1.Design the Architecture

Goal: EC2 hosts the web app; RDS stores user data; S3 stores files; Lambda generates expiring links; IAM manages permissions.

- **VPC:** Create a VPC with:
  - Public Subnet → EC2 instance (web app)
  - Private Subnet → RDS database
- **Internet Gateway:** Attach to VPC for EC2 internet access.
- **NAT Gateway:** Optional if Lambda in private subnet needs internet.
- **Security Groups:** Allow:
  - HTTP/HTTPS to EC2
  - MySQL/Postgres to RDS from EC2 only

2.Set Up IAM Roles & Policies

Roles needed:

1. **EC2 Role**
   - Access S3: `s3:GetObject`, `s3:PutObject`, `s3:ListBucket`
   - Access RDS: If using RDS IAM auth
   - Attach this role to EC2 instance
2. **Lambda Role**
   - Access S3 for generating expiring links
   - Access SES/SNS to send emails
   - Optional: CloudWatch Logs

Step:

- Go to IAM → Roles → Create Role → EC2/Lambda
- Attach policies like `AmazonS3FullAccess` (or custom) and `AmazonRDSFullAccess` (or custom)
- Save role

3.Launch EC2 Instance

Steps:

1. Launch EC2 in **public subnet**.
2. Choose OS (Ubuntu/ Amazon Linux).

3. Assign **EC2 IAM Role** from step 2.
4. Add **EBS volume** for persistent storage (optional, for large files or logs).
5. Configure **User Data** script to install:
    - Web server (Apache/Nginx)
    - PHP/Python/Node (depends on your app)
    - AWS CLI & SDK

Example User Data Script (Ubuntu + Python Flask):

```
#!/bin/bash
sudo apt update -y
sudo apt install python3-pip python3-venv -y
sudo pip3 install flask boto3
```

4.Set Up RDS Database

Steps:

1. Launch RDS instance in **private subnet**.
2. Choose engine (MySQL/PostgreSQL).
3. Set username/password and database name.
4. Configure **security group** to allow EC2 to access RDS.
5. Optionally enable **automatic backups**.

Database Usage:

- Store **user credentials** and **file metadata** (filename, uploader, S3 path, expiration date).

5.Configure S3 Bucket

Steps:

1. Create S3 bucket (e.g., `my-file-storage`).
2. Configure bucket policy to allow EC2/Lambda access only.
3. Enable versioning and encryption if needed.
4. (Optional) Enable S3 lifecycle rules for cleanup of old files.

6.Develop the Web App

Features to implement:

- **Upload:** EC2 server uploads files to S3 via boto3/SDK.
- **Download:** Web app requests pre-signed URL from Lambda or EC2.
- **Metadata:** Store file info in RDS.
- **Expiring Links:** Generate via boto3 or Lambda:

Python Example (Generate Pre-Signed URL):

```
import boto3
s3_client = boto3.client('s3')
url = s3_client.generate_presigned_url(
    'get_object',
    Params={'Bucket': 'my-file-storage', 'Key': 'file.txt'},
    ExpiresIn=3600
)
print(url)
```

7.Set Up Lambda for Email Links

Steps:

1. Create Lambda function with **Lambda IAM Role**.
2. Lambda triggers:
    - Could be invoked via API Gateway when user requests download link.
3. Lambda generates pre-signed URL and sends email via SES/SNS:
    - SES for email
    - SNS for notifications
4. Test function using sample S3 key.

8.Security & Networking

- **Security Groups:**
    - EC2: Allow HTTP/HTTPS from internet
    - RDS: Allow MySQL from EC2 only
- **VPC Endpoint for S3 (optional):**
    - Ensures traffic from EC2/Lambda to S3 stays in AWS network
- **IAM Least Privilege:**
    - Only allow access needed for EC2/Lambda.