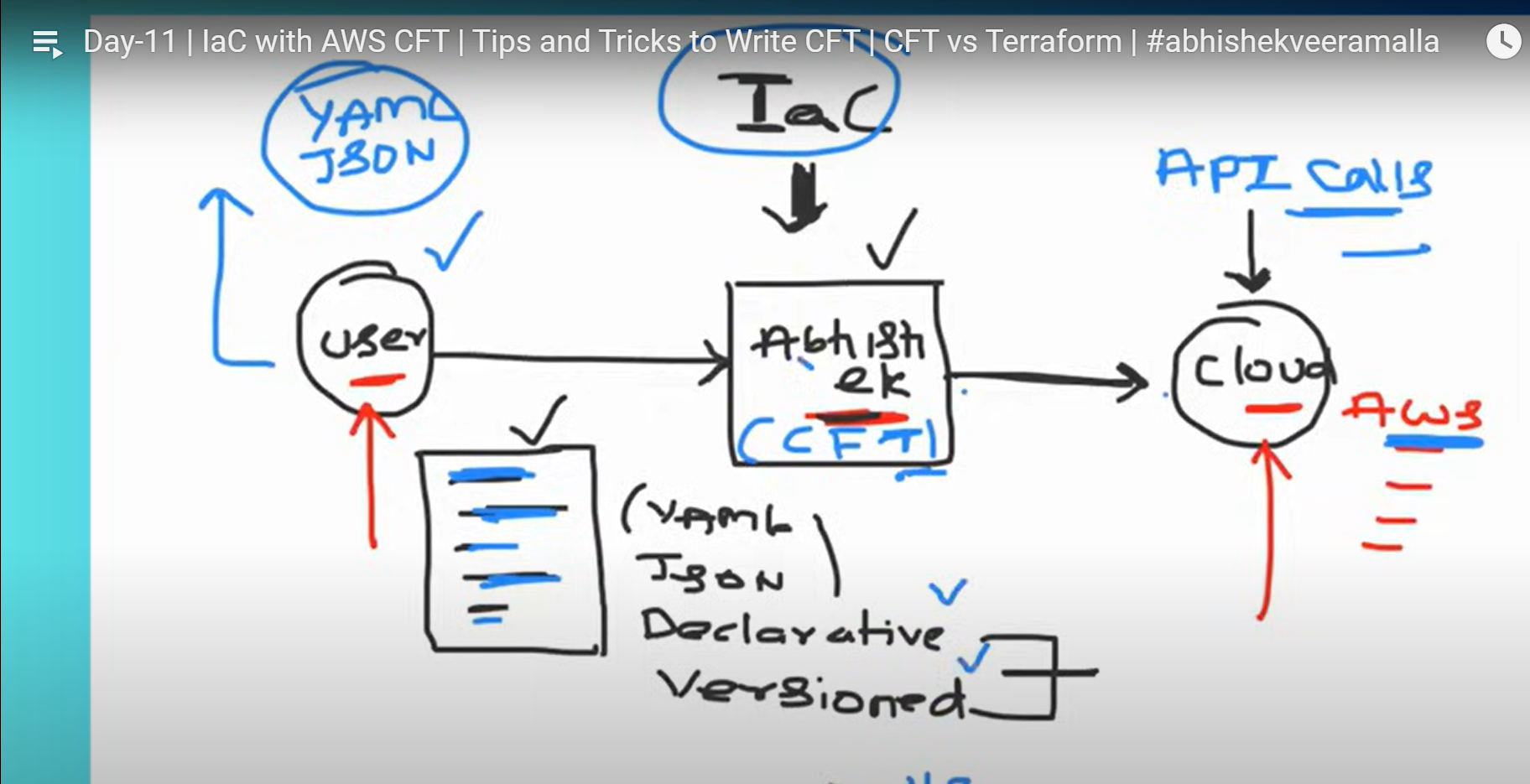
* **Low Latency and High Throughput**: By choosing the closest AWS region, users minimize latency.
* **Features**:
  + **Multi-part upload**: Improves performance when uploading large objects.
  + **Transfer Acceleration**: Uses Amazon’s CloudFront edge locations to speed up uploads.

**AWS CloudFormation Templates (CFT)**

CloudFormation Templates (CFT) are a part of AWS CloudFormation, an **Infrastructure as Code** (IaC) service

that allows users to define and manage AWS infrastructure resources using code**.**

CFT uses declarative templates written in **YAML** or **JSON** to describe the desired state of resources, such as **EC2 instances**, **S3 buckets**,



**Key Principles of IaC:**

1. **Declarative Templates**:
   * The code describes **what resources to create** (like VPCs, EC2 instances, and load balancers) instead of detailing step-by-step how to create them.
   * Looking at a template, you should instantly know which resources it builds.
2. **Version Control**:
   * IaC templates are **saved in version control systems** like Git or S3 buckets.
   * This helps track changes, review updates, and undo mistakes

**IaC vs. AWS CLI**:

* **IaC Tools (like CloudFormation)**:
  + **Automates** complex deployments.
  + Uses **templates** and follows declarative and version control principles.
* **AWS CLI**:
  + Good for **quick commands** but not ideal for managing large setups.
  + It doesn’t follow IaC principles like versioning templates or declarative structure.

**Key features of CFT:**

* **Automates provisioning**: Automatically creates, updates, and deletes AWS resources.
* **Reusability**: Templates can be reused to deploy similar environments.
* **Drift detection**: Identifies changes made outside of CloudFormation to maintain consistency.
* **Stack management**: Resources are grouped into stacks, making it easier to manage related components.

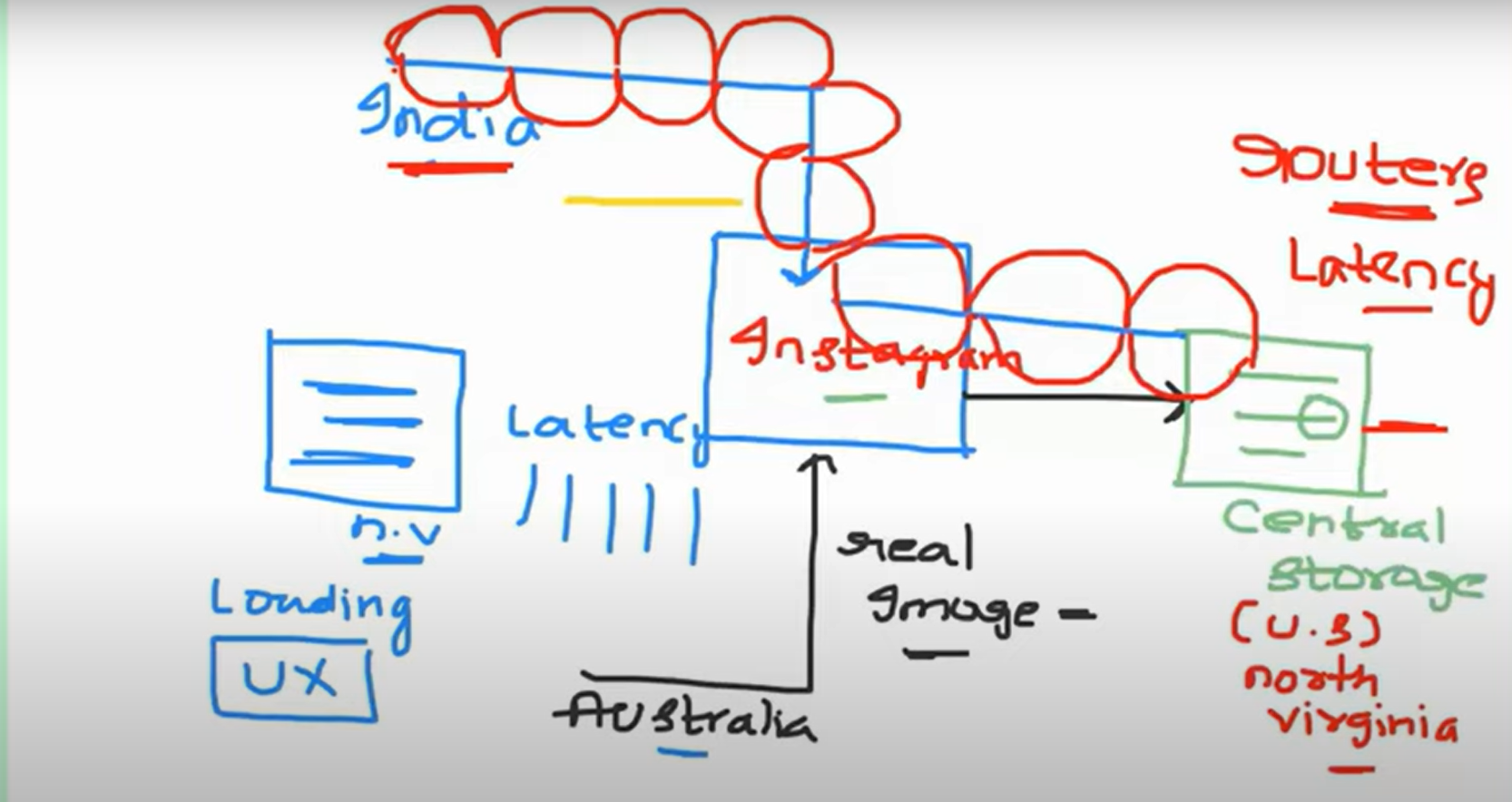
**Key Components of a CloudFormation Template (Simplified)**

CloudFormation templates are made up of different parts. Some are required, while others are optional, giving flexibility in how you create them.

1. Template Format Version:
   * Specifies the version of the template (standard since 2010).
2. Description:
   * A brief explanation of what the template does.
3. Metadata:
   * Information like the author or team responsible for the template (optional).
4. Parameters:
   * Variables that allow users to input values when running the template, making it reusable (e.g., an AMI ID).
5. Rules:
   * Checks to make sure parameter values meet specific requirements (e.g., naming rules or valid instance types).
6. Mappings:
   * Used to map keys to values for specific configurations.
7. Conditions:
   * Specifies when certain resources should be created (e.g., only for a specific environment).
8. Resources:
   * The only required section. It defines the AWS resources (like EC2 or S3) to be created. Each resource has a name, type, and properties.
9. Outputs:
   * Provides values after creating resources (like an instance ID or public IP).

**AWS CloudFront & S3 Static Website Hosting**

* CloudFront is a **Content Delivery Network** (CDN) offered by AWS
* A CDN solves the problem of **latency** when accessing content over the internet
* Without a CDN, requests must travel through **multiple routers (hops)** to reach the central storage location3. This increases latency, especially for users far from the central server
* CDNs work by **caching content in multiple edge locations** around the world2.
* When users request content, it is served from the **nearest edge location** instead of the central storage2.
* This **reduces latency** and improves the user experience



**How CDNs work (using Instagram as an example):**

* When someone uploads an image to Instagram, it's initially stored in a central storage system3.
* Without a CDN, if a user in India wants to see that image, their request would travel to the central storage location (e.g., in the US) and back3. This takes time3.
* With a CDN, a copy of the image is also stored in edge locations4.
* If the user in India requests the image, the CDN will provide it from a local copy within India4.
* This is much faster than retrieving it from the central server

**CloudFront in AWS:**

* CloudFront is the CDN service offered by AWS2.
* It can be used with S3 buckets to host static websites
* **S3 buckets** are object-based storage for static files like images, videos, audio, and HTML5.
* **Best practice is to use CloudFront** instead of directly accessing the S3 bucket
* Direct access is not secure
* Direct access can lead to higher latency
* Direct access is more expensive

**ECR (Elastic Container Registry)**