**Project Title : CONTENT DELIVERY NETWORK WITH IBM CLOUD AKAMAI INTEGRATION**

**1. Integrating Backup Scripts**

Step 1: Understanding the Integration

Provides cloud storage and computing services.

Accelerates content delivery and reduces latency.

Automates the process of backing up data, CDN configurations, or logs..

Prerequisites:

* Install the - ibm\_boto3, requests, akam

**pip install ibm-cos-sdk requests edgegrid-python**

* 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:**

from ibm\_boto3 import client

from ibm\_botocore.client import Config

cos\_client = client(

"s3",

ibm\_api\_key\_id="YOUR\_API\_KEY",

ibm\_service\_instance\_id="YOUR\_INSTANCE\_ID",

config=Config(signature\_version="s3v4"),

endpoint\_url="https://s3.us.cloud-object-storage.appdomain.cloud"

)

def upload\_to\_ibm\_cloud(file\_name, bucket\_name):

try:

cos\_client.upload\_file(file\_name, bucket\_name, file\_name)

print(f"Uploaded {file\_name} to IBM Cloud Storage.")

except Exception as e:

print(f"Error uploading to IBM Cloud: {e}")

Python Script for IBM Cloud & Akamai Integration\*

python

import requests

import json

import hmac

import hashlib

import base64

import time

import ibm\_boto3

from ibm\_botocore.client import Config

IBM Cloud Object Storage Credentials

IBM\_COS\_ENDPOINT = "https://s3.us.cloud-object-storage.appdomain.cloud"

IBM\_COS\_API\_KEY = "your-ibm-cos-api-key"

IBM\_COS\_BUCKET = "your-bucket-name"

IBM\_COS\_RESOURCE\_INSTANCE\_ID = "your-resource-instance-id"

Akamai API Credentials

AKAMAI\_HOST = "https://akab-xxxxxxxxxxx.luna.akamaiapis.net"

AKAMAI\_CLIENT\_TOKEN = "your-client-token"

AKAMAI\_CLIENT\_SECRET = "your-client-secret"

AKAMAI\_ACCESS\_TOKEN = "your-access-token"

AKAMAI\_PURGE\_ENDPOINT = "/ccu/v3/invalidate/url"

Initialize IBM COS Client

cos\_client = ibm\_boto3.client(

"s3",

ibm\_api\_key\_id=IBM\_COS\_API\_KEY,

ibm\_service\_instance\_id=IBM\_COS\_RESOURCE\_INSTANCE\_ID,

config=Config(signature\_version="s3v4"),

endpoint\_url=IBM\_COS\_ENDPOINT

)

Function to Upload File to IBM Cloud Object Storage

def upload\_to\_cos(file\_name, file\_path):

try:

with open(file\_path, "rb") as file\_data:

cos\_client.upload\_fileobj(file\_data, IBM\_COS\_BUCKET, file\_name)

print(f"File '{file\_name}' uploaded successfully to IBM COS.")

except Exception as e:

print(f"Error uploading file: {str(e)}")

Function to Purge Akamai Cache

def purge\_akamai\_cache(urls):

headers = {

"Content-Type": "application/json",

"Authorization": f"EG1-HMAC-SHA256 client\_token={AKAMAI\_CLIENT\_TOKEN};access\_token={AKAMAI\_ACCESS\_TOKEN};timestamp={int(time.time())};nonce=123456;",

}

payload = {

"objects": urls

}

response = requests.post(f"{AKAMAI\_HOST}{AKAMAI\_PURGE\_ENDPOINT}", headers=headers, data=json.dumps(payload))

if response.status\_code == 201:

print("Akamai cache purge request successful.")

else:

print(f"Akamai purge failed: {response.text}")

Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

FILE\_NAME = "sample.jpg"

FILE\_PATH = "./sample.jpg"

Upload file to IBM Cloud

upload\_to\_cos(FILE\_NAME, FILE\_PATH)

Purge Akamai Cache for the new content

CDN\_URL = f"https://cdn.example.com/{FILE\_NAME}"

purge\_akamai\_cache([CDN\_URL])

**2. Recovery System Developmen**

**REST API for Recovery**

Using for monitor the health of your services, trigger failover actions .

**Sample API Code:**

mport requests

import json

IBM Cloud API endpoint for monitoring (example)

monitoring\_url = "https://cloud.ibm.com/v1/monitoring/health"

api\_key = "YOUR\_IBM\_CLOUD\_API\_KEY"

Function to check server health

def check\_server\_health(server\_id):

headers = {

"Authorization": f"Bearer {api\_key}",

"Content-Type": "application/json"

}

response = requests.get(f"{monitoring\_url}/servers/{server\_id}/status", headers=headers)

if response.status\_code == 200:

status = response.json()["status"]

return status

else:

raise Exception("Error checking server health.")

Sample Python Code for Recovery System\*

python

import requests

import json

import time

import ibm\_boto3

from ibm\_botocore.client import Config

IBM Cloud Object Storage Credentials

IBM\_COS\_ENDPOINT = "https://s3.us.cloud-object-storage.appdomain.cloud"

IBM\_COS\_BACKUP\_ENDPOINT = "https://s3.secondary-backup.com" # Failover Storage

IBM\_COS\_API\_KEY = "your-ibm-cos-api-key"

IBM\_COS\_BUCKET = "your-bucket-name"

IBM\_COS\_RESOURCE\_INSTANCE\_ID = "your-resource-instance-id"

Akamai API Credentials

AKAMAI\_HOST = "https://akab-xxxxxxxxxxx.luna.akamaiapis.net"

AKAMAI\_CLIENT\_TOKEN = "your-client-token"

AKAMAI\_CLIENT\_SECRET = "your-client-secret"

AKAMAI\_ACCESS\_TOKEN = "your-access-token"

AKAMAI\_PURGE\_ENDPOINT = "/ccu/v3/invalidate/url"

Initialize IBM COS Clients for primary and backup storage

cos\_client\_primary = ibm\_boto3.client(

"s3",

ibm\_api\_key\_id=IBM\_COS\_API\_KEY,

ibm\_service\_instance\_id=IBM\_COS\_RESOURCE\_INSTANCE\_ID,

config=Config(signature\_version="s3v4"),

endpoint\_url=IBM\_COS\_ENDPOINT

)

cos\_client\_backup = ibm\_boto3.client(

"s3",

ibm\_api\_key\_id=IBM\_COS\_API\_KEY,

ibm\_service\_instance\_id=IBM\_COS\_RESOURCE\_INSTANCE\_ID,

config=Config(signature\_version="s3v4"),

endpoint\_url=IBM\_COS\_BACKUP\_ENDPOINT

)

Function to Check Primary Storage Health

def check\_ibm\_cos\_health():

try:

cos\_client\_primary.list\_buckets()

return True

except Exception:

return False

Function to Sync Content to Backup Storage

def sync\_to\_backup(file\_name):

try:

file\_data = cos\_client\_primary.get\_object(Bucket=IBM\_COS\_BUCKET, Key=file\_name)['Body'].read()

cos\_client\_backup.put\_object(Bucket=IBM\_COS\_BUCKET, Key=file\_name, Body=file\_data)

print(f"Backup created for {file\_name}.")

except Exception as e:

print(f"Backup sync failed: {str(e)}")

Function to Switch to Backup in Case of Failure

def failover(file\_name):

if check\_ibm\_cos\_health():

print("Primary storage is healthy. No failover needed.")

return

print("Primary storage is down! Switching to backup storage...")

backup\_url = f"{IBM\_COS\_BACKUP\_ENDPOINT}/{IBM\_COS\_BUCKET}/{file\_name}"

Akamai Purge Request

purge\_akamai\_cache([backup\_url])

print(f"Traffic redirected to backup storage: {backup\_url}")

Function to Purge Akamai Cache

def purge\_akamai\_cache(urls):

headers = {

"Content-Type": "application/json",

"Authorization": f"EG1-HMAC-SHA256 client\_token={AKAMAI\_CLIENT\_TOKEN};access\_token={AKAMAI\_ACCESS\_TOKEN};timestamp={int(time.time())};nonce=123456;",

}

payload = {

"objects": urls

}

response = requests.post(f"{AKAMAI\_HOST}{AKAMAI\_PURGE\_ENDPOINT}", headers=headers, data=json.dumps(payload))

if response.status\_code == 201:

print("Akamai cache purge request successful.")

else:

print(f"Akamai purge failed: {response.text}")

Function to Monitor and Trigger Recovery

def monitor\_and\_recover(file\_name):

while True:

if not check\_ibm\_cos\_health():

failover(file\_name)

time.sleep(60) # Check every minute

Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

FILE\_NAME = "index.html"

Sync content to backup

sync\_to\_backup(FILE\_NAME)

Monitor and trigger recovery

monitor\_and\_recover(FILE\_NAME)

**3. User Interface Development**

**Building a Recovery Dashboard**

1. **Using Streamlit for a Simple UI**:

* streamlit, requests (for API requests), and json for this implementation.

**Streamlit Code:**

import streamlit as st

import requests

IBM Cloud API configuration

IBM\_API\_KEY = 'YOUR\_IBM\_CLOUD\_API\_KEY'

IBM\_MONITORING\_URL = 'https://cloud.ibm.com/v1/monitoring/health'

Akamai API configuration

AKAMAI\_API\_URL = 'https://akaa-baseurl.akamai.com/ccu/v3/notifications'

AKAMAI\_API\_TOKEN = 'YOUR\_AKAMAI\_API\_TOKEN'

Example backup server URL

BACKUP\_SERVER\_URL = 'https://backup-server.example.com'

Function to check the health of the IBM Cloud server

def check\_server\_health(server\_id):

headers = {

'Authorization': f'Bearer {IBM\_API\_KEY}',

'Content-Type': 'application/json'

}

response = requests.get(f'{IBM\_MONITORING\_URL}/servers/{server\_id}/status', headers=headers)

if response.status\_code == 200:

return response.json()

else:

return None

Function to trigger cache purge on Akamai

def purge\_akamai\_cache():

headers = {

'Authorization': f'Bearer {AKAMAI\_API\_TOKEN}',

'Content-Type': 'application/json'

}

payload = {

"objects": [

"https://yourdomain.com/content-to-purge"

],

"action": "delete"

}

response = requests.post(AKAMAI\_API\_URL, headers=headers, json=payload)

return response.status\_code == 200

Function to reroute traffic to a backup server

def reroute\_traffic():

response = requests.get(f'{BACKUP\_SERVER\_URL}/re-route')

return response.status\_code == 200

Step 1: Backend (Flask)

This backend handles file uploads, cache purging, and monitoring.

Install Dependencies

bash

pip install flask requests ibm-cos-sdk

Flask API (backend.py)

python

from flask import Flask, request, jsonify

import requests

import json

import ibm\_boto3

from ibm\_botocore.client import Config

app = Flask(\_\_name\_\_)

IBM Cloud Object Storage Credentials

IBM\_COS\_ENDPOINT = "https://s3.us.cloud-object-storage.appdomain.cloud"

IBM\_COS\_API\_KEY = "your-ibm-cos-api-key"

IBM\_COS\_BUCKET = "your-bucket-name"

IBM\_COS\_RESOURCE\_INSTANCE\_ID = "your-resource-instance-id"

Akamai API Credentials

AKAMAI\_HOST = "https://akab-xxxxxxxxxxx.luna.akamaiapis.net"

AKAMAI\_CLIENT\_TOKEN = "your-client-token"

AKAMAI\_CLIENT\_SECRET = "your-client-secret"

AKAMAI\_ACCESS\_TOKEN = "your-access-token"

AKAMAI\_PURGE\_ENDPOINT = "/ccu/v3/invalidate/url"

IBM COS Client

cos\_client = ibm\_boto3.client(

"s3",

ibm\_api\_key\_id=IBM\_COS\_API\_KEY,

ibm\_service\_instance\_id=IBM\_COS\_RESOURCE\_INSTANCE\_ID,

config=Config(signature\_version="s3v4"),

endpoint\_url=IBM\_COS\_ENDPOINT

)

Upload file to IBM COS

@app.route("/upload", methods=["POST"])

def upload\_file():

file = request.files["file"]

file\_name = file.filename

try:

cos\_client.upload\_fileobj(file, IBM\_COS\_BUCKET, file\_name)

return jsonify({"message": "File uploaded successfully", "file\_url": f"{IBM\_COS\_ENDPOINT}/{IBM\_COS\_BUCKET}/{file\_name}"})

except Exception as e:

return jsonify({"error": str(e)}), 500

Purge Akamai Cache

@app.route("/purge", methods=["POST"])

def purge\_cache():

urls = request.json.get("urls", [])

headers = {

"Content-Type": "application/json",

"Authorization": f"EG1-HMAC-SHA256 client\_token={AKAMAI\_CLIENT\_TOKEN};access\_token={AKAMAI\_ACCESS\_TOKEN};timestamp=12345678;nonce=123456;",

}

payload = {"objects": urls}

response = requests.post(f"{AKAMAI\_HOST}{AKAMAI\_PURGE\_ENDPOINT}", headers=headers, json=payload)

if response.status\_code == 201:

return jsonify({"message": "Akamai cache purge request successful."})

else:

return jsonify({"error": response.text}), response.status\_code

Get CDN Status

@app.route("/status", methods=["GET"])

def get\_status():

try:

cos\_client.list\_buckets()

cos\_status = "IBM Cloud Storage: Available"

except:

cos\_status = "IBM Cloud Storage: Down"

return jsonify({"cdn\_status": cos\_status})

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

Step 2: Frontend (React.js)

This React app allows users to upload files, purge cache, and check the CDN status.

Install React & Dependencies

bash

npx create-react-app cdn-dashboard

cd cdn-dashboard

npm install axios

React Code (App.js)

javascript

import React, { useState } from "react";

import axios from "axios";

function App() {

const [file, setFile] = useState(null);

const [status, setStatus] = useState("Checking...");

const [purgeUrl, setPurgeUrl] = useState("");

// Handle file selection

const handleFileChange = (event) => {

setFile(event.target.files[0]);

};

// Upload file to IBM Cloud

const handleUpload = async () => {

const formData = new FormData();

formData.append("file", file);

try {

const response = await axios.post("http://localhost:5000/upload", formData);

alert("File uploaded: " + response.data.file\_url);

} catch (error) {

alert("Upload failed: " + error.response.data.error);

}

};

// Purge Akamai Cache

const handlePurge = async () => {

try {

const response = await axios.post("http://localhost:5000/purge", { urls: [purgeUrl] });

alert(response.data.message);

} catch (error) {

alert("Purge failed: " + error.response.data.error);

}

};

// Fetch CDN Status

const fetchStatus = async () => {

try {

const response = await axios.get("http://localhost:5000/status");

setStatus(response.data.cdn\_status);

} catch (error) {

setStatus("Error fetching status.");

}

};

return (

<div style={{ padding: "20px", maxWidth: "500px", margin: "auto" }}>

<h2>CDN Management Dashboard</h2>

<div>

<h3>Upload File</h3>

<input type="file" onChange={handleFileChange} />

<button onClick={handleUpload}>Upload</button>

</div>

<div>

<h3>Purge Cache</h3>

<input

type="text"

placeholder="Enter URL to purge"

value={purgeUrl}

onChange={(e) => setPurgeUrl(e.target.value)}

/>

<button onClick={handlePurge}>Purge</button>

</div>

<div>

<h3>CDN Status</h3>

<button onClick={fetchStatus}>Check Status</button>

<p>{status}</p>

</div>

</div>

);

}

This setup assumes that you are using IBM Cloud Object Storage and want to serve your content through Akamai's CDN.

Steps:

1.IBM Cloud Object Storage: Store your static assets.

2. Akamai Configuration: Set up an edge hostname and caching rules.

3. IBM Cloud Load Balancer (Optional): Improve redundancy.

4. DNS Configuration: Point your domain to Akamai.

Here’s a sample configuration script using Terraform and Akamai’s APIs:

hcl

provider "akamai" {

edgerc = "~/.edgerc"

section = "default"

}

resource "akamai\_property" "cdn\_property" {

name = "ibm-cloud-cdn"

contract\_id = "ctr\_ABC123"

group\_id = "grp\_12345"

product\_id = "prd\_Fresca"

rule\_format = "latest"

hostnames {

cname\_from = "cdn.example.com"

cname\_to = "example.edgesuite.net"

}

rules {

name = "default"

type = "tree"

rules = [

{

name = "Origin"

behaviors = [

{

name = "origin"

options = {

hostname = "s3.us.cloud-object-storage.appdomain.cloud"

originType = "CUSTOMER"

}

}

]

}

]

}

}