Various Scenario based questions on S3:part 1

🔹 **1. What happens if you upload an object to a bucket without specifying a storage class?**

- Default storage class is **S3 Standard** ✓
- It provides **high durability (99.999999999%, 11 9's)**, **availability (99.99%)**, and **low latency** ✓
- Ideal for frequently accessed data ✓
- Data is stored redundantly across multiple AZs in the same region ✓

🔹 **2. What steps would you take if your application is seeing slow download speeds from S3?**

- Use **S3 Transfer Acceleration** to speed up cross-region transfers ✓
- Enable **CloudFront CDN** for faster global delivery ✓
- Verify the bucket is in the right **Region** closest to your users ✓
- Check **network latency** and increase **download concurrency** ✓
- Consider **multipart downloads** for large files ✓
- Use **S3 byte-range fetches** for parallel downloads of specific portions of files ✓
- Consider using **S3 Select** to retrieve only needed portions of objects ✓

What it means:

S3 Select lets you **retrieve only specific parts** of an object stored in S3, instead of downloading the entire object. This saves time, bandwidth, and cost, especially when dealing with large files.

Example:

Imagine you have a large CSV file (say, 1 GB) in S3 containing millions of sales records with columns like:

Date	ProductID	Region	SalesAmount	CustomerID
2025-01-01	1001	US	200	C123
2025-01-02	1002	UK	150	C124
...	...	...	...	...

Now, you only want to get the **total sales amount for the US region** without downloading the entire CSV.

Without S3 Select:

- You would have to download the full 1 GB CSV file.
- Then process it locally to filter and sum sales for the US region.

With S3 Select:

- You send a SQL-like query to S3, e.g.:

```
sql
CopyEdit
SELECT SUM(SalesAmount) FROM S3Object WHERE Region = 'US'
```

- S3 processes the query **on the server side**, scans only the needed data inside the file.
- It returns **just the result** (the total sales for US), which is much smaller.

Benefits:

- Faster response time.
- Reduced data transfer costs.
- Lower memory and processing load on your app.

🔹 3. How would you design an S3 solution for millions of users uploading files simultaneously?

- Use **pre-signed URLs** for secure direct uploads ✓
- Implement **multipart uploads** for large files and reliability ✓
- Store uploads with **randomized key prefixes** to avoid performance **hotspots** (not just unique folders) ✓
- Enable **event notifications** to Lambda, SQS, or SNS for post-processing ✓
- Apply **lifecycle policies** for automated storage management ✓
- Consider using **S3 Transfer Acceleration** for faster uploads ✓
- Implement client-side **retry logic** with exponential backoff ✓

🔹 4. How would you integrate S3 with a CI/CD pipeline?

- Store build artifacts in S3 ✓
- Use **S3 as a deployment target** (e.g., for static sites, config files) ✓
- Automate via tools like **GitHub Actions**, **Jenkins**, or **AWS CodePipeline** ✓
- Use **versioning** to manage releases and enable rollbacks ✓
- Implement **post-deployment validation** checks ✓
- Configure **cache invalidation** for CloudFront when deploying updates ✓

🔹 5. How do you manage S3 buckets using Terraform / CloudFormation?

- Define buckets in **Terraform HCL** or **CloudFormation YAML/JSON** ✓
- Specify **versioning**, **encryption**, **lifecycle rules**, and **policies** ✓
- Use **state management** in Terraform for tracking changes ✓
- Apply **IaC best practices** for repeatability and automation ✓
- Implement **CORS configuration** when needed ✓
- Use **parameter stores** or **secret management** for sensitive values ✓
- Include **tagging strategies** for resource organization and cost allocation ✓

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-s3-bucket-name-12345"

  tags = {
    Name        = "MyBucket"
    Environment = "Dev"
  }
}

resource "aws_s3_bucket_versioning" "versioning" {
  bucket = aws_s3_bucket.my_bucket.id

  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket_public_access_block" "public_access" {
  bucket = aws_s3_bucket.my_bucket.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}
```

What AWS feature helps you securely allow users to upload files directly to S3?

A. IAM Roles
B. Pre-signed URLs
C. S3 Event Notification
D. CloudTrail

✅ **Answer: B. Pre-signed URLs**

A pre-signed URL in Amazon S3 is a **secure, time-limited URL** that gives temporary access to a private object in a bucket.

🔐 Key Points about Pre-signed URLs:

- **Purpose:** Allows anyone (even without AWS credentials) to **upload, download, or view** an object in S3 **temporarily**.
- **Who generates it:** You (the bucket owner or someone with permissions) generate it using AWS SDK, CLI, or programmatically via code.
- **Expiration time:** You specify how long the link is valid (e.g., 5 minutes, 1 hour).
- **Security:** Includes authentication info in the URL, so only the holder of the URL can access the object for the limited time.

📦 Use Cases:

- Allowing **users to download a private file** (e.g., invoices, reports).
- Allowing **temporary upload** of files without giving full S3 access.
- **Mobile or web apps** letting users upload images or files directly to S3.

In a CI/CD pipeline, what is a common use of S3?

A. Database Hosting
B. Log Analysis
C. Storing Build Artifacts
D. Managing EC2 Instances

✅ **Answer: C. Storing Build Artifacts**

What is the first step to fix a publicly accessible S3 bucket?

A. Enable versioning
B. Delete the bucket
C. Revoke public access and remove bucket policy
D. Enable CloudWatch logging

✅ **Answer: C. Revoke public access and remove bucket policy**
