Project Title: *Data Backup and Recovery System*

1 Overview of Backup and Recovery Implementation

This phase focuses on deploying a reliable data backup and recovery system leveraging IBM Cloud Object Storage. The goal is to ensure data durability, accessibility, and recovery in case of disasters or data corruption. This implementation involves creating secure storage buckets, automating backups, and developing interfaces for seamless recovery and user interaction.

2 Configuring IBM Cloud Object Storage

2.1 Steps to Set Up Object Storage

1. Create and Configure Buckets:

Step 1: Log in to the IBM Cloud Dashboard

- Visit the IBM Cloud website and sign in using your credentials.
- If you don't have an account, create one by following the registration process.

Step 2: Navigate to the "Object Storage" Section

- After logging in, locate the Search bar at the top.
- Search for Object Storage and click on the resulting service.
- If Object Storage is not yet provisioned in your account, click Create Resource and choose "Cloud Object Storage."
- Select the appropriate Region and Resource Group to proceed.

Step 3: Create a Storage Bucket

- Inside the Object Storage service dashboard, locate the Buckets tab and click Create Bucket.
- Provide a unique name for the bucket. Bucket names must be globally unique across all IBM Cloud users.
- Choose a storage class based on your data retention and access requirements:
    - Standard: Best for frequently accessed data.
    - Vault: Designed for data that is accessed less frequently but needs to be stored longer.
    - Cold Vault: Ideal for archival data that is rarely accessed.
- Select the desired location for your data (Regional, Cross-Regional, or Single-Site).

Step 4: Apply Access Control and Encryption Settings

- Access Control:
    - Use IBM Cloud IAM (Identity and Access Management) to assign roles to users or service IDs.
    - Restrict public access unless explicitly required. Buckets should ideally remain private for security reasons.
    - Enable fine-grained policies, such as read-only or write-only access for specific users or applications.

- Encryption Settings:
    - Enable Server-Side Encryption (SSE) to ensure all objects stored in the bucket are encrypted automatically.
    - You can either use IBM-managed keys or your own encryption keys via IBM Key Protect or Hyper Protect Crypto Services for enhanced security.

2.2 Integrating Backup Scripts

Step 1: Use Python with the ibm_boto3 Library to Automate Backups

The ibm_boto3 library allows you to interact programmatically with IBM Cloud Object Storage for operations such as uploading, downloading, and listing objects.

Prerequisites:

- Install the ibm_boto3 library:

pip install ibm-cos-sdk

- Obtain your IBM Cloud Object Storage credentials:
    - Go to the IBM Cloud dashboard.
    - Select Service Credentials under the Object Storage instance.
    - Click Create Credential and copy the generated API key, access key, and other details.

Sample Code:

```
import ibm_boto3
from ibm_botocore.client import Config, ClientError


# Configuration
cos = ibm_boto3.client('s3',
                       ibm_api_key_id='your-api-key',
                       ibm_service_instance_id='your-service-instance-id',
                       config=Config(signature_version='oauth'),
                       endpoint_url='https://s3.<region>.cloud-object-storage.appdomain.cloud')
def backup_file(file_path, bucket_name, object_name):
    """
    Backs up a file to IBM Cloud Object Storage.
    :param file_path: Path to the file to back up.
    :param bucket_name: Name of the bucket.
    :param object_name: Name of the object to create in the bucket.
    """
    try:
```

```
        # Upload the file
        cos.upload_file(Filename=file_path, Bucket=bucket_name, Key=object_name)
        print(f"File '{file_path}' successfully uploaded to bucket '{bucket_name}' as '{object_name}'.")
    except ClientError as e:
        print(f"Error uploading file: {e}")

# Usage
file_path = "local_data.csv"  # Local file path
bucket_name = "my-storage-bucket"  # Name of the bucket
object_name = "backup_data/local_data.csv"  # Object name in the bucket
backup_file(file_path, bucket_name, object_name)
```

3.**Scheduling Backups**:

- Utilize tools like cron (Linux) or Task Scheduler (Windows) for periodic execution.

4.Recovery System Development

REST API for Recovery

Using Flask, we create a REST API to handle data retrieval from Object Storage.

Sample API Code:

```
from flask import Flask, jsonify
import ibm_boto3
app = Flask(__name__)
# Initialize the COS client
cos = ibm_boto3.client('s3', ibm_api_key_id='your-api-key', ...)
@app.route('/restore/<bucket_name>/<file_name>', methods=['GET'])
def restore_file(bucket_name, file_name):
    try:
        cos.download_file(bucket_name, file_name, f'restored_{file_name}')
        return jsonify({"message": f"File {file_name} restored successfully."})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
if __name__ == '__main__':
    app.run(debug=True)
```

5. User Interface Development

Building a Recovery Dashboard

1. **Using Streamlit for a Simple UI**:
    - Streamlit allows users to view available backups and restore files interactively.

Streamlit Code:

```
import streamlit as st

import requests

st.title("Data Backup and Recovery System")

# Input for bucket and file name

bucket = st.text_input("Enter Bucket Name", "my-bucket")

file_name = st.text_input("Enter File Name to Restore", "data_backup.csv")

# Restore Button

if st.button("Restore File"):

    response = requests.get(f"http://localhost:5000/restore/{bucket}/{file_name}")

    st.write(response.json())
```

2. **Advanced Interface with React**:
    - React can provide a dynamic and responsive UI for viewing storage contents and triggering backups or restores.

6. IBM Cloud Platform Features and Considerations

- **Scalability**: IBM Cloud Object Storage supports scaling to accommodate large datasets.
- **Security**: Enable IAM policies and encryption for sensitive data.
- **Monitoring**: Use IBM Cloud Monitoring to track API calls and storage usage.
- **Cost Efficiency**: Optimize storage classes to minimize costs.

Feature	Benefits	Best Practices
Scalability	Handles large datasets with automatic scaling.	Monitor usage and plan for proactive scaling during peak loads.
Security	Protects sensitive data with IAM, encryption, and access policies.	Use least privilege policies, enable encryption, and enforce MFA for admin roles.
Monitoring	Provides insights into storage operations and alerts for anomalies.	Set thresholds for critical metrics and use dashboards for trend analysis.
Cost Efficiency	Reduces expenses through optimal storage class selection and lifecycle policies.	Analyze access patterns and configure automated transitions to lower-cost classes.

7. Conclusion

This phase integrates IBM Cloud Object Storage for a robust data backup and recovery system. The system ensures data protection and easy retrieval through automation and user-friendly interfaces. By

leveraging IBM Cloud's scalable and secure platform, the project is ready for deployment in production environments.

8. Further Enhancement

Automated Backup Scheduling: Implement an automated scheduling system that backs up data at regular intervals (e.g., daily, weekly). You can use IBM Cloud Functions or a cron job to trigger backups at specified times.

Backup Encryption: Encrypt your backup files before uploading them to IBM Cloud Object Storage for enhanced security, using AES encryption or any other algorithm.

Versioning for Backup Files: Enable versioning on the IBM Cloud Object Storage bucket to retain multiple versions of backup files. This ensures that you can restore from any previous backup version.

Backup Health Monitoring: Set up logging or monitoring to track backup status, errors, or success. Use IBM Cloud Monitoring services to alert you if any backups fail.

Data Recovery System: Implement a data recovery process that retrieves the most recent or specific backup file from IBM Cloud Object Storage. Include versioning and date-based recovery logic.

Data Integrity Check: Before restoring data, implement a checksum or hash comparison to ensure the integrity of the backup file.

Recovery Time Objective (RTO) and Recovery Point Objective (RPO): Ensure your system meets both RTO and RPO by fine-tuning the backup intervals and ensuring minimal downtime during recovery.