



Talent Exhibition Project: Multi-Cloud File Storage



ARISTIDE KATAGARUKA

BTS CLOUD COMPUTING
LYCÉE GUILLAME KROLL

SUBMISSION DATE: JULY 2ND 2025

Contents

Certification and Declaration of Honor	3
1. Overview.....	4
2. Features	4
3. Project Structure	5
4. Development Environment Setup (3 hours, June 3, 2025)	6
Objective:	6
Steps	6
Troubleshooting	10
5. Cloud Account Setup (2 hours, June 3, 2025).....	10
Objective:	10
Steps	10
Troubleshooting	23
6. Web App Frontend Development (8 hours, June 5, 2025)	23
Objective:	23
Steps	23
Troubleshooting	31
7. Cloud Storage API Integration (10 hours, June 7, 2025)	31
Objective:	31
Steps	31
Troubleshooting	37
8. Application Containerization (6 hours, June 9, 2025)	37
Objective:	37
Steps	37
Troubleshooting	40
9. Multi-Cloud Simulation with Load Balancing (6 hours, June 10, 2025).....	40
Objective:	40
Steps	40
Troubleshooting	42
10. Sustainability and Alerts Implementation (8 hours, June 12, 2025)	42
Objective:	42
Steps	42
Troubleshooting	58

11. Testing File Operations and Demo Scenarios (6 hours, June 15, 2025).....	59
Objective:	59
Steps	59
Troubleshooting	60
12. Performance Optimization	61
Objective:	61
Steps	61
Troubleshooting	63
13. Documentation and Presentation (9 hours, June 17, 2025).....	64
Objective:	64
Steps	64
14. Technical Architecture	67
15. Key Artifacts	68
Flask Backend (backend/app.py)	68
16. Video of the project overview.....	70
17. Project Management.....	70
18. Conclusion	71
19. Learning Resources	71
19. Resource Links for Multi-Cloud Storage System Project.....	71
20. Table of Figures	73

Certification and Declaration of Honor

I, the undersigned, hereby declare that the work contained in this project report titled "**Multi-Cloud File Storage System**" is my own and has been completed as part of the PROMA and TBUCO course requirements for the BTS Cloud Computing program at Lycée Guillaume Kroll.

I affirm that:

1. I have conducted the project ethically and honestly, adhering to the principles of integrity and academic conduct.
2. All external references, sources, and contributions by others have been appropriately acknowledged in the report.
3. The report has not been plagiarized, and no part of this work has been submitted previously for any other assignment, certification, or purpose.

Furthermore, I acknowledge that failure to comply with the rules of academic honesty may result in disqualification of this project and the revocation of grades awarded for the PROMA and TBUCO course.

Signatures of Team Members:

- Aristide KATAGARUKA



Date: 2/07/2025

1. Overview

This project implements a multi-cloud file storage system leveraging free-tier services from Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). The system is designed to demonstrate cost optimization and sustainability for businesses while serving as a zero-budget prototype for a talent exhibition. It includes a React-based frontend, a Flask backend, Dockerized environments, and Nginx for load balancing, with features like file uploads, sustainability metrics, and API usage alerts.

2. Features

- **File Operations:** Upload/download files across AWS S3, Azure Blob Storage, and GCP Cloud Storage.
 - **Batch Upload:** Support for uploading multiple files simultaneously.
 - **Demo Scenarios:** Mock object (backup), block (VM storage), and file (sharing) storage use cases.
 - **Dashboard:** Displays storage usage, simulated cost savings, API limit alerts, and carbon footprint.
 - **Provider Comparison:** Table comparing pricing, free-tier limits, and egress fees.
 - **Storage Info:** Technical details and use cases for object, block, and file storage.
 - **Sustainability:** Energy estimation (0.01 kWh/GB), lifecycle policies (Glacier, Archive, Coldline).
 - **Alerts:** Notifications for API usage limits (80% of free-tier thresholds).
 - **Deployment:** Containerized with Docker, orchestrated with Docker Compose, load-balanced with Nginx.
-

3. Project Structure

```

Multi-Cloud-Storage/
    ├── backend/
    │   ├── app.py                  # Flask backend main application
    │   ├── file_upload.py          # Cloud storage upload logic
    │   ├── energy_estimator.py    # Energy usage calculation
    │   ├── requirements.txt        # Python dependencies
    │   ├── temp/                   # Temporary file storage
    │   └── key.json                # GCP service account key
    ├── docs/
    │   ├── business-case.md       # Business case documentation
    │   ├── technical-docs.md      # Technical documentation
    │   └── user-guide.md          # User guide for setup and usage
    ├── storage-app/
    │   ├── package.json           # React app config and dependencies
    │   ├── Dockerfile              # Dockerfile for frontend
    │   ├── tailwind.config.js     # Tailwind CSS configuration
    │   └── src/
    │       ├── components/
    │       │   ├── FileUpload.js    # File upload UI and logic
    │       │   └── StorageDashboard.js # Dashboard with usage, savings,
    │       └── StorageInfo.js       # Info and use cases for storage
    ├── alerts
    ├── types
    │   ├── ProviderComparison.js # Cloud provider comparison table
    │   └── pages/
    │       └── Home.js            # Main page composing dashboard and
    ├── components
    │   ├── services/
    │   │   └── api.js             # API call utilities
    │   ├── App.js                 # App entry point
    │   ├── index.js               # React entry point
    │   └── index.css              # Tailwind CSS styles
    └── docker-compose.yml        # Orchestration for backend, frontend,
                                 # Redis, Nginx
    └── nginx.conf                # Nginx load balancer configuration
    └── .gitignore                 # Git ignore rules

```

4. Development Environment Setup (3 hours, June 3, 2025)

Objective: Configure development tools and version control.

Steps

1. Install Visual Studio Code (1 hour):

- Download VS Code from code.visualstudio.com.
- Install extensions (Ctrl+Shift+X):
 - **Python**: Linting, debugging, IntelliSense.
 - **JavaScript (ES6) snippets**: React development.
 - **Docker**: Dockerfile and Docker Compose support.
 - **GitLens**: Git commit history and blame.
- Configure Python interpreter: Ctrl+Shift+P > “Python: Select Interpreter” > Python 3.9+.

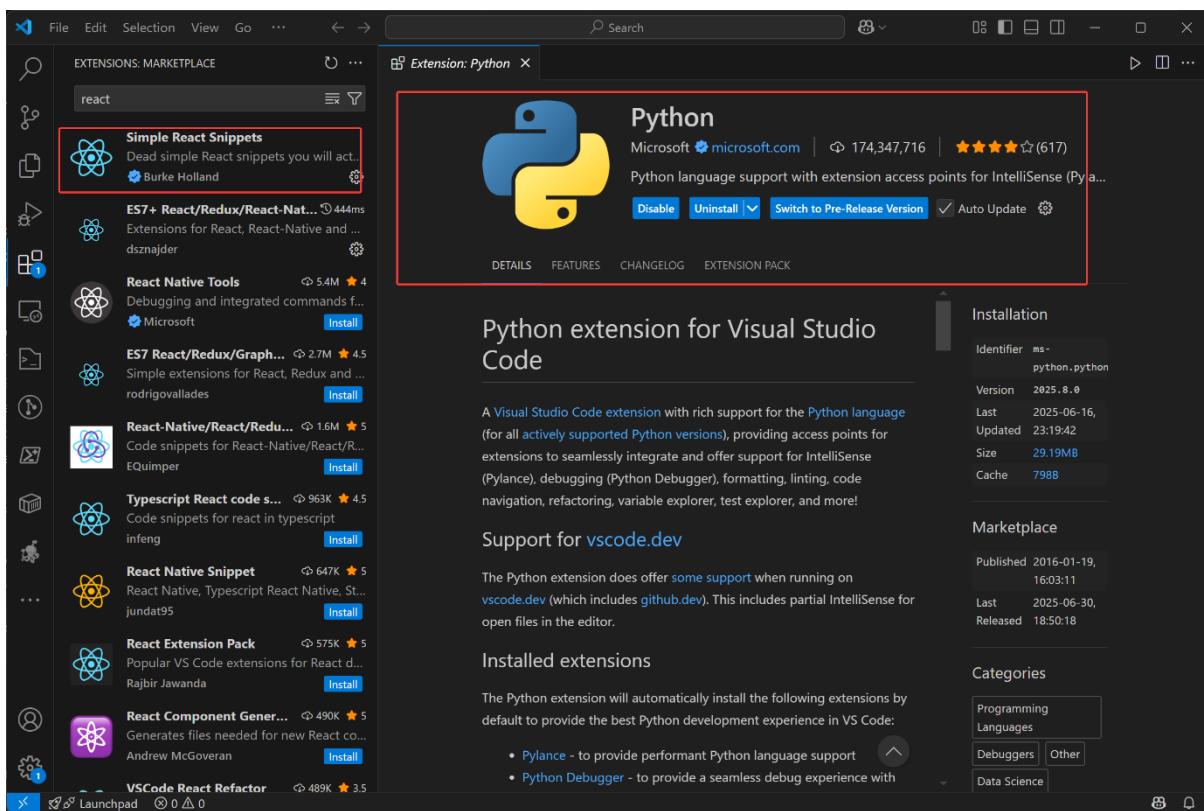


Figure 1: VS Code Extensions panel with installed extensions, Python file open

- **Why:** VS Code provides a lightweight, extensible IDE for Python and React.

2. Install Python and Node.js

- Download Python 3.9+ from python.org (e.g., 3.9.13).

- Windows: Check “Add Python to PATH.”
- macOS/Linux: Use brew install python or sudo apt install python3.
- Download Node.js 16+ (LTS) from nodejs.org.

- Verify:

```
bash

python --version # Check Python version
node --version    # Check Node.js version
npm --version     # Check npm version
```

Figure 2: Verifying versions

- Expected: Python 3.9.x, Node.js v16.x.x, npm 8.x.x.

```
PS C:\Users\akata\Desktop\Talent_Exhibition_Project_Aristide> python --version # Check Python version
Python 3.13.5
PS C:\Users\akata\Desktop\Talent_Exhibition_Project_Aristide> node --version      # Check Node.js version
v22.16.0
PS C:\Users\akata\Desktop\Talent_Exhibition_Project_Aristide> npm --version       # Check npm version
10.9.2
```

Figure 3: Terminal showing version outputs

- Why: Python for Flask backend, Node.js for React frontend.

3. Set Up React Project:

- Create React app:
- npx create-react-app storage-app # Initialize React project
- cd storage-app # Navigate to project directory
- npm install tailwindcss postcss autoprefixer axios # Install dependencies
- npx tailwindcss init -p # Initialize Tailwind CSS with PostCSS

- Update storage-app/tailwind.config.js:
- module.exports = { // Export Tailwind configuration
- content: ["./src/**/*.{js,jsx,ts,tsx}"], // Specify files for Tailwind to scan
- theme: { extend: {} }, // Extend Tailwind theme (empty for now)
- plugins: [], // Add Tailwind plugins (none for now)
- };

- Update storage-app/src/index.css:
- @tailwind base; /* Apply Tailwind base styles */
- @tailwind components; /* Apply Tailwind component styles */
- @tailwind utilities; /* Apply Tailwind utility classes */

- Add proxy to storage-app/package.json for backend API calls:

```
"proxy": "http://localhost:5000" // Proxy API requests to
Flask backend
```

- Start app:

```
npm start # Start development server
```



Figure 4: Browser showing default React app with Tailwind styling

- Verify at <http://localhost:3000>.
- Why: React for dynamic UI, Tailwind for responsive styling, axios for API calls.

4. Configure GitHub Repository:

- Create repository multi-cloud-storage on github.com.
- Initialize Git:
 - git init # Initialize Git repository
 - git add . # Add all files to staging
 - git commit -m "Initial commit with React setup" # Commit changes
 - git remote add origin <repository-url> # Set remote repository URL
 - git push -u origin main # Push to main branch
- Create .gitignore:
 - node_modules # Ignore Node.js dependencies
 - build # Ignore React build output
 - venv # Ignore Python virtual environment
 - __pycache__ # Ignore Python bytecode
 - *.pyc # Ignore Python compiled files
 - temp # Ignore temporary files

- backend/key.json # Ignore GCP service account key
backend/.env # Ignore environment variables
- Set up GitHub Actions in .github/workflows/ci.yml:
 - name: CI # Name of the workflow
 - on: # Trigger events
 - push: # Run on push to main branch
 - branches: [main]
 - jobs: # Define jobs
 - build: # Build job
 - runs-on: ubuntu-latest # Run on Ubuntu latest
 - steps: # Define steps
 - uses: actions/checkout@v3 # Checkout code
 - name: Set up Node.js # Step to set up Node.js
 - uses: actions/setup-node@v3
 - with:
 - node-version: '16' # Use Node.js version 16
 - name: Install dependencies # Step to install dependencies
 - run: npm install # Run npm install
 - name: Build # Step to build React app
 - run: npm run build # Run build script
 - name: Test # Step to run tests
 - run: npm test --if-present # Run tests if available
 - Commit and push:
 - git add . # Add all changes
 - git commit -m "Add GitHub Actions CI" # Commit CI workflow
 - git push # Push to GitHub

The screenshot shows the GitHub Actions tab for the repository "Multi-Cloud-Storage". It displays a single workflow named "ci.yml" with one job named "CI". The job has a single step that runs "npm install" and "npm run build". The status of the run is "SUCCEEDED" with a green checkmark icon. The run was triggered by a pull request from "Aris-Kata26" and completed 2 weeks ago. The code for the workflow is visible in the main pane.

```

name: CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./storage-app
    steps:
      - uses: actions/checkout@v3
      with:
        path: storage-app
      - uses: actions/setup-node@v3
      with:
        node-version: '16'
        cache: 'npm'
      - run: npm install
      - run: npm run build
  
```

Figure 5: GitHub “Actions” tab showing successful CI run

- Why: GitHub for version control, Actions for CI/CD.

Troubleshooting

- **npm start fails:** Delete node_modules and package-lock.json, run npm install, then npm start. Check port conflicts (lsof -i :3000).
- **Git errors:** Verify repository URL (git remote -v), ensure SSH/HTTPS authentication.
- **Extension issues:** Restart VS Code after installing extensions.

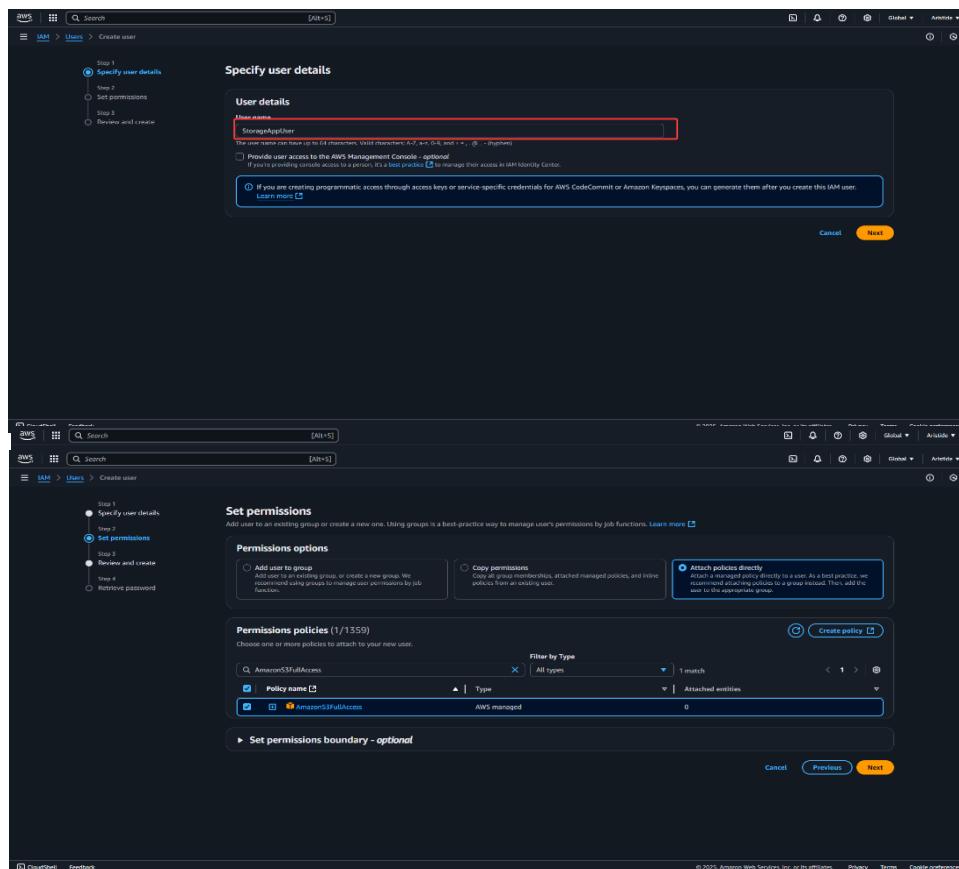
5. Cloud Account Setup (2 hours, June 3, 2025)

Objective: Register and configure free-tier accounts for AWS, Azure, and GCP.

Steps

5. AWS Free Tier :

- Go to aws.amazon.com/free, click “Create a Free Account.”
- Use unique email, provide payment info (no charges within 5GB S3, 20,000 GET requests).
- Select “Basic Plan (Free).”
- In AWS Console:
 - IAM > Users > Add User: storage-user, programmatic access.



The figure consists of three vertically stacked screenshots of the AWS IAM 'Create user' wizard, showing the process of specifying user creation details.

- Step 1: Specify user details** (radio button selected): Shows the 'Set permissions' step. It includes options for 'Add user to group' (selected), 'Copy permissions', and 'Attach policies directly'. A note says: 'Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions.' A 'Create group' button is available.
- Step 2: Set permissions** (radio button selected): Shows the 'Review and create' step. It displays the user name 'StorageAppUser', console password type 'Autogenerated', and 'Require password reset' set to 'Yes'. The 'Permissions summary' table shows two managed policies: 'AmazonS3FullAccess' and 'IAMUserChangePassword'. A 'Tags - optional' section is present, showing 'No tags associated with the resource' and a 'Add new tag' button.
- Step 3: Review and create** (radio button selected): Shows the 'Retrieve password' step. It displays the user name 'StorageAppUser' and a 'Console sign-in details' section with a 'Console sign-in URL' (link to https://596371355130.signin.aws.amazon.com/console) and a 'Email sign-in instructions' button. Buttons for 'Cancel', 'Download .csv file', and 'Return to users list' are at the bottom.

Figure 6: Specify user creation details(AWS)

▪ Attach AmazonS3FullAccess policy.

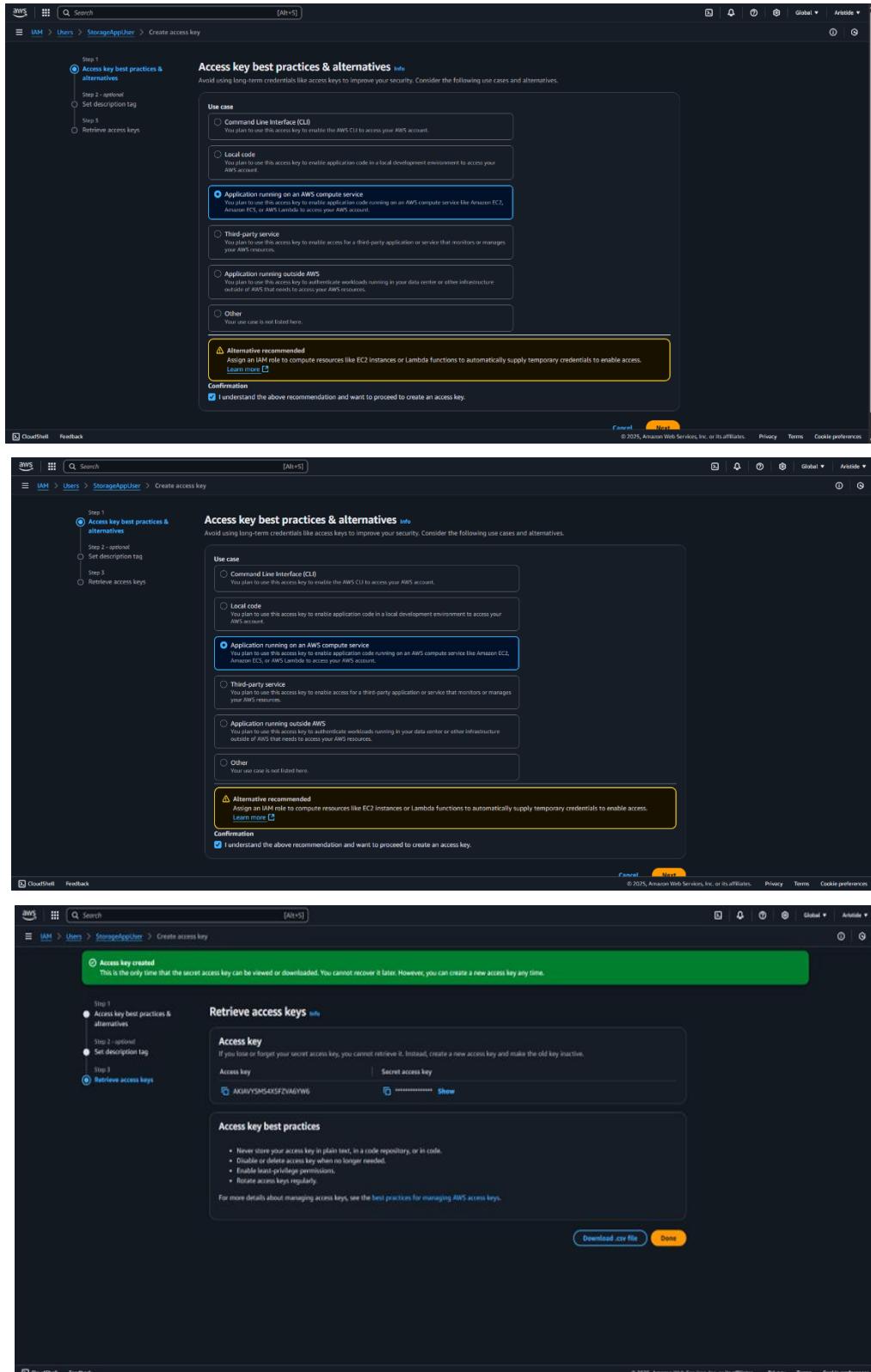


Figure 7: Creating an Access ID and Secret Key

▪ Save Access Key ID and Secret Access Key in backend/.env:

```
▪ AWS_ACCESS_KEY_ID=your-access-key # AWS access key for S3
  AWS_SECRET_ACCESS_KEY=your-secret-key # AWS secret key for S3
```

6. Azure Free Tier:

- Visit azure.microsoft.com/free, sign up with Microsoft account.
- Provide payment info for free services (5GB Blob Storage).
- In Azure Portal:
 - Create Storage Account: mystorageaccount, select “Locally-redundant storage” (LRS).
 - Go to Access keys, copy connection string to backend/.env:

```
AZURE_STORAGE_CONNECTION_STRING=your-connection-string # Azure Blob Storage connection string
```

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure', a search bar, and various icons. Below the navigation bar, there's a section titled 'Azure services' with a 'Create a resource' button and several service icons: Storage accounts, Quickstart Center, Azure AI Foundry, Kubernetes services, Virtual machines, App Services, SQL databases, Azure Cosmos DB, and More services. The 'Storage accounts' icon is highlighted with a red box. Below this is a 'Resources' section with tabs for 'Recent' and 'Favorite'. It shows a table with columns for 'Name', 'Type', and 'Last Viewed'. A message says 'No resources have been viewed recently' with a 'View all resources' button. At the bottom, there's a 'Navigate' section with links for 'Subscriptions', 'Resource groups', 'All resources', and 'Dashboard'.

This screenshot shows the same Azure portal interface, specifically the 'Storage accounts' page. The '+ Create' button is highlighted with a red box. There are also other buttons for 'Restore', 'Manage view', 'Refresh', 'Export to CSV', 'Open query', 'Assign tags', and 'Delete'. A message at the top says 'You are viewing a new version of Browse experience. Some features may be missing. Click here to access the old experience.' Below the buttons is a filter bar with options like 'Filter for any field...', 'Subscription equals all', 'Resource Group equals all', 'Location equals all', and '+ Add filter'. At the bottom, there's a message 'No storage accounts to display' with a link to 'Create'.

This screenshot shows the 'Storage accounts' page again. At the bottom left, there's a 'Showing 1 - 0 of 0. Display count: auto' dropdown menu.

Microsoft Azure

Home > Storage accounts >

Create a storage account

Basics Advanced Networking Data protection Encryption Tags Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more about Azure storage accounts](#)

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription * Azure subscription 1
 Resource group * (New) StorageDemoRG [Create new](#)

Instance details

Storage account name * aristidestorage2025
 Region * (Europe) France Central [Deploy to an Azure Extended Zone](#)
 Primary service Azure Blob Storage or Azure Data Lake Storage Gen 2
 Performance * Standard: Recommended for most scenarios (general-purpose v2 account)
 Premium: Recommended for scenarios that require low latency.
 Redundancy * Locally-redundant storage (LRS)

Review + create

Previous Next Review + create

Secure transfer Enabled
 Blob anonymous access Disabled
 Allow storage account key access Enabled
 Default to Microsoft Entra authorization in the Azure portal Disabled
 Minimum TLS version Version 1.2
 Permitted scope for copy operations From any storage account (preview)
Networking
 Network connectivity Public endpoint (all networks)
 Default routing tier Microsoft network routing
Data protection
 Point-in-time restore Disabled
 Blob soft delete Enabled
 Blob retention period in days 7
 Container soft delete Enabled
 Container retention period in days 7
 File share soft delete Enabled
 File share retention period in days 7
 Versioning Disabled
 Blob change feed Disabled
 Version-level immutability support Disabled
Encryption
 Encryption type Microsoft-managed keys (MMK)
 Enable support for customer-managed keys Blobs and files only
 Enable infrastructure encryption Disabled

... Initializing deployment...
 Initializing template deployment to resource group 'StorageDemoRG.'

Previous Next Create Give feedback

Microsoft Azure

aristidestorage2025_1750081140983 | Overview

Your deployment is complete

Deployment name: aristidestorage2025_1750081140983
Subscription: Azure subscription 1
Resource group: StorageDemoRG

Start time: 16/06/2025, 15:39:53
Correlation ID: bd8b14b6-c032-4397-826c-740e7c5cebcf

Deployment succeeded
Deployment "aristidestorage2025_1750081140983" to resource group "StorageDemoRG" was successful.

Next steps

Go to resource

Give feedback
Tell us about your experience with deployment

Cost Management
Get notified to stay within your budget and prevent unexpected charges on your bill.
Set up cost alerts >

Microsoft Defender for Cloud
Secure your apps and infrastructure
Go to Microsoft Defender for Cloud >

Free Microsoft tutorials
Start learning today >

Work with an expert
Azure experts are service provider partners who can help manage your assets on Azure and be your first line of support.
Find an Azure expert >

Storage account

aristidestorage2025 | Containers

Containers

Name	Last modified	Anonymous access level	Lease state
Slogs	16/06/2025, 15:40:16	Private	Available

<https://portal.azure.com/#@aristikataganuk2025outlook.onmicrosoft.com/resource/subscriptions/60ae257b-93bb-4d1e-bff7-41ac0a1396/resourcegroups/StorageDemoRG/providers/Microsoft.Storage/storageAccounts/aristidestorage2025/containersList>

The screenshot shows two pages from the Microsoft Azure portal related to a storage account named 'aristidestorage2025'.

Top Page (Container Creation):

- URL: [https://portal.azure.com/#blade/HubsBlade/resourceType/storageAccounts/resourceName/aristidestorage2025/containers](#)
- Section: Containers
- Form Fields:
 - Name: my-free-tier-container
 - Anonymous access level: Private (no anonymous access)
 - Note: "The access level is set to private because anonymous access is disabled on this storage account."
- Buttons: Create (highlighted with a red box), Give feedback, Refresh, Delete, Restore containers, Change access level.

Bottom Page (Container Listing):

- URL: [https://portal.azure.com/#blade/HubsBlade/resourceType/storageAccounts/resourceName/aristidestorage2025/containers](#)
- Section: Containers
- Table Data:

Name	Last modified	Anonymous access level	Lease state	More
Slogs	16/06/2025, 15:40:16	Private	Available	...
my-free-tier-container	16/06/2025, 15:47:37	Private	Available	...
- Buttons: Search, Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, Storage Mover, Partner solutions, Resource visualizer, Help.

Figure 8: Azure Portal showing connection string

7. GCP Free Tier (0.6 hour):

- Sign up at cloud.google.com/free with \$300 credit (90 days).
- In Google Cloud Console:
 - Create project: multi-cloud-storage.
 - Enable Cloud Storage API (APIs & Services > Library).
 - Create service account: storage-service-account, assign “Storage Admin” role, download JSON key as backend/key.json.
 - Set in backend/.env:

```
GOOGLE_APPLICATION_CREDENTIALS=/app/key.json # Path to
GCP service account key
```

The screenshot shows the Google Cloud Home page with a sidebar on the left containing links like Cloud Hub, Cloud overview, Solutions, Billing, IAM & Admin, Marketplace, APIs & Services, Vertex AI, Compute Engine, Kubernetes Engine, Cloud Storage, Security, BigQuery, Monitoring, Cloud Run, VPC Network, and Cloud SQL. A modal window titled "Enabled APIs & services" is open, showing sections for Library, Credentials, OAuth consent screen, and Page usage agreements. Below the sidebar, there's a "Products" section with cards for Create a VM, Deploy an application, Build a mobile or web app, Create a Kubernetes cluster, Create a storage bucket, Distribute network traffic, Accelerate content delivery, and Monitor your workloads.

API Library

Welcome to the API Library. The API Library has documentation, links, and a smart search experience.

Search bar: cloud storage

Maps

- Visibility: Public (489), Private (2)
- Category: Analytics (11), Big data (22), Databases (7), Machine learning (16), Maps (31), DevOps (24), Compute (13)
- APIs listed: Maps SDK for Android, Maps SDK for iOS, Maps JavaScript API, Places API, Roads API, Directions API.

Machine learning

- APIs listed: Dialogflow API, Cloud Vision API, Cloud Natural Language API, Cloud Speech-to-Text API, Cloud Translation API, AI Platform Training & Prediction API.

Cloud Storage API

15 results

- Cloud Storage: Google Cloud Storage is a RESTful service for storing and accessing your data on Google's infrastructure.
- Google Cloud Storage JSON API: Lets you store and retrieve potentially-large, immutable data objects.
- Cloud Storage API: Lets you store and retrieve potentially-large, immutable data objects. (This item is highlighted with a red box.)
- Cloud Storage for Firebase API: The Cloud Storage for Firebase API enables programmatic management of Cloud Storage buckets for use in Firebase projects.
- Storage Transfer API: Transfers data from external data sources to a Google Cloud Storage bucket or between Google Cloud Storage buckets.

Storage Insights API

<https://console.cloud.google.com/api/library/storage.googleapis.com?tab=1#/and-Al/0#&project=high-empire-463106-v4>

Google Cloud Project: My First Project

Cloud Storage API (Google Enterprise API)

Lets you store and retrieve potentially-large, immutable data objects.

Manage Try this API API Enabled

Overview Documentation Related Products

Overview

Lets you store and retrieve potentially-large, immutable data objects.

Additional details

- Type: [Saas & APIs](#)
- Last product update: 4/11/23
- Category: [Google Enterprise APIs](#)
- Service name: storage.googleapis.com

Tutorials and documentation

[Learn more](#)

Terms of Service

By using this product you agree to the terms and conditions of the following license: [Google Cloud Platform](#).

API/Service Details

To use this API you may need credentials.

Create credentials

Storage API

Principal Access Boundary: Lets you store and retrieve potentially-large, immutable data objects.

Organization: Google Enterprise API

Identity & Organization:

- Policy Troubleshooter
- Policy Analyzer
- Organization Policies
- Service Accounts** (selected)
- Workload Identity Federation
- Workforce Identity Federation

Type: Public API Status: Enabled Documentation: [Learn more](#) [Try in API Explorer](#)

Metrics Credentials Cost

No data is available for the selected time frame.

View all products

<https://console.cloud.google.com/iam-admin/serviceaccounts?tab=iam&project=high-expense-463106#>

IAM & Admin / Service accounts

+ Create service account

Service accounts for project "My First Project"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. [Learn more about service accounts](#)

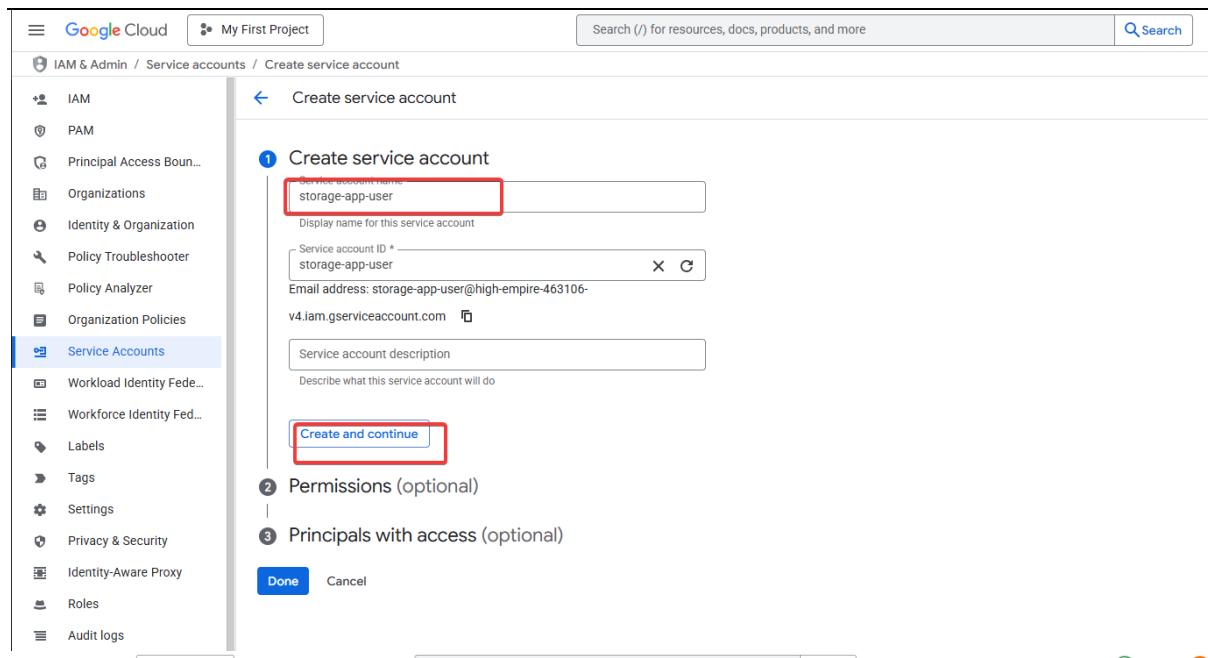
Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. [Learn more about service account organization policies](#)

Filter: Enter property name or value

Email	Status	Name	Description	Key ID	Key creation date	OAuth 2 Client ID	Actions
No rows to display							

Recommended for you

- Service accounts overview** Help document IAM service accounts, permissions, and common lifecycle considerations.
- Create service accounts** Help document Create service accounts using the IAM API, the Google Cloud console, and the gcloud CLI.
- List and edit service accounts** Help document List and edit service accounts using the IAM API, the Google Cloud console, and the gcloud CLI.
- Disable and enable service accounts** Help document Disable and enable service accounts using the IAM API, the Google Cloud console, and the gcloud CLI.
- Delete and undelete service accounts** Help document Delete and undelete service accounts using the IAM API, the Google Cloud console, and the gcloud CLI.
- Manage access to projects, folders, and organizations** Help document This page describes how to grant, change, and revoke access to projects, folders, and organizations.



Create service account

Service account name: **storage-app-user**

Service account ID: **storage-app-user**

Email address: **storage-app-user@high-empire-463106-v4.iam.gserviceaccount.com**

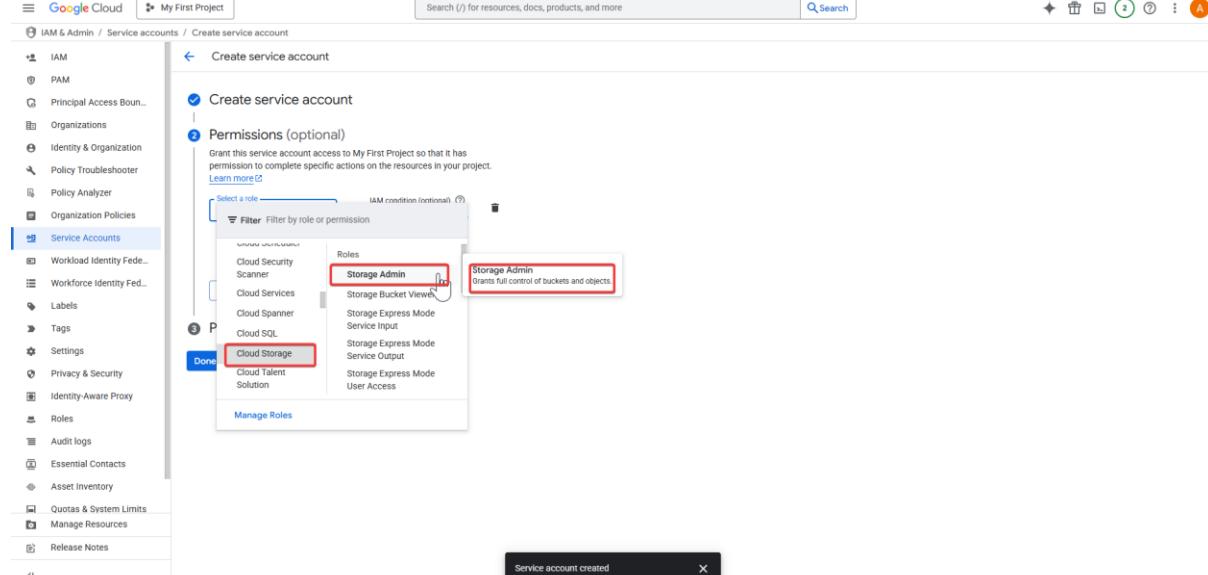
Service account description:

Create and continue

Permissions (optional)

Principals with access (optional)

Done **Cancel**



Create service account

Permissions (optional)

Grant this service account access to My First Project so that it has permission to complete specific actions on the resources in your project.

Select a role

Storage Admin Grants full control of buckets and objects

Cloud Storage

Storage Admin

Manage Roles

Service account created

Google Cloud IAM & Admin / Service accounts

Service accounts

Service accounts for project "My First Project". A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. Learn more about service accounts.

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. Learn more about service account organization policies.

Email	Status	Name	Description	Key ID	Key creation date	OAuth 2 Client ID	Actions
storage-app-user@high-empire-46310e-v4iam.gserviceaccount.com	Enabled	storage-app-user		No keys		115851505715234834375	⋮

No change - principal already exists on the policy.

Google Cloud IAM & Admin / Service accounts / Service account: 115851505715234834375 / Keys

storage-app-user

Keys

Service account keys could pose a security risk if compromised. We recommend you avoid downloading service account keys and instead use the Workload Identity Federation. Learn more about the best way to authenticate service accounts on Google Cloud.

Google automatically disables service account keys detected in public repositories. You can customize this behavior by using the 'iam.serviceAccountKeyExposureResponse' organization policy. Learn more.

Add a new key pair or upload a public key certificate from an existing key pair.

Block service account key creation using organization policies. Learn more about setting organization policies for service accounts.

Add key

Type	Status	Key	Creation date	Expiration date
No rows to display				

Google Cloud IAM & Admin / Service accounts / Service account: 115851505715234834375 / Keys

storage-app-user

Keys

Service account keys could pose a security risk if compromised. We recommend you avoid downloading service account keys and instead use the Workload Identity Federation. Learn more about the best way to authenticate service accounts on Google Cloud.

Google automatically disables service account keys detected in public repositories. You can customize this behavior by using the 'iam.serviceAccountKeyExposureResponse' organization policy. Learn more.

Add a new key pair or upload a public key certificate from an existing key pair.

Block service account key creation using organization policies. Learn more about setting organization policies for service accounts.

Add key

Create new key

Key Creation date Expiration date

Upload existing

Google Cloud IAM & Admin / Service accounts / Service account: 115851505715234834375 / Keys

Create private key for "storage-app-user"

Key type

- JSON** Recommended
- P12 For backward compatibility with code using the P12 format

Cloud Storage Overview

Welcome to Cloud Storage, Aristide!

Create bucket

Get Started

Pick a globally unique, permanent name. [Naming guidelines](#)

Tip: Don't include any sensitive information

Labels (optional)

Location type

- Multi-region** Highest availability across largest area
- Dual-region** High availability and low latency across 2 regions
- Region** Lowest latency within a single region

Choose where to store your data

This choice defines the geographic placement of your data and affects cost, performance, and availability. Cannot be changed later. [Learn more](#)

eu (multiple regions European Union)

Add cross-bucket replication via Storage Transfer Service

As data is added or changed, replicate it to another bucket, enabling you to store a copy that follows different bucket settings - e.g., region, storage class, etc. [Learn more](#)

Good to know

Location pricing

Storage rates vary depending on the storage class of your data and location of your bucket. [Pricing details](#)

Current configuration: Multi-region / Standard

Item	Cost
eu (multiple regions in European Union)	\$0.026 per GB-month
With default replication	\$0.020 per GB written

[Estimate your monthly cost](#)

Google Cloud My First Project

Create a bucket

- Choose how to store your data**
 - Autoclass
 - Set a default class

Standard: Best for short-term storage and frequently accessed data

 - Newline: Best for backups and data accessed less than once a month
 - Coldline: Best for disaster recovery and data accessed less than once a quarter
 - Archive: Best for long-term digital preservation of data accessed less than once a year

Optimize storage for data-intensive workloads

Enable Hierarchical namespace on this bucket

Continue

Marketplace Release Notes

Cloud Storage My First Project

Create a bucket

- Choose how to store your data**
 - Standard
 - Hierarchical namespace: Disabled
- Choose how to control access to objects**
 - Public access prevention: On
 - Access control: Uniform
- Choose how to protect object data**
 - Your data is always protected with Cloud Storage but you can also choose from these additional data protection options to add extra layers of security.

Data protection

 - Soft delete policy (For data recovery) When you delete this bucket and its objects will be kept for a specified period after they're deleted and can be restored during this time. [Learn more](#)
 - User default retention duration All objects have a 7 day soft delete duration by default, unless this default has been customized by your organization administrator.
 - Set custom retention duration Specify how long this bucket and its objects should be retained after they're deleted. Setting a "0" duration disables soft delete, meaning any deleted objects will be permanently deleted.

Object versioning (For version control) For restoring deleted or overwritten objects. To minimize the cost of storing versions, we keep only the most recent version of each object and scheduling them to expire after a number of days. [Learn more](#)

Retention (For compliance) For preventing the deletion or modification of the bucket's objects for a specified period of time.

Data Encryption

Create **Cancel**

click "Create"

Google Cloud My First Project

Bucket details

aristide-free-tier-2025

Location	Storage class	Public access	Protection
eu (multiple regions in European Union)	Standard	Not public	Soft Delete

Objects Configuration Permissions Protection Lifecycle Observability New Inventory Reports Operations

Buckets > aristide-free-tier-2025

Create folder Upload Transfer data Other services

Created bucket aristide-free-tier-2025 Your bucket is ready. Just add data. Drag files and folders here or use the upload button. To move a lot of data.

Get started with Cloud Storage

 - Getting bucket information**
 - Uploading objects**
 - Downloading objects**
 - Use cases for Cloud Storage**
 - Terraform samples**
 - Architecture guides for storage**
 - Prevent public access**

Figure 9:: GCP Console showing enabled API and downloaded key

Troubleshooting

- **Account conflicts:** Use unique emails, clear cookies.
 - **Payment issues:** Try virtual card or contact support.
 - **API not enabled:** Verify Cloud Storage API in GCP Console.
-

6. Web App Frontend Development (8 hours, June 5, 2025)

Objective: Build a React interface with components for file operations, dashboard, storage info, and provider comparison.

Steps

8. Set Up React Structure:

- **Create folder structure:**

```
○ cd storage-app # Navigate to React project
○ mkdir src/components src/pages src/services # Create
    component, pages, and services directories
```
- **Create storage-app/src/pages/Home.js:**

```
○ import React from 'react'; // Import React library
○ function Home() { // Define Home component
○   return ( // Render JSX
○     <div className="p-4 bg-gray-100 min-h-screen"> // Container with padding and background
○       <h1 className="text-2xl font-bold text-blue-600">Multi-Cloud Storage Demo</h1> // Main title
○       <p className="text-gray-700">Manage files across AWS, Azure, and GCP.</p> // Description
○     </div>
○   );
○ }
○ export default Home; // Export Home component
```
- **Update storage-app/src/App.js:**

```
○ import Home from './pages/Home'; // Import Home component
○ function App() { // Define App component
○   return ( // Render JSX
○     <div className="App"> // App container
○       <Home /> // Render Home component
○     </div>
○   );
○ }
○ export default App; // Export App component
```
- **Test:** npm start, verify at <http://localhost:3000>.



Figure 10: Browser showing Home.js with styled header

9. Implement File Upload/Download UI:

- o Create storage-app/src/components/FileUpload.js:
- o import React, { useState } from 'react'; // Import React and useState hook
- o import { uploadFile } from '../services/api'; // Import API utility for file upload
- o function FileUpload() { // Define FileUpload component
- o const [file, setFile] = useState(null); // State for selected file
- o const [provider, setProvider] = useState('s3'); // State for selected cloud provider
- o const [status, setStatus] = useState(''); // State for upload status message
- o const handleFileChange = (e) => { // Handle file input change
- o if (e.target.files.length > 0) { // Check if file is selected
- o setFile(e.target.files[0]); // Update file state
- o }
- o };
- o const handleUpload = async () => { // Handle file upload
 if (!file) { // Check if file is selected
 setStatus('No file selected'); // Set error status
 return;
 }
 setStatus('Uploading...'); // Set uploading status
 try { // Handle API call
 await uploadFile(file, provider); // Call upload API
 setStatus(`Uploaded to \${provider.toUpperCase()}`); // Set success status
 } catch (error) { // Handle errors

```

o         setStatus(`Upload failed: ${error.message}`); // Set
o         error status
o     }
o   };
o   return ( // Render JSX
o     <div className="m-4 bg-white p-4 rounded shadow"> // Container with styling
o       <h2 className="text-xl font-semibold text-gray-800">File Upload</h2> // Section title
o       <select
o         value={provider} // Bind provider state
o         onChange={(e) => setProvider(e.target.value)} // Update provider state
o         className="mb-2 border rounded p-2" // Styling for dropdown
o       >
o         <option value="s3">AWS S3</option> // AWS option
o         <option value="azure">Azure Blob</option> // Azure option
o         <option value="gcp">GCP Cloud Storage</option> // GCP option
o       </select>
o       <input
o         type="file" // File input
o         onChange={handleFileChange} // Bind file change handler
o         className="mb-2 border rounded p-2" // Styling for input
o       />
o       <button
o         onClick={handleUpload} // Bind upload handler
o         className="
o           bg-blue-500 text-white p-2 rounded hover:bg-blue-600"
// Styling for button
o       >
o         Upload File // Button text
o       </button>
o       <p className="mt-2 text-gray-700">{status}</p> // Display status message
o     </div>
o   );
o }
o export default FileUpload; // Export FileUpload component

o Create storage-app/src/services/api.js:
o import axios from 'axios'; // Import axios for API calls
o export const uploadFile = async (file, provider) => { // Function to upload single file
o   const formData = new FormData(); // Create FormData for file upload
o   formData.append('file', file); // Append file to FormData
o   const response = await axios.post(`/upload/${provider}`, formData); // Send POST request
o   return response.data; // Return response data
o };
o export const uploadBatch = async (files) => { // Function to upload multiple files
o   const formData = new FormData(); // Create FormData for file upload

```

```

o      for (let i = 0; i < files.length; i++) { // Loop through
files
o      formData.append('files', files[i]); // Append each file to
FormData
o      }
o      const response = await axios.post('/upload/batch', formData);
// Send POST request
o      return response.data; // Return response data
};

```

10. Implement Storage Dashboard:

- o Create storage-app/src/components/StorageDashboard.js:

```

o import React, { useState, useEffect } from 'react'; // Import
React, useState, and useEffect
o import axios from 'axios'; // Import axios for API calls
o function StorageDashboard() { // Define StorageDashboard
component
o   const [data, setData] = useState({ // Initialize state for
dashboard data
o     usage: { s3: 0, azure: 0, gcp: 0 }, // Storage usage for
each provider
o     savings: 0, // Cost savings
o     alerts: [], // API limit alerts
o     energy: 0 // Carbon footprint
o   });
o   useEffect(() => { // Fetch data on component mount
o     axios.get('/dashboard') // Send GET request to dashboard
endpoint
o       .then(response => setData(response.data)) // Update
state with response data
o       .catch(error => console.error('Error fetching dashboard
data:', error)); // Log errors
o   }, []); // Empty dependency array for single fetch
o   return (
o     <div className="m-4 bg-white p-4 rounded shadow"> // Container with styling
o       <h2 className="text-xl font-semibold text-gray-800">Storage Dashboard</h2> // Section title
o       <p className="text-gray-700">AWS S3: {data.usage.s3} MB</p> // Display S3 usage
o       <p className="text-gray-700">Azure Blob: {data.usage.azure} MB</p> // Display Azure usage
o       <p className="text-gray-700">GCP Cloud Storage: {data.usage.gcp} MB</p> // Display GCP usage
o       <p className="text-green-600">Savings: ${data.savings}/month (10TB archival)</p> // Display savings
o       <p className="text-gray-700">Carbon Footprint: {data.energy} kWh</p> // Display energy usage
o       <div className="bg-red-200 p-2 mt-2 rounded"> // Container for alerts
o         {data.alerts.map((alert, i) => ( // Map through alerts
o           <p key={i} className="text-red-700">{alert}</p> // Display each alert
o         )));
o       </div>
o     </div>
o   );
o }

```

```
export default StorageDashboard; // Export StorageDashboard
component
```

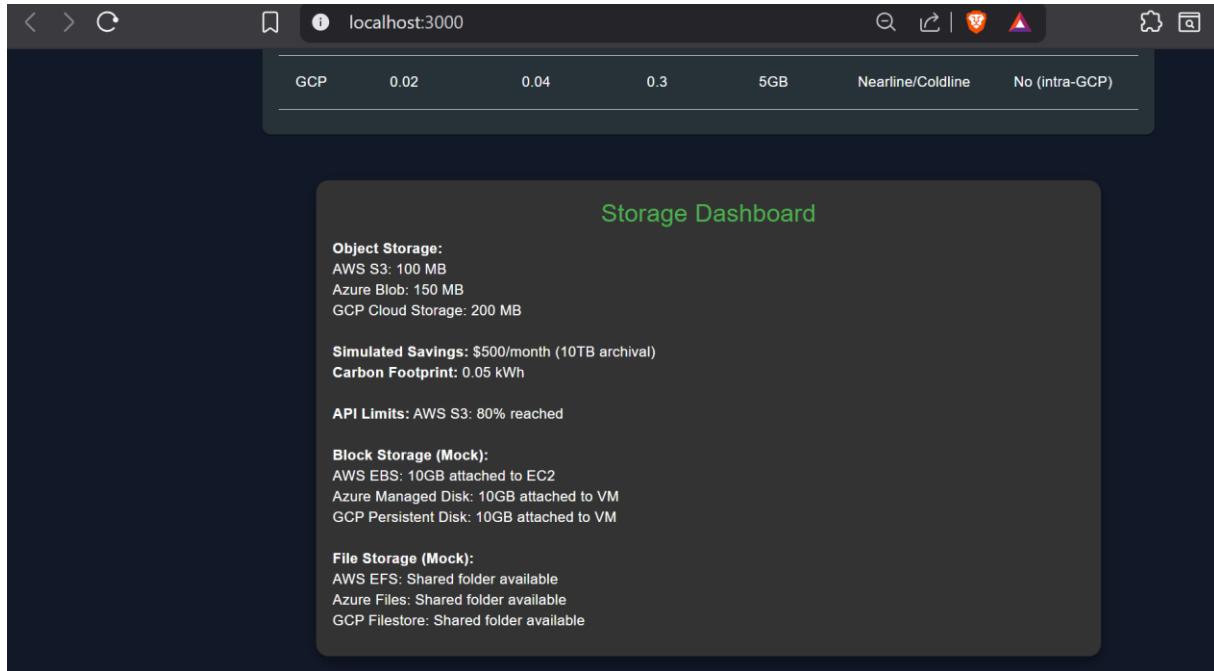


Figure 11: Browser showing dashboard with mock data

11. Implement Storage Info and Use Cases:

- Create storage-app/src/components/StorageInfo.js:
- import React, { useState, useEffect } from 'react'; // Import React, useState, and useEffect
- import axios from 'axios'; // Import axios for API calls
- function StorageInfo() { // Define StorageInfo component
- const [blockData, setBlockData] = useState({}); // State for block storage data
- const [fileData, setFileData] = useState({}); // State for file storage data
- useEffect(() => { // Fetch data on component mount
- axios.get('/demo/block') // Send GET request for block storage data
- .then(response => setBlockData(response.data)) // Update block data state
- .catch(error => console.error('Error fetching block data:', error)); // Log errors
- axios.get('/demo/file') // Send GET request for file storage data
- .then(response => setFileData(response.data)) // Update file data state
- .catch(error => console.error('Error fetching file data:', error)); // Log errors
- }, []); // Empty dependency array for single fetch
- return (// Render JSX
- <div className="m-4 bg-white p-4 rounded shadow"> // Container with styling
- <h2 className="text-xl font-semibold text-gray-800">Storage Types & Use Cases</h2> // Section title
- <div className="mt-2"> // Container for storage types

```

○          <h3 className="text-lg font-medium">Object Storage</h3>
    // Object storage title
○          <p className="text-gray-700">Ideal for backups,
    archives (e.g., AWS S3 Glacier).</p> // Object storage
    description
○          <h3 className="text-lg font-medium mt-2">Block
    Storage</h3> // Block storage title
○          <p className="text-gray-700">AWS: {blockData.aws}</p>
    // Display AWS block data
○          <p className="text-gray-700">Azure:
    {blockData.azure}</p> // Display Azure block data
○          <p className="text-gray-700">GCP: {blockData.gcp}</p>
    // Display GCP block data
○          <h3 className="text-lg font-medium mt-2">File
    Storage</h3> // File storage title
○          <p className="text-gray-700">AWS: {fileData.aws}</p>
    // Display AWS file data
○          <p className="text-gray-700">Azure:
    {fileData.azure}</p> // Display Azure file data
○          <p className="text-gray-700">GCP: {fileData.gcp}</p>
    // Display GCP file data
○          </div>
○          </div>
○      );
○  }
  export default StorageInfo; // Export StorageInfo component

```

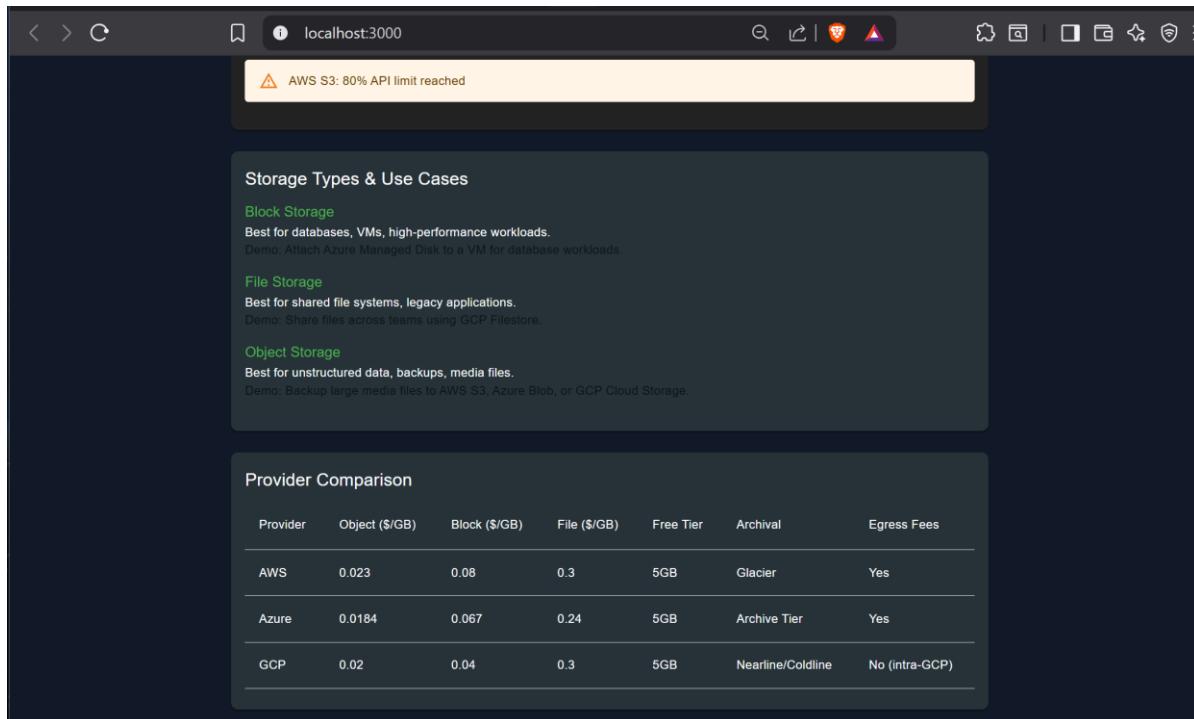


Figure 12: Browser showing storage types and use cases

12. Implement Provider Comparison:

- Create storage-app/src/components/ProviderComparison.js:
- import React, { useState, useEffect } from 'react'; // Import React, useState, and useEffect

- o import axios from 'axios'; // Import axios for API calls
- o function ProviderComparison() { // Define ProviderComparison component
- o const [providers, setProviders] = useState([]); // State for provider data
- o useEffect(() => { // Fetch data on component mount
- o axios.get('/provider/comparison') // Send GET request for comparison data
- o .then(response => setProviders(response.data)) // Update providers state
- o .catch(error => console.error('Error fetching comparison data:', error)); // Log errors
- o }, []); // Empty dependency array for single fetch
- o return (// Render JSX
- o <div className="m-4 bg-white p-4 rounded shadow"> // Container with styling
- o <h2 className="text-xl font-semibold text-gray-800">Provider Comparison</h2> // Section title
- o <table className="w-full mt-2 text-left"> // Table for comparison data
- o <thead> // Table header
- o <tr className="bg-gray-200"> // Header row styling
- o <th className="p-2">Provider</th> // Provider column
- o <th className="p-2">Cost</th> // Cost column
- o <th className="p-2">Free Tier</th> // Free Tier column
- o <th className="p-2">Egress Fees</th> // Egress Fees column
- o </tr>
- o </thead>
- o <tbody> // Table body
- o {providers.map((provider, i) => (// Map through providers
- o <tr key={i} className="border-t"> // Table row with border
- o <td className="p-2">{provider.name}</td> // Display provider name
- o <td className="p-2">{provider.cost}</td> // Display cost
- o <td className="p-2">{provider.freeTier}</td> // Display free tier
- o <td className="p-2">{provider.egress}</td> // Display egress fees
- o </tr>
- o))}
- o </tbody>
- o </table>
- o </div>
- o);
- o }
- o export default ProviderComparison; // Export ProviderComparison component

- o **Update Home.js to include all components:**

- o import React from 'react'; // Import React library
- o import FileUpload from '../components/FileUpload'; // Import FileUpload component

```

○ import StorageDashboard from '../components/StorageDashboard';
  // Import StorageDashboard component
○ import StorageInfo from '../components/StorageInfo'; // Import
  StorageInfo component
○ import ProviderComparison from
  '../components/ProviderComparison'; // Import
  ProviderComparison component
○ function Home() { // Define Home component
○   return ( // Render JSX
○     <div className="p-4 bg-gray-100 min-h-screen"> // Container with styling
○       <h1 className="text-2xl font-bold text-blue-600">Multi-
  Cloud Storage Demo</h1> // Main title
○       <FileUpload /> // Render file upload component
○       <StorageDashboard /> // Render dashboard component
○       <StorageInfo /> // Render storage info component
○       <ProviderComparison /> // Render provider comparison
  component
○     </div>
○   );
○ }
○
○ export default Home; // Export Home component

```

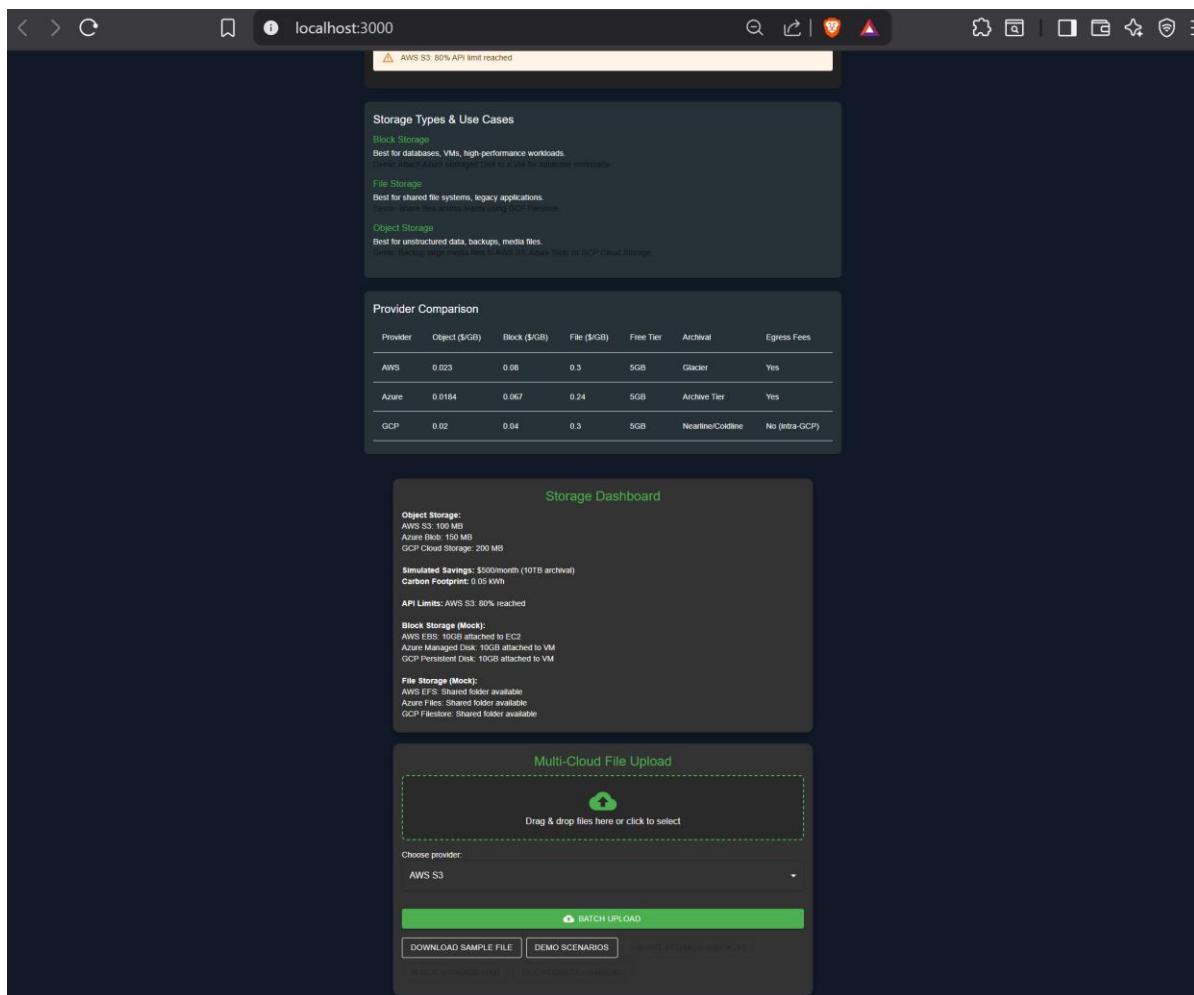


Figure 13: Browser showing all components (upload, dashboard, info, comparison).

Troubleshooting

- **UI not rendering:** Check browser console (Ctrl+Shift+J), verify imports.
- **Tailwind issues:** Ensure tailwind.config.js includes src/**/*.{js,jsx}.
- **API call failures:** Verify package.json proxy and backend running.

7. Cloud Storage API Integration (10 hours, June 7, 2025)

Objective: Develop Flask backend with endpoints for file uploads, mock scenarios, dashboard data, and provider comparison.

Steps

13. Set Up Flask Backend:

- **Create backend folder:**

```
mkdir backend # Create backend directory
cd backend # Navigate to backend directory
python -m venv venv # Create Python virtual environment
source venv/bin/activate # Activate virtual environment
(Windows: venv\Scripts\activate)
pip install flask flask-cors boto3 azure-storage-blob google-cloud-storage redis # Install dependencies
```
- **Create backend/requirements.txt:**

```
flask==2.0.1 # Flask framework for backend
flask-cors==3.0.10 # CORS support for Flask
boto3==1.24.0 # AWS SDK for Python
azure-storage-blob==12.13.0 # Azure Blob Storage SDK
google-cloud-storage==2.5.0 # GCP Cloud Storage SDK
redis==3.5.3 # Redis client for caching
```
- **Create backend/app.py:**

```
from flask import Flask, request # Import Flask for web server and request handling
from flask_cors import CORS # Import CORS for cross-origin requests
app = Flask(__name__) # Initialize Flask app
CORS(app) # Enable CORS for frontend communication
@app.route('/') # Define root endpoint
def home(): # Home route handler
    return 'Backend is running!' # Return status message
if __name__ == '__main__': # Check if script is run directly
    app.run(debug=True, port=5000) # Start Flask server on port 5000
```
- **Create temp folder for temporary file storage:**

```
mkdir temp # Create temporary file storage directory
```

-
- Start backend:
 - pip install -r requirements.txt # Install dependencies
python app.py # Run Flask app
 - Verify at <http://localhost:5000>.

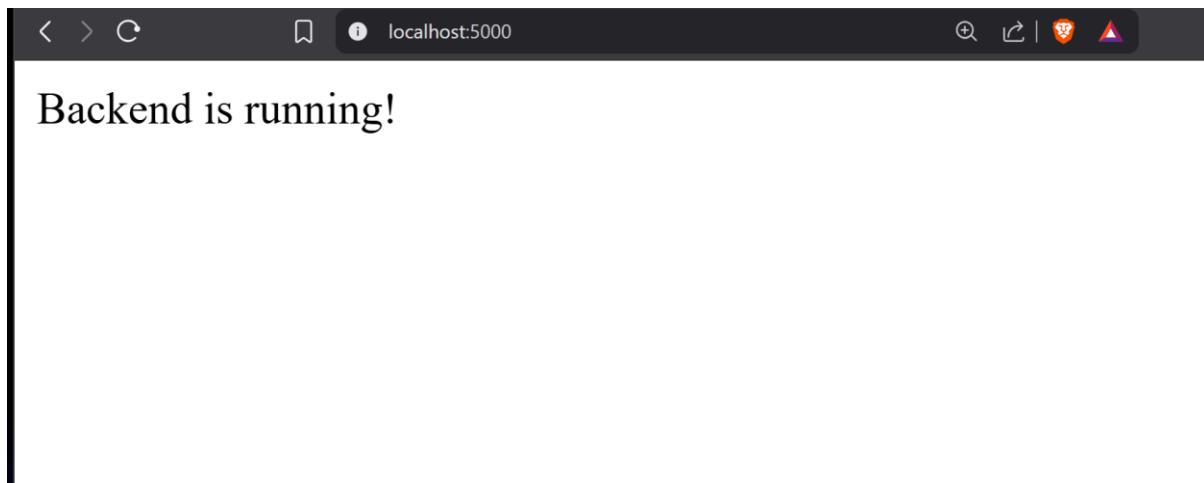


Figure 14: Browser showing “Backend is running!”.

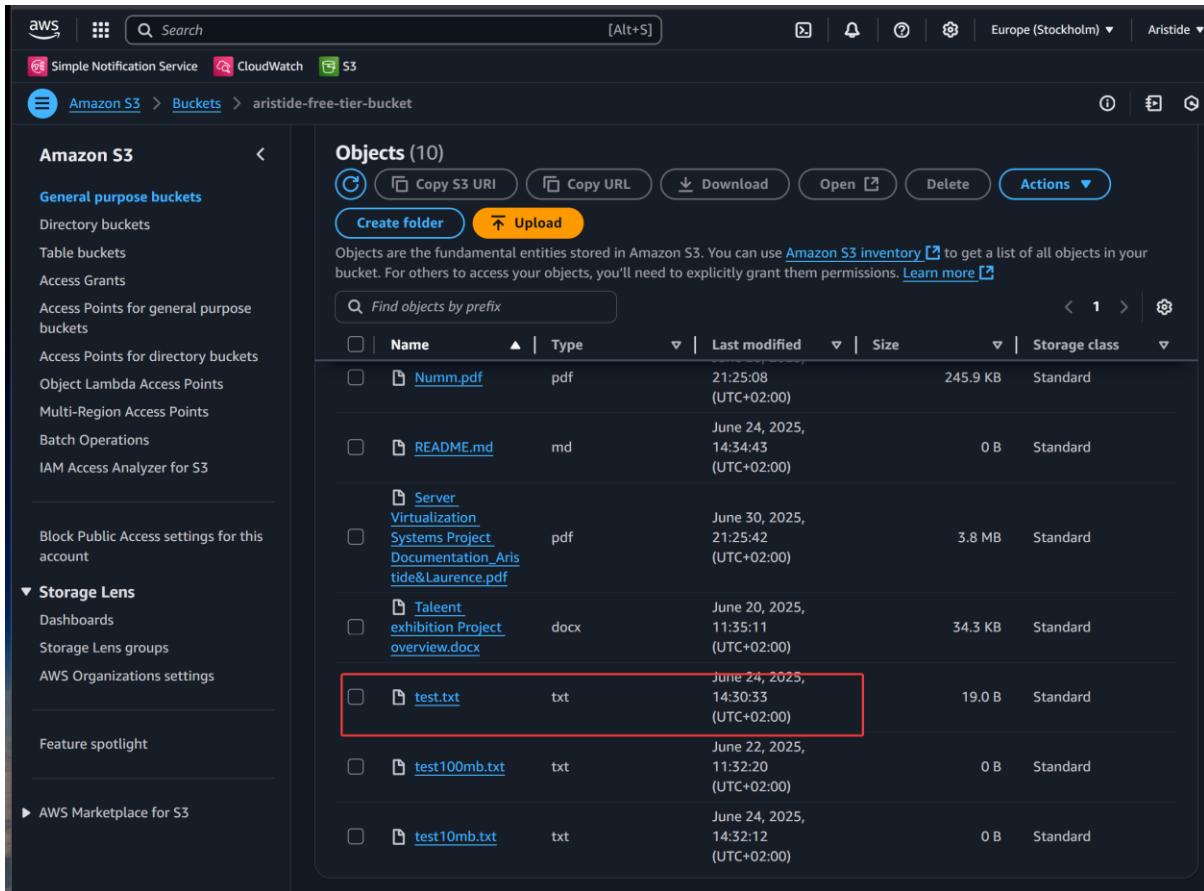
14. Integrate AWS S3:

- Create backend/.env:
- AWS_ACCESS_KEY_ID=your-access-key # AWS access key for S3
- AWS_SECRET_ACCESS_KEY=your-secret-key # AWS secret key for S3
- AZURE_STORAGE_CONNECTION_STRING=your-connection-string # Azure Blob Storage connection string
- GOOGLE_APPLICATION_CREDENTIALS=/app/key.json # Path to GCP service account key
- Create backend/file_upload.py (see Section 15).
- Update app.py with S3 upload endpoint:


```
from file_upload import upload_to_s3 # Import S3 upload function
from energy_estimator import estimate_energy # Import energy estimation function
import redis # Import Redis for caching
cache = redis.Redis(host='redis', port=6379) # Initialize Redis client
def get_cached(key): # Function to get cached data
    return cache.get(key) # Retrieve value from Redis
def set_cached(key, value, ttl=300): # Function to set cached data
    cache.setex(key, ttl, value) # Store value in Redis with TTL
@app.route('/upload/s3', methods=['POST']) # Define S3 upload endpoint
def upload_s3(): # S3 upload handler
    if 'file' not in request.files: # Check if file is included
        return 'No file provided', 400 # Return error if no file
    file = request.files['file'] # Get uploaded file
```

- o file_path = f'temp/{file.filename}' # Define temporary file path
- o cache_key = f'upload_s3_{file.filename}' # Create cache key for file
- o if get_cached(cache_key): # Check if file is already uploaded
 - o return 'File already uploaded', 200 # Return cached response
- o file.save(file_path) # Save file to temporary path
- o upload_to_s3(file_path, 'my-free-tier-bucket', file.filename) # Upload to S3
- o set_cached(cache_key, 'Uploaded') # Cache upload status
- o energy = estimate_energy(file.content_length / (1024 * 1024)) # Calculate energy usage
 - o return f'File uploaded to S3: {file.filename}, Energy: {energy:.4f} kWh', 200 # Return success message
- o In AWS Console: Create bucket my-free-tier-bucket (us-east-1, disable public access block).
- o Test:

```
curl -X POST -F "file=@test.txt"
http://localhost:5000/upload/s3 # Test S3 upload
```



The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with navigation links like 'Amazon S3', 'General purpose buckets', 'Storage Lens', 'Feature spotlight', and 'AWS Marketplace for S3'. The main area is titled 'Objects (10)' and shows a list of files in the 'aristide-free-tier-bucket'. The 'test.txt' file is highlighted with a red box. The table below lists the objects:

Name	Type	Last modified	Size	Storage class
Numm.pdf	pdf	21:25:08 (UTC+02:00)	245.9 KB	Standard
README.md	md	June 24, 2025, 14:34:43 (UTC+02:00)	0 B	Standard
Server Virtualization Systems Project Documentation_Aristide&Laurence.pdf	pdf	June 30, 2025, 21:25:42 (UTC+02:00)	3.8 MB	Standard
Talent exhibition Project overview.docx	docx	June 20, 2025, 11:35:11 (UTC+02:00)	34.3 KB	Standard
test.txt	txt	June 24, 2025, 14:30:33 (UTC+02:00)	19.0 B	Standard
test100mb.txt	txt	June 22, 2025, 11:32:20 (UTC+02:00)	0 B	Standard
test10mb.txt	txt	June 24, 2025, 14:32:12 (UTC+02:00)	0 B	Standard

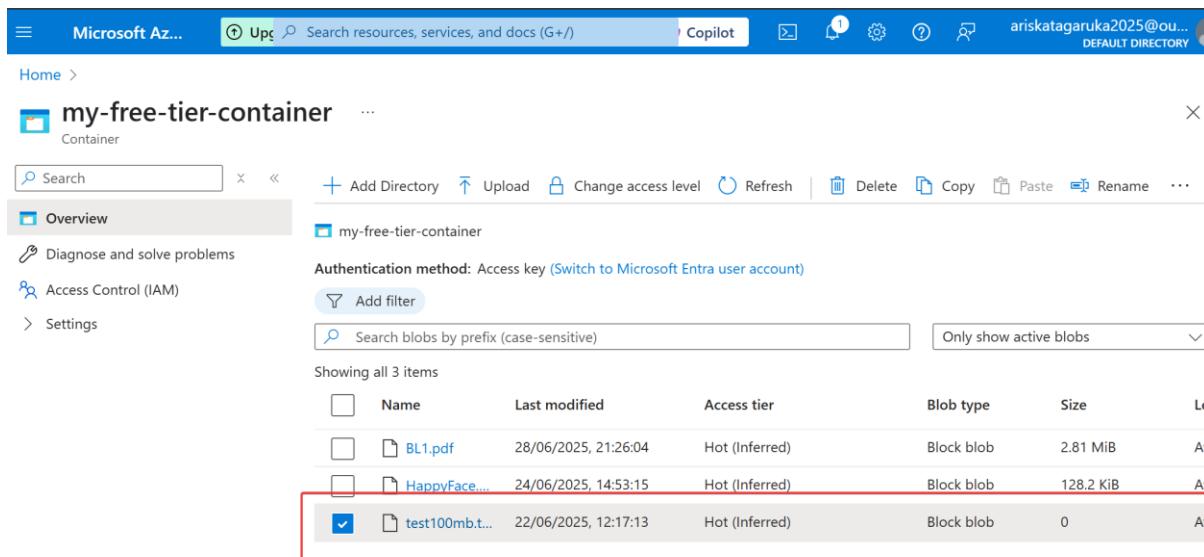
Figure 15: AWS S3 Console showing my-free-tier-bucket with test.txt

15. Integrate Azure Blob Storage:

- Add to app.py:


```
from file_upload import upload_to_azure # Import Azure upload function
@app.route('/upload/azure', methods=['POST']) # Define Azure upload endpoint
def upload_azure(): # Azure upload handler
    if 'file' not in request.files: # Check if file is included
        return 'No file provided', 400 # Return error if no file
    file = request.files['file'] # Get uploaded file
    file_path = f'temp/{file.filename}' # Define temporary file path
    cache_key = f'upload_azure_{file.filename}' # Create cache key for file
    if get_cached(cache_key): # Check if file is already uploaded
        return 'File already uploaded', 200 # Return cached response
    file.save(file_path) # Save file to temporary path
    upload_to_azure(file_path, 'my-free-tier-container', file.filename) # Upload to Azure
    set_cached(cache_key, 'Uploaded') # Cache upload status
    energy = estimate_energy(file.content_length / (1024 * 1024)) # Calculate energy usage
    return f'File uploaded to Azure: {file.filename}, Energy: {energy:.4f} kWh', 200 # Return success message
```
- In Azure Portal: Create container my-free-tier-container.
- Test:

```
curl -X POST -F "file=@test100mb.txt"
http://localhost:5000/upload/azure # Test Azure upload
```



Name	Last modified	Access tier	Blob type	Size	Le...
BL1.pdf	28/06/2025, 21:26:04	Hot (Inferred)	Block blob	2.81 MiB	Av...
HappyFace...	24/06/2025, 14:53:15	Hot (Inferred)	Block blob	128.2 KiB	Av...
test100mb.txt	22/06/2025, 12:17:13	Hot (Inferred)	Block blob	0	Av...

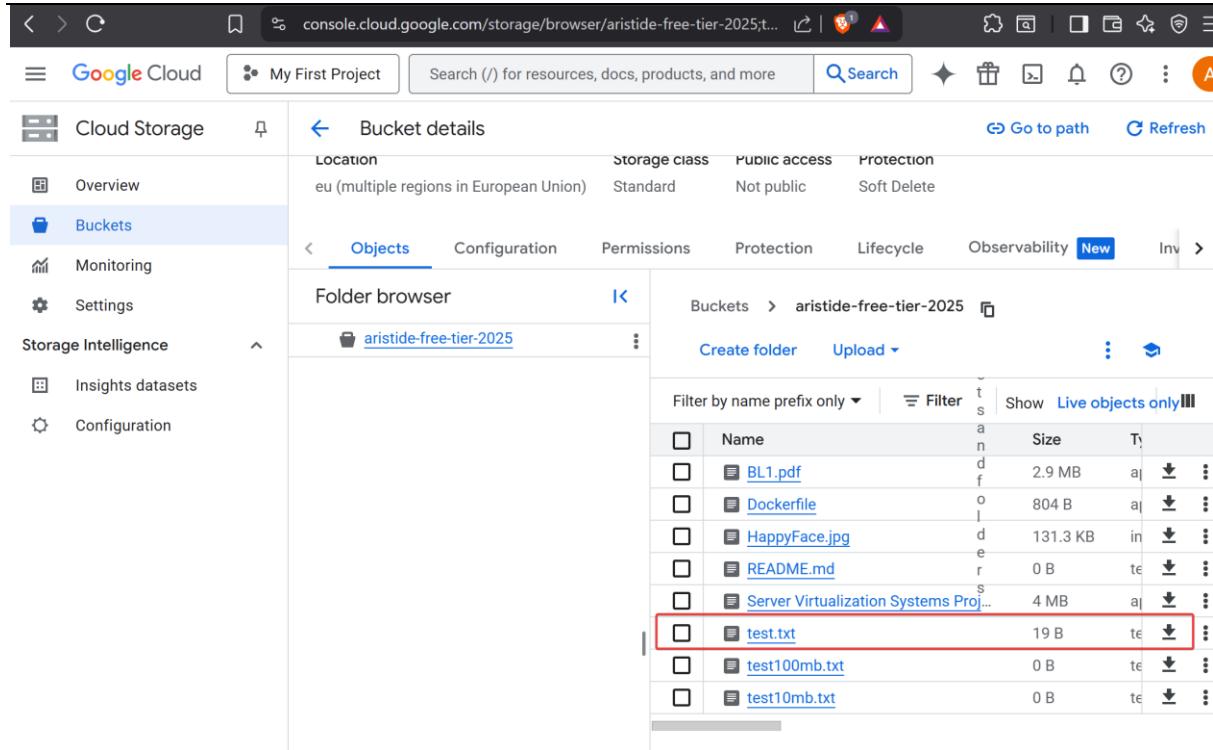
Figure 16: Azure Portal showing my-free-tier-container with test100mb.txt.

16. Integrate GCP Cloud Storage:

- Add to app.py:


```
from file_upload import upload_to_gcp # Import GCP upload function
@app.route('/upload/gcp', methods=['POST']) # Define GCP upload endpoint
def upload_gcp(): # GCP upload handler
    if 'file' not in request.files: # Check if file is included
        return 'No file provided', 400 # Return error if no file
    file = request.files['file'] # Get uploaded file
    file_path = f'temp/{file.filename}' # Define temporary file path
    cache_key = f'upload_gcp_{file.filename}' # Create cache key for file
    if get_cached(cache_key): # Check if file is already uploaded
        return 'File already uploaded', 200 # Return cached response
    file.save(file_path) # Save file to temporary path
    upload_to_gcp(file_path, 'my-free-tier-bucket', file.filename) # Upload to GCP
    set_cached(cache_key, 'Uploaded') # Cache upload status
    energy = estimate_energy(file.content_length / (1024 * 1024)) # Calculate energy usage
    return f'File uploaded to GCP: {file.filename}, Energy: {energy:.4f} kWh', 200 # Return success message
```
- In GCP Console: Create bucket my-free-tier-bucket (Standard class).
- Test:

```
curl -X POST -F "file=@test.txt"
http://localhost:5000/upload/gcp # Test GCP upload
```



The screenshot shows the Google Cloud Storage console. On the left, there's a sidebar with 'Google Cloud' at the top, followed by 'Cloud Storage', 'Overview', 'Buckets' (which is selected), 'Monitoring', 'Settings', 'Storage Intelligence' (with 'Insights datasets' and 'Configuration' options), and 'New'. The main area is titled 'Bucket details' for 'aristide-free-tier-2025'. It shows the location as 'eu (multiple regions in European Union)', storage class as 'Standard', public access as 'Not public', and protection as 'Soft Delete'. Below this, there are tabs for 'Objects', 'Configuration', 'Permissions', 'Protection', 'Lifecycle', and 'Observability'. The 'Objects' tab is active, showing a 'Folder browser' for 'aristide-free-tier-2025'. It lists several files: 'BL1.pdf', 'Dockerfile', 'HappyFace.jpg', 'README.md', 'Server Virtualization Systems Proj...', 'test.txt', 'test100mb.txt', and 'test10mb.txt'. The 'test.txt' file is highlighted with a red border.

Figure 17: GCP Console showing my-free-tier-bucket with test.txt.

17. Add Mock Endpoints and Dashboard Data:

- Add to app.py:


```

@app.route('/demo/block', methods=['GET']) # Define block
storage demo endpoint
def demo_block(): # Block storage demo handler
    return { # Return mock block storage data
        'aws': 'EBS: 10GB volume attached for VM',
        'azure': 'Managed Disk: 10GB for VM',
        'gcp': 'Persistent Disk: 10GB for VM'
    }, 200
@app.route('/demo/file', methods=['GET']) # Define file
storage demo endpoint
def demo_file(): # File storage demo handler
    return { # Return mock file storage data
        'aws': 'EFS: Shared folder created',
        'azure': 'Azure Files: Shared folder created',
        'gcp': 'Filestore: Shared folder created'
    }, 200
@app.route('/dashboard', methods=['GET']) # Define dashboard
data endpoint
def dashboard_data(): # Dashboard data handler
    return { # Return mock dashboard data
        'usage': {'s3': 100, 'azure': 150, 'gcp': 200},
        'savings': 500,
        'alerts': ['AWS S3: 80% API limit reached'],
        'energy': 0.05
    }, 200
@app.route('/provider/comparison', methods=['GET']) # Define
provider comparison endpoint
def comparison_data(): # Provider comparison handler

```

```

o      return [  # Return mock comparison data
o          {'name': 'AWS S3', 'cost': '$0.023/GB', 'freeTier':
o              '5GB', 'egress': 'Yes'},
o          {'name': 'Azure Blob', 'cost': '$0.0018/GB (Archive)', 
o              'freeTier': '5GB', 'egress': 'Yes'},
o          {'name': 'GCP Cloud Storage', 'cost': '$0.020/GB',
o              'freeTier': '5GB', 'egress': 'No'}
], 200
    
```

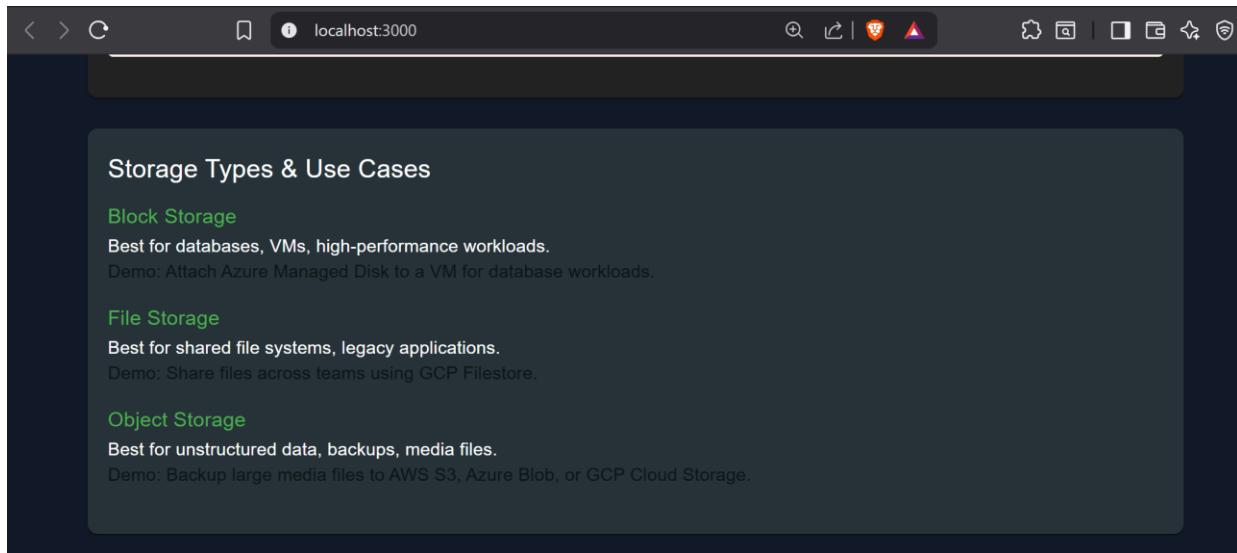


Figure 18: Browser showing mock endpoint responses in StorageInfo.js

Troubleshooting

- **CORS errors:** Ensure flask-cors is installed and enabled in app.py.
 - **API failures:** Check backend logs (python app.py), verify bucket/container names.
 - **Permission issues:** Confirm IAM roles (AmazonS3FullAccess, Storage Blob Data Contributor, Storage Admin).
 - **GCP auth errors:** Ensure key.json is correctly placed in backend.
-

8. Application Containerization (6 hours, June 9, 2025)

Objective: Containerize Flask backend and React frontend.

Steps

18. Install Docker:

- Download Docker Desktop from docker.com.
- Enable Docker service (Linux: systemctl start docker).
- Verify:

```
docker --version # Check Docker version
```

✓ TERMINAL
 ● PS C:\Users\akata\Desktop\Talent_Exhibition_Project_Aristide> docker -v
 Docker version 28.1.1, build 4eba377

Figure 19: Terminal showing docker --version, Docker Desktop UI.

19. Dockerfile for Flask:

- Create backend/Dockerfile:


```
FROM python:3.9-slim # Use Python 3.9 slim base image
WORKDIR /app # Set working directory
COPY . . # Copy all files to container
RUN pip install -r requirements.txt # Install Python dependencies
CMD ["python", "app.py"] # Run Flask app
```
- Create backend/.dockerignore:


```
venv # Ignore virtual environment
__pycache__ # Ignore Python bytecode
*.pyc # Ignore compiled Python files
temp # Ignore temporary files
key.json # Ignore GCP key
```
- Build and test:


```
cd backend # Navigate to backend directory
docker build -t flask-backend . # Build Docker image
docker run -p 5000:5000 --env-file .env -v
$(pwd)/key.json:/app/key.json flask-backend # Run container
```
- Verify at <http://localhost:5000>.

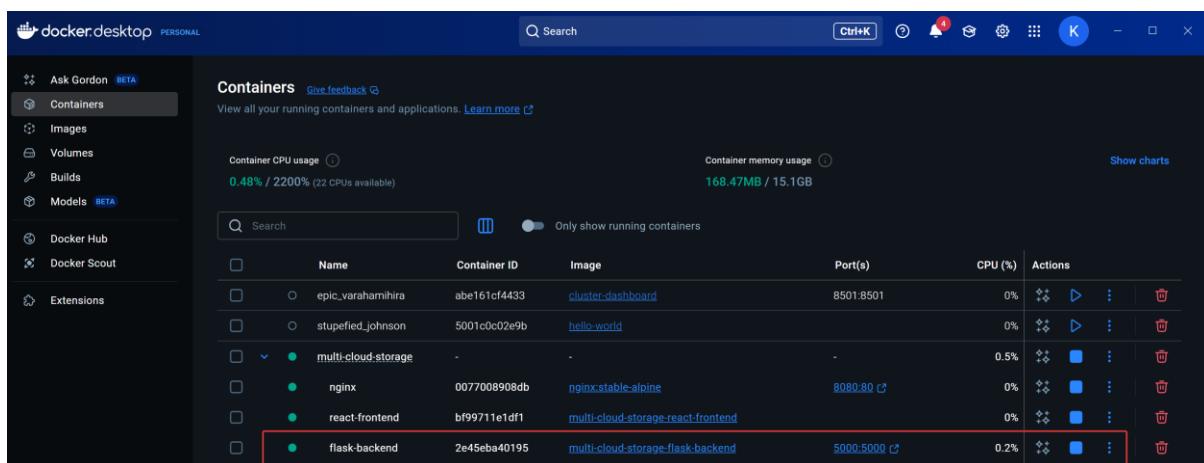


Figure 20:Docker desktop showing Flask backend on port:5000:5000

20. Dockerfile for React:

- Create storage-app/Dockerfile:


```
FROM node:16 # Use Node.js 16 base image
WORKDIR /app # Set working directory
COPY . . # Copy all files to container
RUN npm install # Install Node.js dependencies
RUN npm run build # Build React app
FROM nginx:alpine # Use Nginx alpine base image
COPY --from=0 /app/build /usr/share/nginx/html # Copy React build to Nginx
EXPOSE 80 # Expose port 80
CMD ["nginx", "-g", "daemon off;"] # Run Nginx
```
- Create storage-app/.dockerignore:


```
node_modules # Ignore Node.js dependencies
build # Ignore React build output
```
- Build and test:


```
cd storage-app # Navigate to React project
docker build -t react-frontend . # Build Docker image
docker run -p 80:80 react-frontend # Run container
```
- Verify at <http://localhost:80>.

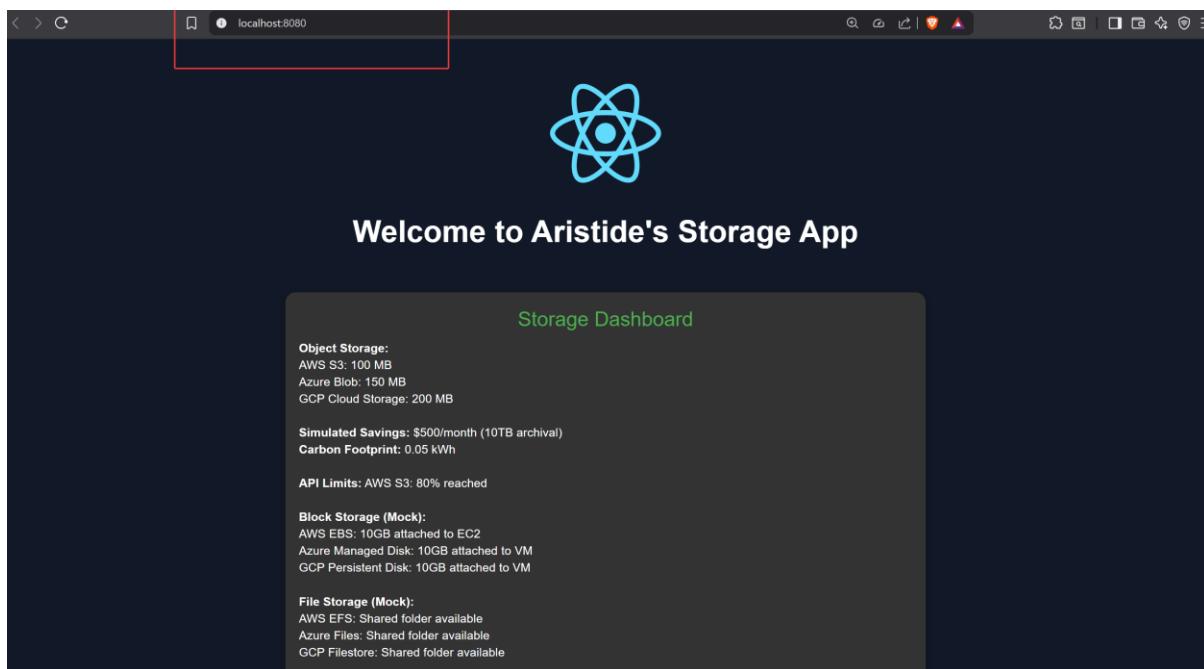


Figure 21: WebUI showing browser at <http://localhost:80>.

21. Test Containers Locally:

- Test Flask and React containers, verify API calls with Postman.

Troubleshooting

- **Port conflicts:** Check with lsof -i :80, change ports (e.g., -p 8080:80).
 - **Build failures:** Verify dependencies in requirements.txt or package.json.
 - **GCP auth:** Ensure key.json is mounted correctly in Docker.
-

9. Multi-Cloud Simulation with Load Balancing (6 hours, June 10, 2025)

Objective: Simulate multi-cloud environments with Docker Compose and Nginx.

Steps

22. Set Up Docker Compose:

- Create docker-compose.yml:

```
version: '3' # Specify Docker Compose version
services:
  flask-backend: # Backend service
    build: ./backend # Build from backend directory
    ports:
      - "5000:5000" # Map host port 5000 to container port 5000
    environment:
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID} # AWS access key
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY} # AWS secret key
      -
      AZURE_STORAGE_CONNECTION_STRING=${AZURE_STORAGE_CONNECTION_STRING} # Azure connection string
      - GOOGLE_APPLICATION_CREDENTIALS=/app/key.json # GCP key path
    volumes:
      - ./backend:/app # Mount backend directory
      - ./backend/key.json:/app/key.json # Mount GCP key
  react-frontend: # Frontend service
    build: ./storage-app # Build from storage-app directory
    ports:
      - "80:80" # Map host port 80 to container port 80
  redis: # Redis service
    image: redis:latest # Use latest Redis image
    ports:
      - "6379:6379" # Map host port 6379 to container port 6379
  nginx: # Nginx service
    image: nginx:latest # Use latest Nginx image
    ports:
      - "8080:80" # Map host port 8080 to container port 80
    volumes:
      - ./backend:/app # Mount volumes
```

- - ./nginx.conf:/etc/nginx/nginx.conf # Mount Nginx config
- - ./storage-app/build:/usr/share/nginx/html # Mount React build
- depends_on: # Specify dependencies
 - flask-backend # Ensure backend starts first
- Run:


```
docker-compose up --build # Build and start all services
```
- Access frontend at <http://localhost:8080>, backend at <http://localhost:5000>.

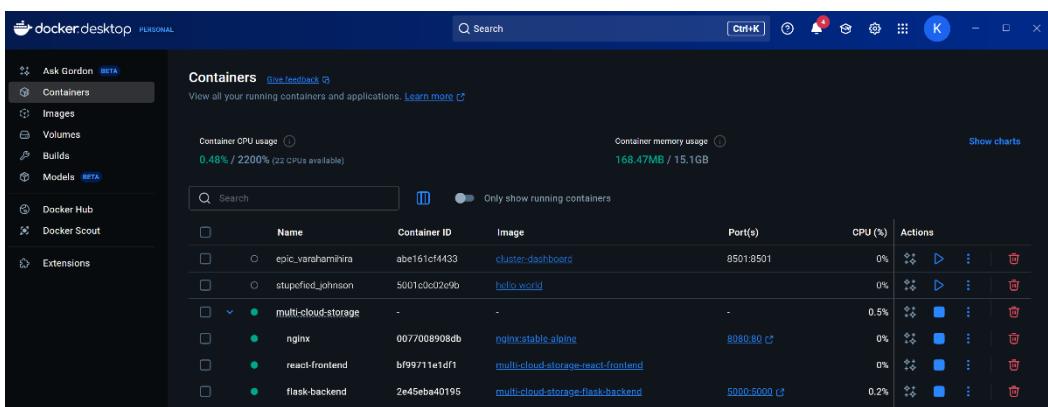


Figure 22: Docker desktop showing docker-compose up at <http://localhost:8080>.

23. Configure Nginx Load Balancing:

- Create nginx.conf:


```
events {} # Define Nginx events block (empty for default settings)
http { # Define HTTP server block
  upstream backend { # Define upstream backend server
    server flask-backend:5000; # Route to Flask backend on port 5000
  }
  server { # Define server block
    listen 80; # Listen on port 80
    location /api/ { # Route API requests
      proxy_pass http://backend; # Forward to Flask backend
      proxy_set_header Host $host; # Pass host header
      proxy_set_header X-Real-IP $remote_addr; # Pass client IP
    }
    location / { # Route all other requests
      root /usr/share/nginx/html; # Serve React build
      try_files $uri /index.html; # Fallback to index.html for SPA
    }
  }
}
```
- Test file uploads via frontend, verify routing to /upload/s3, /upload/azure, /upload/gcp.

Troubleshooting

- **Nginx errors:** Check docker logs nginx, ensure flask-backend is running.
- **404 errors:** Verify nginx.conf paths and service names.
- **CORS issues:** Confirm flask-cors in app.py.

10. Sustainability and Alerts Implementation (8 hours, June 12, 2025)

Objective: Add energy estimation, lifecycle policies, and API usage alerts.

Steps

24. Energy Estimation Script:

- Create backend/energy_estimator.py:
- def estimate_energy(file_size_mb): # Function to estimate energy usage
 - # Assume 0.01 kWh/GB upload
 - energy_kwh = (file_size_mb / 1024) * 0.01 # Convert MB to GB and calculate energy
 - return energy_kwh # Return estimated energy in kWh

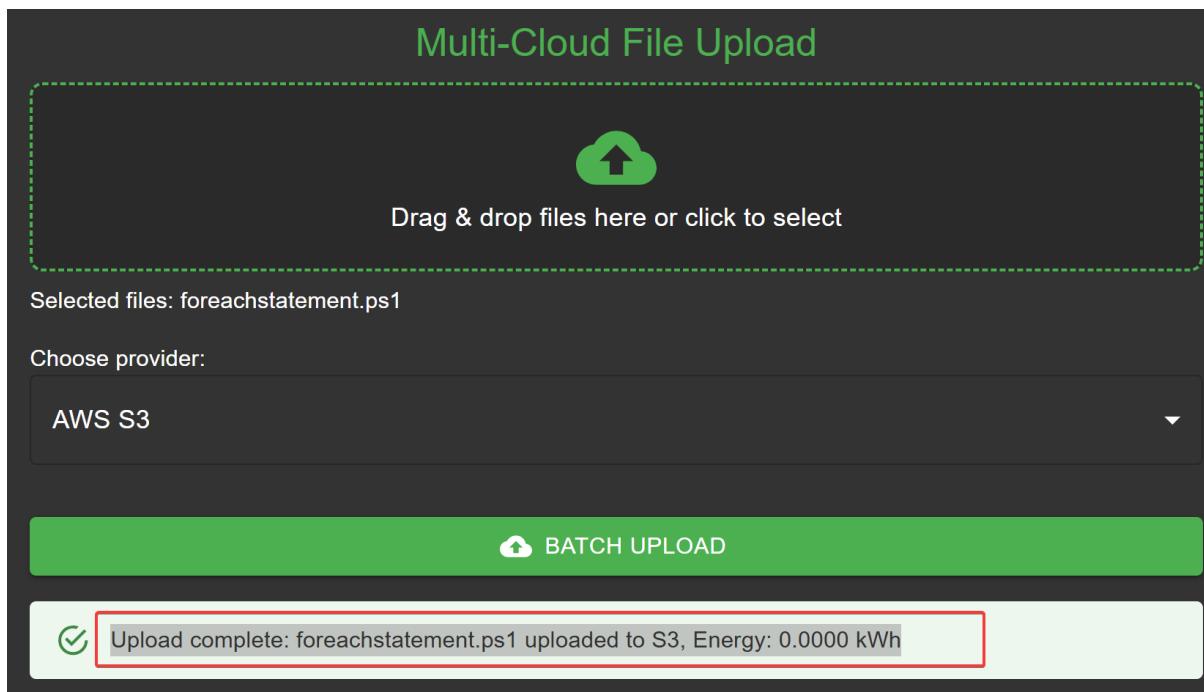


Figure 23: Browser showing upload response with energy estimation.

25. Lifecycle Policies:

- AWS S3:

- In AWS Console: S3 > my-free-tier-bucket > Management > Create lifecycle rule: ArchiveRule, transition to Glacier after 30 days.

The screenshot shows the AWS S3 console interface for creating a lifecycle rule. The URL in the address bar is `Amazon S3 > Buckets > aristide-free-tier-bucket > Lifecycle configuration > Create lifecycle rule`. The page is titled "Create lifecycle rule".

Lifecycle rule configuration

- Lifecycle rule name:** ArchiveRule
- Choose a rule scope:**
 - Limit the scope of this rule using one or more filters
 - Apply to all objects in the bucket
- Filter type:** You can filter objects by prefix, object tags, object size, or whatever combination suits your usecase.
- Prefix:** Add filter to limit the scope of this rule to a single prefix. Enter prefix: `Enter prefix`
- Object tags:** You can limit the scope of this rule to the key/value pairs added below. Add tag: `Add tag`
- Object size:** You can limit the scope of this rule to apply to objects based on their size. Learn more. Specify minimum object size: Specify maximum object size:

Lifecycle rule actions

Transitions are charged per request. For a lifecycle transition action, each request corresponds to an object transition. For details on lifecycle transition pricing, see requests pricing info on the requests & requests tab of the Amazon S3 pricing page. I acknowledge that this lifecycle rule will incur a transition cost per request.

- By default, objects less than 128KB will not transition across any storage class.** We don't recommend transitioning objects less than 128 KB because the transition costs can outweigh the storage savings. If your use case requires transitioning objects less than 128 KB, specify a minimum object size filter for each applicable lifecycle rule with a transition action.
- Transition current versions of objects between storage classes** Choose transitions to move current versions of objects between storage classes listed on your use case scenario and performance access requirements. These transitions start from when the objects are created and are consecutively applied. Learn more.
- Choose storage class transition:** Glacier Instant Retrieval → Days after object creation: 30 → **Next**
- Expire current versions of objects** For version-enabled buckets, Amazon S3 adds a delete marker and the current version of an object is retained as a noncurrent version. For non-versioned buckets, Amazon S3 permanently removes the object. Learn more.
- Days after object creation:** Enter number of days: `Expiration is required for the selected action. Enter a value or delete the action.`
- Review transition and expiration actions**
- Current version actions:** Days 0
- Noncurrent versions actions:** Days 0

The screenshot shows two screenshots of the AWS S3 Lifecycle Configuration interface.

Top Screenshot: A 'Create lifecycle rule' dialog box is open. It shows the 'Scope' section with 'Apply to all objects in the bucket' selected. Below it, the 'Lifecycle rule actions' section is highlighted with a red border. It contains several options: 'Transition current versions of objects between storage classes' (selected), 'Transition noncurrent versions of objects between storage classes', 'Delete current versions of objects', 'Delete noncurrent versions of objects', and 'Delete expired object delete marks or incomplete multipart uploads'. A note at the bottom states: 'Transitions are charged per request. For more information about lifecycle transition actions, each request corresponds to an object transition. For details on lifecycle transition pricing, see requests pricing info on the requests pricing info on the Storage & requests tab of the Amazon S3 pricing page.' A checkbox 'I acknowledge that this lifecycle rule will incur a transition cost per request' is checked.

Bottom Screenshot: The 'Lifecycle configuration' page shows a success message: 'The rule "ArchiveRule" has been successfully added and the lifecycle configuration has been updated. It may take some time for the configuration to be updated. Refresh the lifecycle rules list if changes to the configuration aren't displayed.' Below this, the 'Lifecycle rules' table is shown. It has one row with the following data:

Lifecycle rule name	Status	Scope	Current version actions	Noncurrent versions actions	Expired object delete actions	Incomplete multipart upload actions
ArchiveRule	Enabled	Entire bucket	Transition to Glacier Instant Ret.	-	-	-

Figure 24:AWS Console showing lifecycle rule.

- **Azure Blob Storage:**

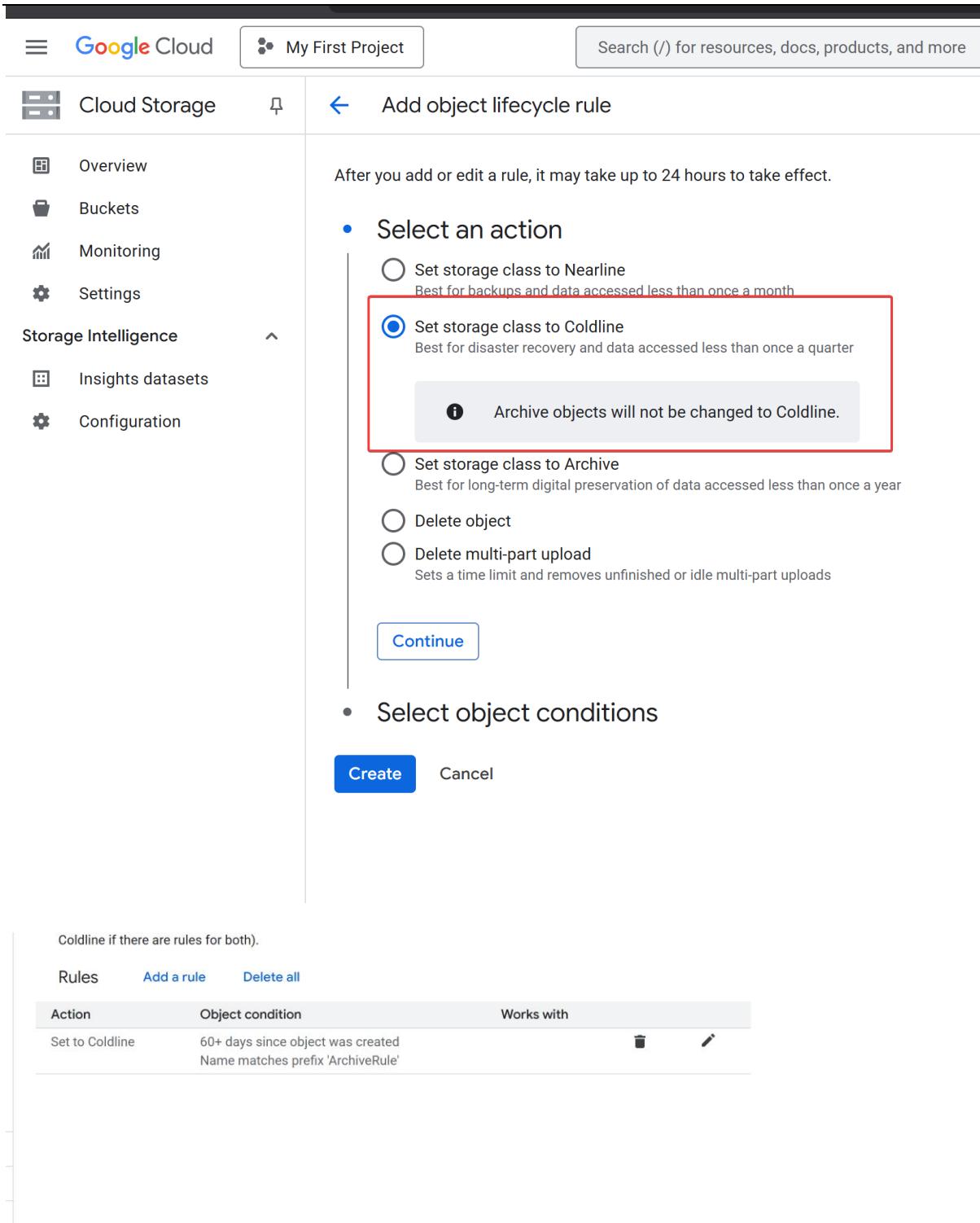
- In Azure Portal: Storage Account > Lifecycle Management > Add rule: ArchiveRule, move to Archive after 90 days.

The screenshot shows the Azure Storage Account Lifecycle management interface. At the top, there's a navigation bar with 'Microsoft Azure', a search bar, and a 'Copilot' button. Below the navigation is a breadcrumb trail: Home > aristidestorage2025. The main area is titled 'aristidestorage2025 | Lifecycle management'. On the left, a sidebar has sections like 'Data management', 'Redundancy', 'Object replication', and 'Lifecycle management', with 'Lifecycle management' being the active tab and highlighted with a red box. The main content area has tabs for 'List View' (selected) and 'Code View'. It shows a table with columns: Name, Status, and Blob type. One row is listed: 'ArchiveRule' (Status: Enabled, Blob type: Block). A red box highlights the 'ArchiveRule' name in the table.

Name	Status	Blob type
ArchiveRule	Enabled	Block

Figure 25: Azure Portal showing lifecycle rule.

- **GCP Cloud Storage:**
 - In GCP Console: Cloud Storage > my-free-tier-bucket > Lifecycle > Add rule: ArchiveRule, transition to Coldline after 60 days.



After you add or edit a rule, it may take up to 24 hours to take effect.

- Select an action**
 - Set storage class to Nearline
Best for backups and data accessed less than once a month
 - Set storage class to Coldline
Best for disaster recovery and data accessed less than once a quarter

i Archive objects will not be changed to Coldline.
 - Set storage class to Archive
Best for long-term digital preservation of data accessed less than once a year
 - Delete object
 - Delete multi-part upload
Sets a time limit and removes unfinished or idle multi-part uploads

Continue

- Select object conditions**

Create **Cancel**

Coldline if there are rules for both).

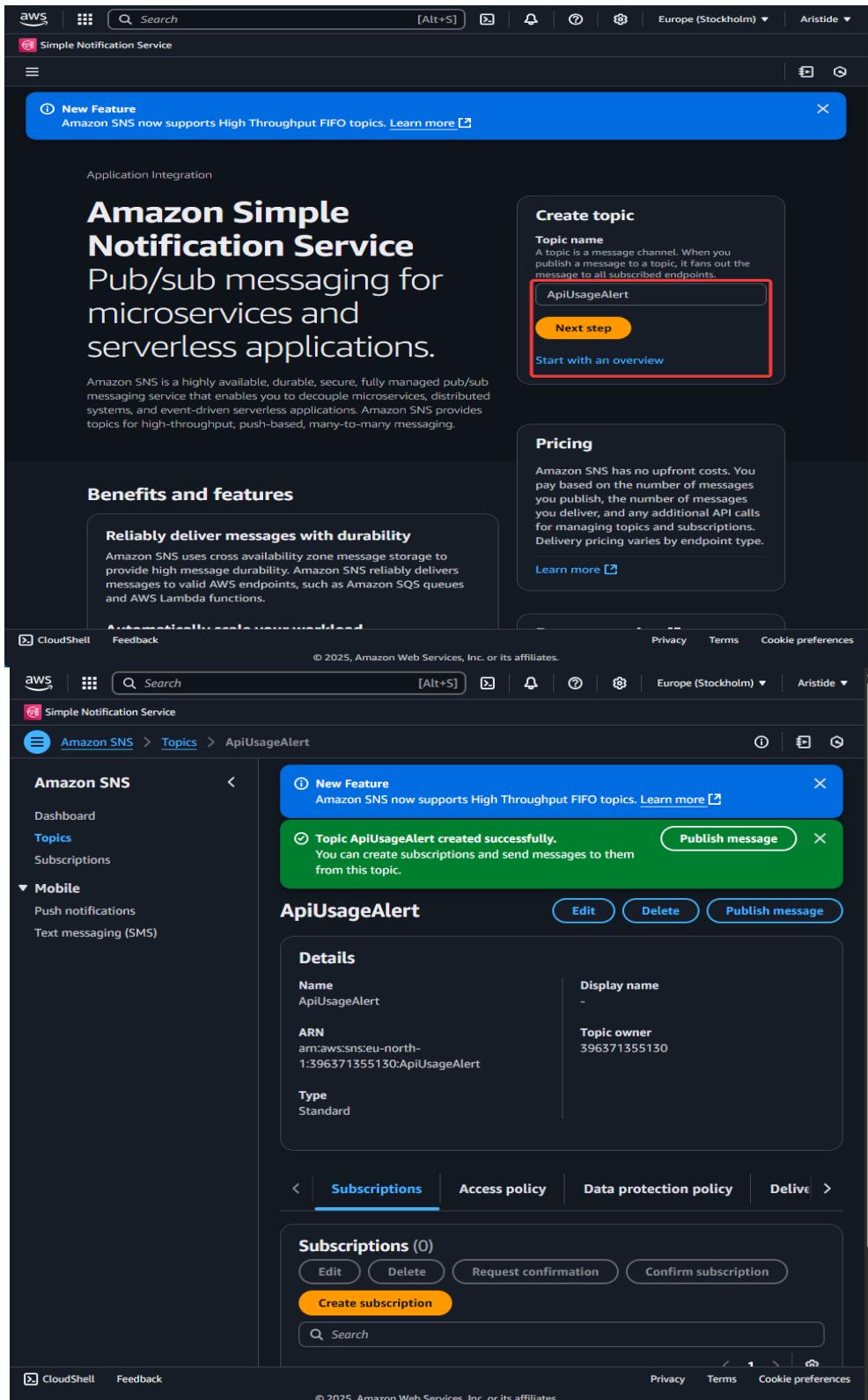
Rules	Add a rule	Delete all
Action	Object condition	Works with
Set to Coldline	60+ days since object was created Name matches prefix 'ArchiveRule'	

Figure 26: GCP Console showing lifecycle rule.

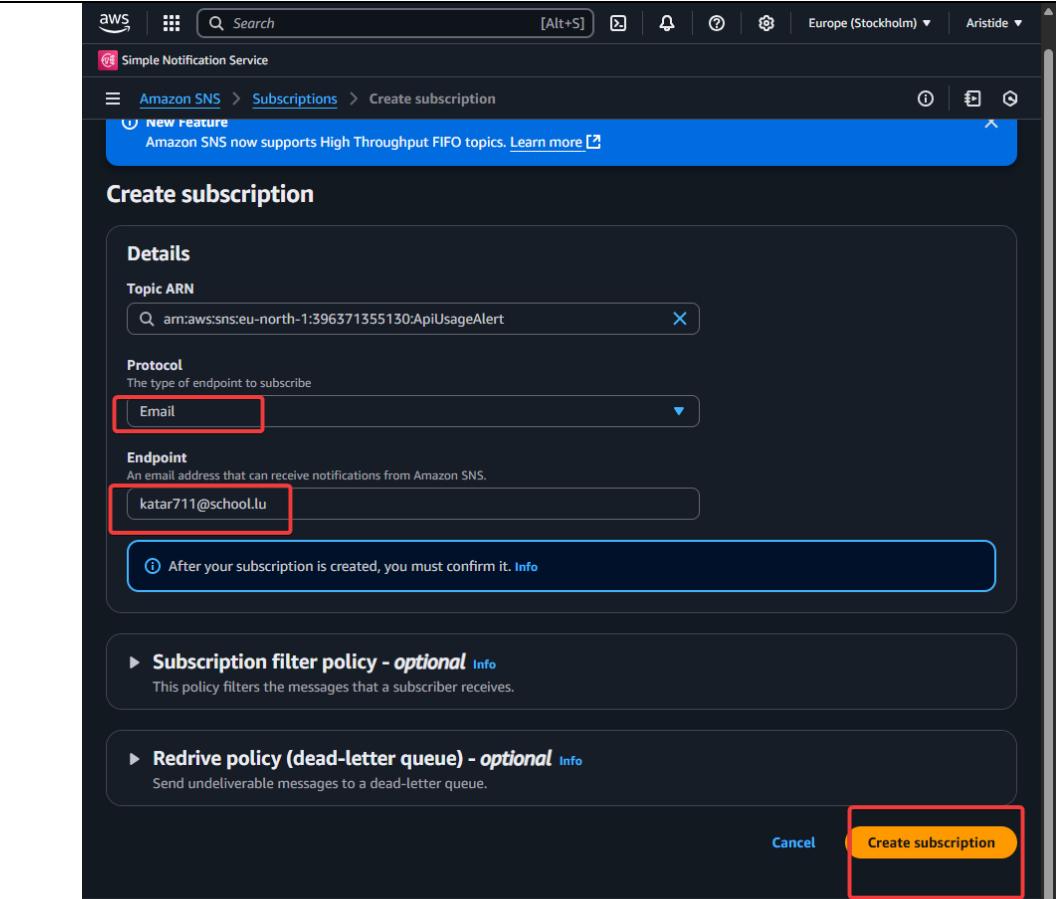
26. API Usage Alerts:

- **AWS SNS/CloudWatch:**
 - In AWS Console: SNS > Topics > Create Topic: ApiUsageAlert, add email subscription.

- In CloudWatch: Create alarm for S3 NumberOfRequests > 16,000 (80% of 20,000 GET limit), notify ApiUsageAlert.



The screenshot shows the AWS Simple Notification Service (SNS) console. At the top, there is a blue banner announcing "Amazon SNS now supports High Throughput FIFO topics". Below this, the main content area features a large heading "Amazon Simple Notification Service" with the subtext "Pub/sub messaging for microservices and serverless applications". To the right, a "Create topic" dialog box is open, showing a text input field for "Topic name" containing "ApiUsageAlert" and a yellow "Next step" button. A red box highlights the "Topic name" field and the "Next step" button. Further down, there is a "Pricing" section with a note about no upfront costs and delivery pricing varying by endpoint type, followed by a "Learn more" link. On the left side, a sidebar menu includes "Topics" under "Amazon SNS" and "Push notifications" under "Mobile". The main navigation bar at the top includes "CloudShell", "Feedback", "Search", and "Europe (Stockholm)". The bottom of the page includes standard AWS footer links for "Privacy", "Terms", and "Cookie preferences".



Create subscription

Details

Topic ARN
arn:aws:sns:eu-north-1:396371355130:ApiUsageAlert

Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
katar711@school.lu

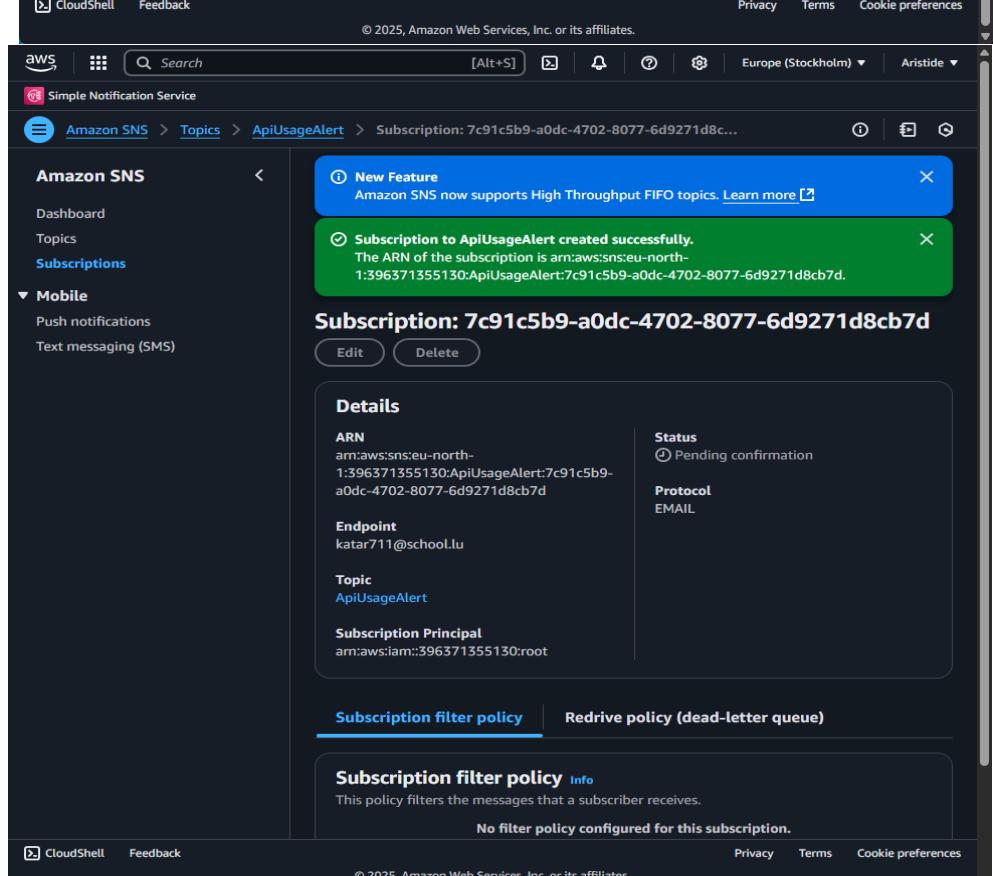
After your subscription is created, you must confirm it. [Info](#)

Subscription filter policy - optional [Info](#)
This policy filters the messages that a subscriber receives.

Redrive policy (dead-letter queue) - optional [Info](#)
Send undeliverable messages to a dead-letter queue.

Create subscription

CloudShell Feedback Privacy Terms Cookie preferences © 2025, Amazon Web Services, Inc. or its affiliates.



Amazon SNS

Subscriptions

Subscription: 7c91c5b9-a0dc-4702-8077-6d9271d8cb7d

Details

ARN
arn:aws:sns:eu-north-1:396371355130:ApiUsageAlert:7c91c5b9-a0dc-4702-8077-6d9271d8cb7d

Endpoint
katar711@school.lu

Topic
ApiUsageAlert

Subscription Principal
arn:aws:iam::396371355130:root

Status
Pending confirmation

Protocol
EMAIL

Subscription filter policy [Info](#)
This policy filters the messages that a subscriber receives.

No filter policy configured for this subscription.

CloudShell Feedback Privacy Terms Cookie preferences © 2025, Amazon Web Services, Inc. or its affiliates.

Bucket metrics Info

Explore metrics for usage, request, and data transfer activity within your bucket. Metrics are also available in Amazon CloudWatch. [Learn more](#)

Storage metrics **Request metrics**

The metrics are available at 1-minute intervals after some latency to process. You can monitor S3 requests for your bucket to quickly identify and act on operational issues.

Filters

To see your request metrics, you must choose a filter.

You don't have any filters

To view request metrics, create a filter.

[Create filter](#)

[Manage filters](#)

Name

Filter name

This can't be edited after the filter is created.

Filter names can only contain letters, numbers, periods, dashes, and underscores.

Scope

You can create a filter that includes all the objects in your bucket. Or you can filter objects by prefix, object tags, and Access Point or a combination of all three. [Learn more](#)

Choose a filter scope

Limit the scope of this filter using prefix, object tags, and Access Point, or a combination of all three.

This filter applies to *all* objects in the bucket

Filter type

Choose at least one filter type.

Prefix

Object tags

Access Point

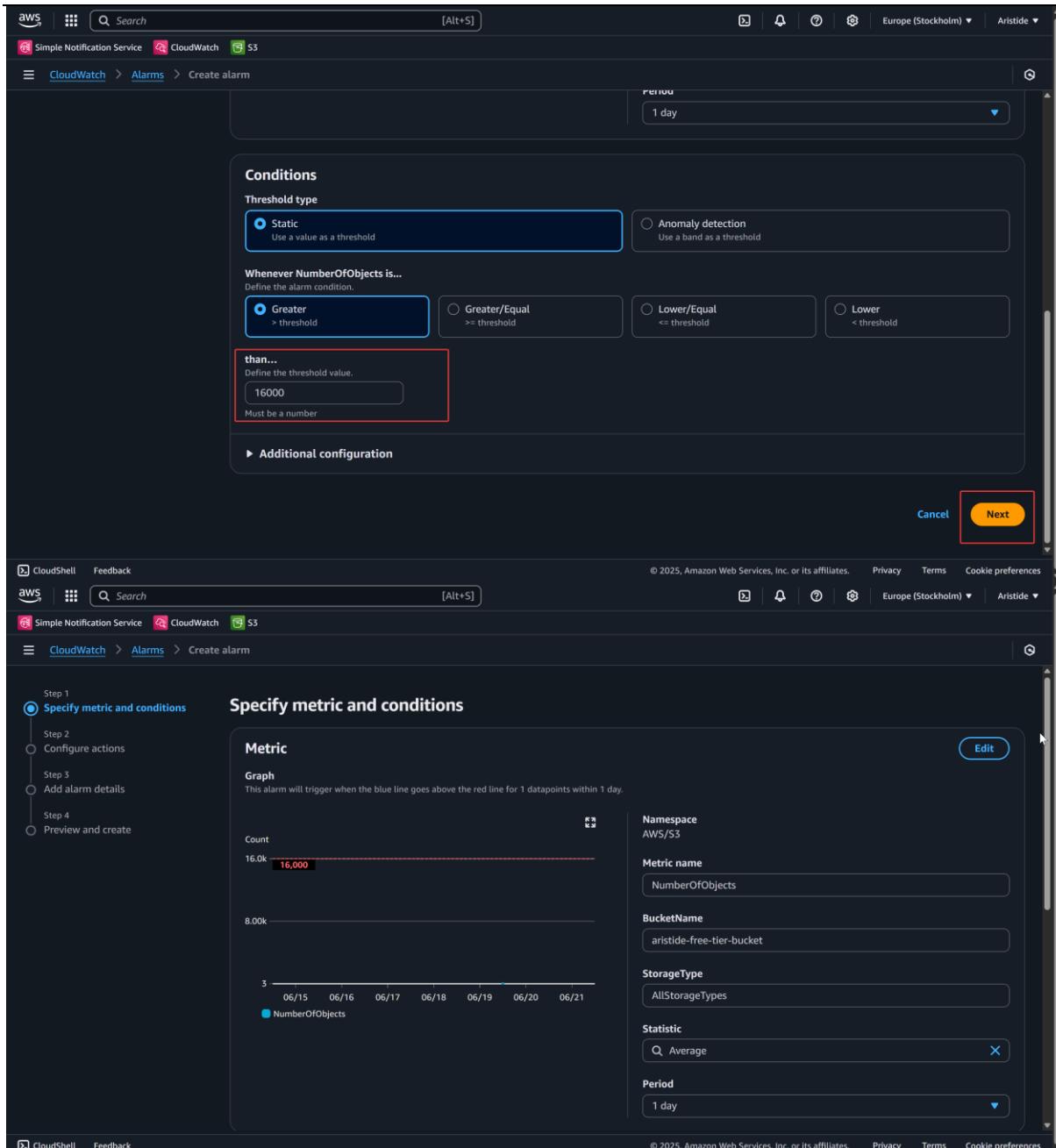
Prefix

Limit this filter to a single prefix.

Don't include the bucket name in the prefix. Using certain characters in key names can cause problems with some applications and protocols. Add a "/" to limit scope to a folder with the prefix name. [Learn more](#)

[Cancel](#) [Create filter](#)

The screenshot shows the AWS CloudWatch Metrics console. The user is on Step 1: Specify metric and conditions. The 'Select metric' button is highlighted with a red box. The graph area shows an empty line chart with the message "Your CloudWatch graph is empty. Select some metrics to appear here." Below the graph, there's a search bar and a section for "Metrics (11)" with two tabs: "Request Metrics Per Filter" (9) and "Storage Metrics" (2), which is also highlighted with a red box.



The screenshot shows the AWS CloudWatch Metrics & Alarms interface. A new alarm is being created for the metric `NumberOfObjects` from the `AWS/S3` namespace.

Conditions:

- Threshold type:** Static (selected)
- Whenever `NumberOfObjects` is...**: Greater than 16000
- than...**: 16000

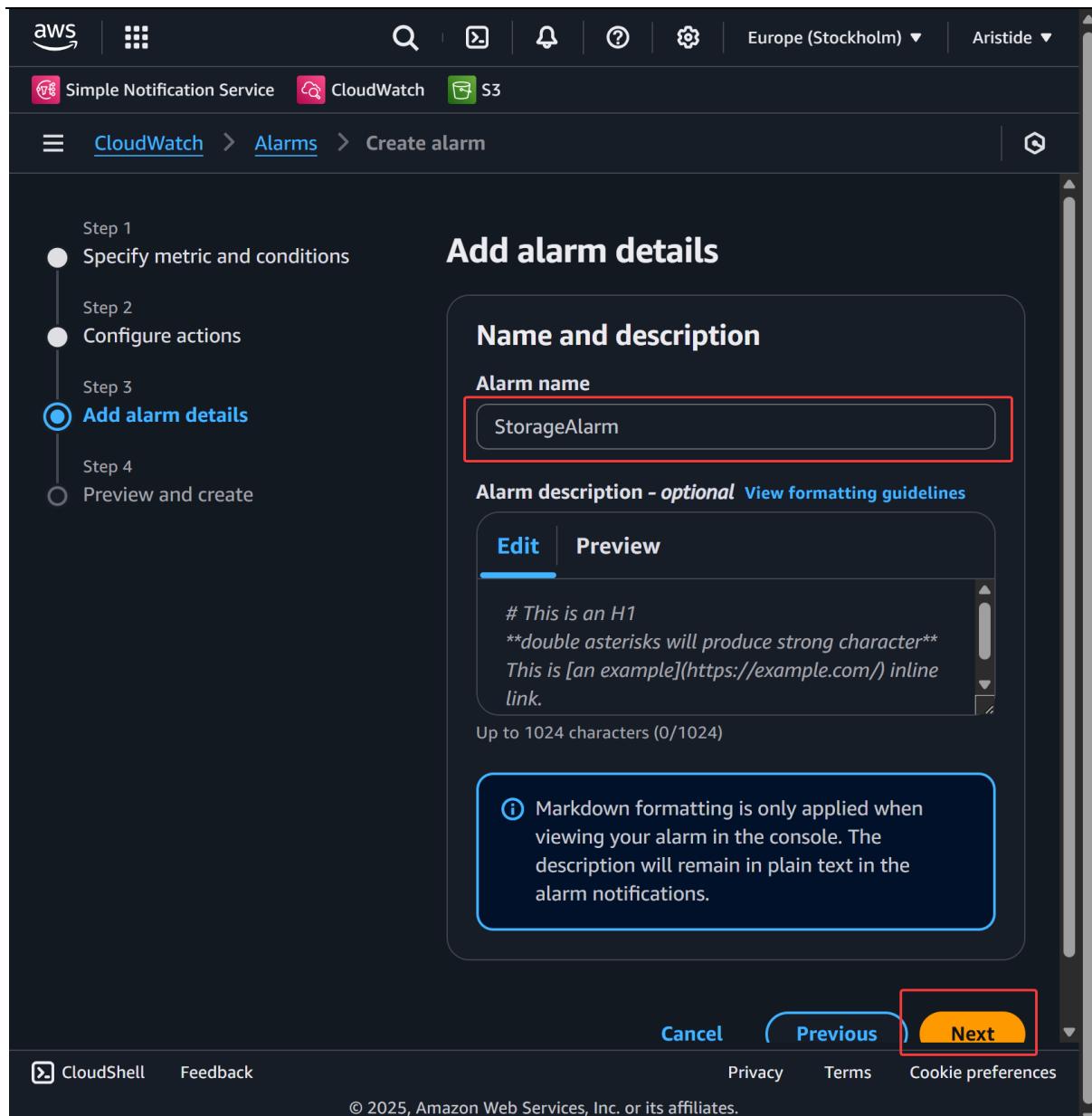
Additional configuration:

Next Step

Specify metric and conditions:

Metric:

- Graph:** Shows a line chart for the metric `NumberOfObjects` over time (06/15 to 06/21). The current value is 16.0k, and the threshold is set at 16,000.
- Namespace:** AWS/S3
- Metric name:** NumberOfObjects
- BucketName:** aristide-free-tier-bucket
- StorageType:** AllStorageTypes
- Statistic:** Average
- Period:** 1 day



The screenshot shows the AWS CloudWatch 'Create alarm' wizard at step 3: 'Add alarm details'. The left sidebar lists steps 1 through 4. Step 1 ('Specify metric and conditions') is the first unselected step. Step 2 ('Configure actions') is the second unselected step. Step 3 ('Add alarm details') is the selected step, indicated by a blue circle with a dot. Step 4 ('Preview and create') is the fourth unselected step.

Add alarm details

Name and description

Alarm name

Alarm description - optional View formatting guidelines

Edit **Preview**

```
# This is an H1
**double asterisks will produce strong character**
This is [an example](https://example.com/) inline link.
```

Up to 1024 characters (0/1024)

Markdown formatting is only applied when viewing your alarm in the console. The description will remain in plain text in the alarm notifications.

Cancel **Previous** **Next** Next

CloudShell Feedback Privacy Terms Cookie preferences

© 2025, Amazon Web Services, Inc. or its affiliates.

Preview and create

Step 1: Specify metric and conditions

Metric

Graph

This alarm will trigger when the blue line goes above the red line for 1 datapoints within 1 day.

Count

16.000

8.000

3

06/15 06/16 06/17 06/18 06/19 06/20 06/21

NumberofObjects

Namespace
AWS/S3

Metric name
NumberOfObjects

BucketName
aristide-free-tier-bucket

StorageType
AllStorageTypes

Statistic
Average

Period
1 day

Conditions

Threshold type
Static

Whenever NumberOfObjects is
Greater (>)
than...
16000

► Additional configuration

Step 2: Configure actions

Actions

Notification
When In alarm, send a notification to "ApiUsageAlert"

Step 3: Add alarm details

Alarm details

Name
StorageAlarm

Description
-

Create alarm

CloudWatch Alarms

Alarms (1)

Successfully created alarm StorageAlarm.

Actions

Name: StorageAlarm State: OK Last state update (UTC): 2025-06-22 10:13:58 Conditions: NumberOfObjects > 16000 for 1 datapoints within 1 day Actions: Actions enabled Warning

Create alarm

Figure 27: CloudWatch showing the creation of an alarm.

- **Azure Monitor:**

- In Azure Portal: Storage Account > Alerts > Create alert rule: Total Transactions > 16,000, email notification.

Storage accounts

Default Directory

+ Create ... Group by none

You are viewing a new version of Browse experience. Some features may be missing. Click here to access the old experience.

Name: aristidestorage2025

Settings

- Blob inventory
- Static website
- Lifecycle management
- Azure AI Search
- Configuration
- Data Lake Gen2 upgrade
- Resource sharing (CORS)
- Advisor recommendations
- Endpoints
- Locks
- Monitoring
- Insights
- Alerts
- Metrics
- Workbooks
- Diagnostic settings
- Logs

Show 1 - 1 of 1. Display count: auto

Set up alert rules on this resource

Get notified when important monitoring events happen on your resource.

Create alert rule

Scope Condition Actions Details Tags Review + create

Configure when the alert rule should trigger by selecting a signal and defining its logic.

Signal name * ⓘ

Transactions

We have set the condition configuration automatically based on popular settings for this metric. Please review and make changes as needed.

Threshold type ⓘ

Static Dynamic

Aggregation type ⓘ

Total

Value is ⓘ

Greater than

Unit ⓘ

Count

Threshold * ⓘ

16000

Alert logic

Preview

Whenever the total Transactions is greater than 16000

\$0.10 USD/month

Preview time range: Over the last 6 hours Time series: Aggregate

Transactions (Sum), aristidestorage2025 | 16

Review + create Previous Next: Actions >

Create an alert rule

Select action groups

Select up to five action groups to attach to this rule.

Subscription: Azure subscription 1

Action group name: NewActionGroupAlert

Contains actions: TestAlert

Location: global

Review + create **Previous** **Next: Details >** **Select**

Create action group

Basics

Subscription: Azure subscription 1
Resource group: StorageDemoRG
Region: global
Action group name: NewActionGroupAlert
Display name: TestAlert

Notifications

Notification type: Email/SMS message/Push/Voice
Name: Aristide
Selected: Email

Actions

None

Tags

None

Create **Previous**

Figure 28: Azure Portal showing the creation alert rule.

- **GCP Cloud Monitoring:**
 - In GCP Console: Monitoring > Alerting > Create policy:
storage.googleapis.com/api/request_count > 80% quota, email notification.

The screenshot shows the Google Cloud Observability Monitoring Alerting interface. The left sidebar includes sections for Overview, Dashboards, Application monitoring, Explore (Metrics explorer, Logs explorer, Log analytics, Trace explorer), Detect (Alerting, Error reporting, Uptime checks, Synthetic monitoring, SLAs), Configure (Integrations, Log-based metrics), and Observability Scopes (My First Project). The main area displays a summary of incidents (0 firing, 0 acknowledged, 0 policies) and lists for Incidents, Snoozes, and Policies. A red box highlights the 'Alerting' section in the sidebar.

Create alerting policy

Select a metric (GCS Bucket - Request count)

ALERT CONDITIONS (GCS Bucket - Request count, Configure trigger)

ALERT DETAILS (Notifications and name, Review alert)

Transform data

- Within each time series** (Rolling window: 1 min, Rolling window function: rate)
- Across time series** (Add secondary data transformation)

Estimated monthly cost (Coming 2026, \$0.10 \$0.00)

Shows cost breakdown

Uploads and My First Project operations (Dockerfile, Complete)

1 file successfully uploaded

Google Cloud My First Project alert X Q Search ⋮ 2 ? A

[Create alerting policy](#) [+ Add alert condition](#) [Delete alert condition](#) [View Code](#)

ALERT CONDITIONS

- GCS Bucket - Request count
- [Configure trigger](#)

ALERT DETAILS

- [Notifications and name](#)
- [Review alert](#)

Configure alert trigger

Condition Types

- Threshold**
Condition triggers if a time series rises above or falls below a value for a specific duration window
- Metric absence**
Condition triggers if any time series in the metric has no data for a specific duration window
- Forecast** [Preview](#)
Condition triggers if any timeseries in the metric is projected to cross the threshold in the near future.

Alert trigger: Any time series violates **Threshold position**: Above threshold
Threshold value: 0.1 /s

Advanced Options

Condition name *: GCS Bucket - Request count

Next

GCS Bucket - Request count

0.12/s
0.1/s
0.08/s
0.06/s
0.04/s
0.02/s
0 UTC+2 10:20 AM 10:30 AM 10:40 AM 10:50 AM 11:00 AM

Filter Enter property name or value

method ↑	Value
ListObjects	0.017 /s
WriteObject	0.017 /s

Estimated monthly cost [Coming 2026](#)
\$0.10 \$0.00
Pricing will be introduced in 2026. Estimates do not include taxes or fees.

[Create Policy](#) [Provide feedback](#) [Cancel](#) [Dockerfile](#) [Complete](#)

Uploads and My First Project operations

Google Cloud My First Project alert X Q Search ⋮ 2 ? A

[Create alerting policy](#) [+ Add alert condition](#) [Delete alert condition](#) [View Code](#)

ALERT CONDITIONS

- GCS Bucket - Request count
- [Configure trigger](#)

ALERT DETAILS

- [Notifications and name](#)
- [Review alert](#)

Configure notifications and finalize alert

Configure notifications Recommended

Use notification channel

Notification Channels: AristideEmailNotification

Notification subject line

We recommend that you create multiple notification channels for redundancy purposes. Google has no control of many of the delivery systems after we have passed the notification to that system. Additionally, a single Google service supports Cloud Console Mobile App, PagerDuty, Webhooks, and Slack. If you use one of these notification channels, then use email, SMS, or Pub/Sub as the redundant channel.

[Learn more](#)

Notify on incident closure

Incident autoclose duration: 7 days

If data is absent, select a duration after which Incident will automatically close.

Application labels

Linking alert policies to App Hub will enable alert policies, and alerts, to be displayed within App Hub pages, and will help link alerts back to the application.

[Create Policy](#) [Provide feedback](#) [Cancel](#) [Dockerfile](#) [Complete](#)

Uploads and My First Project operations

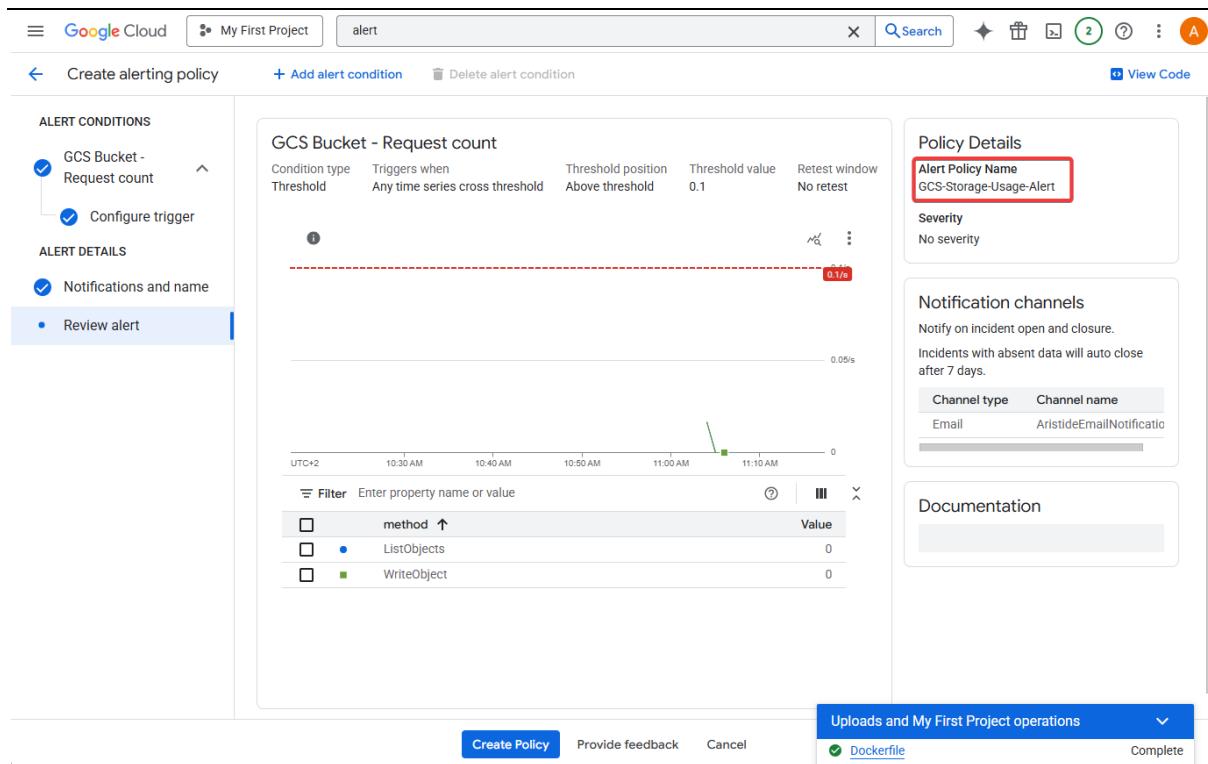


Figure 29: GCP Console showing monitoring policy.

Troubleshooting

- Alerts not triggering:** Verify thresholds, email subscriptions.
- Lifecycle issues:** Check rule application in consoles.

11. Testing File Operations and Demo Scenarios (6 hours, June 15, 2025)

Objective: Validate file operations, mock scenarios, and alerts.

Steps

27. Test File Operations:

- Upload 10MB file to AWS S3, Azure Blob, GCP Cloud Storage via frontend.
- Download file using provider-generated links (e.g., S3 pre-signed URL).
- Share file link via API.
- Test mock endpoints (/demo/block, /demo/file) via StorageInfo.js.

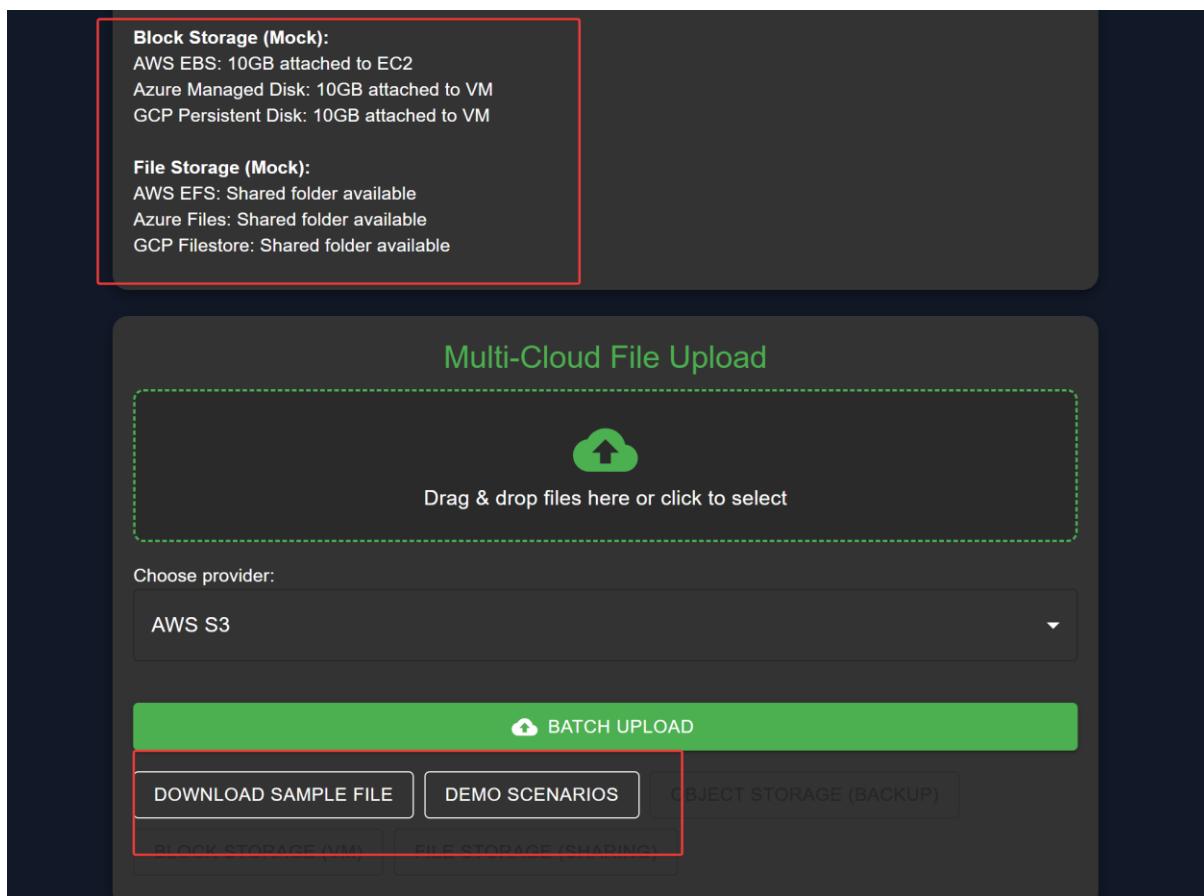


Figure 30: Browser showing upload/download and mock scenario outputs.

28. Test Load Balancing and Alerts:

- Send multiple upload requests, verify Nginx routing.
- Simulate high request counts to trigger alerts (e.g., 16,000 requests).
- Confirm email notifications.

Troubleshooting

- **Upload failures:** Check docker logs flask-backend for errors.
 - **Alerts not received:** Verify console configurations.
-

12. Performance Optimization

Objective: Implement caching and batch API calls.

Steps

29. Local Caching with Redis:

- Install Redis dependency (already in requirements.txt).
- Ensure Redis is included in docker-compose.yml.

□	Name	Container ID	Image	Port(s)	CPU (%)	Actions
□	epic_varahamihira	abe161cf4433	cluster-dashboard	8501:8501	0%	⋮
□	stupefied_johnson	5001c0c02e9b	hello-world		0%	⋮
□	multi-cloud-storage	-	-		0.46%	⋮
□	nginx	0077008908db	nginx-stable-alpine	8080:80	0%	⋮
□	react-frontend	bf99711e1df1	multi-cloud-storage-react-frontend		0%	⋮
□	flask-backend	2e45eba40195	multi-cloud-storage-flask-backend	5000:5000	0.2%	⋮
□	redis-1	e06970437521	redis:7	6379:6379	0.26%	⋮

Figure 31: Docker desktop showing Redis container running.

30. Batch API Calls:

- Update app.py:


```
@app.route('/upload/batch', methods=['POST']) # Define batch upload endpoint
def upload_batch(): # Batch upload handler
    if 'files' not in request.files: # Check if files are included
        return 'No files provided', 400 # Return error if no files
    files = request.files.getlist('files') # Get list of uploaded files
    responses = [] # Initialize response list
    for file in files: # Iterate through files
        file_path = f'temp/{file.filename}' # Define temporary file path
        cache_key = f'upload_s3_{file.filename}' # Create cache key for file
        if get_cached(cache_key): # Check if file is already uploaded
            responses.append(f'File {file.filename} already uploaded') # Add cached response
            continue # Skip to next file
        file.save(file_path) # Save file to temporary path
        upload_to_s3(file_path, 'my-free-tier-bucket', file.filename) # Upload to S3
        set_cached(cache_key, 'Uploaded') # Cache upload status
        responses.append(f'Uploaded {file.filename} to S3') # Add success response
```

```
return {'results': responses}, 200 # Return all responses
```

- Update FileUpload.js:
- import React, { useState } from 'react'; // Import React and useState hook
- import axios from 'axios'; // Import axios for API calls
- function FileUpload() { // Define FileUpload component
- const [files, setFiles] = useState([]); // State for selected files
- const [provider, setProvider] = useState('s3'); // State for selected cloud provider
- const [status, setStatus] = useState(''); // State for upload status message
- const handleFileChange = (e) => { // Handle file input change
 setFiles(e.target.files); // Update files state
 };
- const handleUpload = async () => { // Handle file upload
 if (files.length === 0) { // Check if files are selected
 setStatus('No files selected'); // Set error status
 return;
 }
 setStatus('Uploading...'); // Set uploading status
 try { // Handle API call
 const formData = new FormData(); // Create FormData for file upload
 for (let i = 0; i < files.length; i++) { // Loop through files
 formData.append('files', files[i]); // Append each file
 }
 const endpoint = files.length > 1 ? '/api/upload/batch' :
 `/api/upload/\${provider}`; // Choose endpoint
 const response = await axios.post(endpoint, formData);
 // Send POST request
 setStatus(files.length > 1 ?
 response.data.results.join(', ') : response.data); // Set status
 } catch (error) { // Handle errors
 setStatus(`Upload failed: \${error.message}`); // Set error status
 }
 };
 return (// Render JSX
 <div className="m-4 bg-white p-4 rounded shadow"> // Container with styling
 <h2 className="text-xl font-semibold text-gray-800">File Upload</h2> // Section title
 <select
 value={provider} // Bind provider state
 onChange={(e) => setProvider(e.target.value)} // Update provider state
 className="mb-2 border rounded p-2" // Styling for dropdown
 >
 <option value="s3">AWS S3</option> // AWS option
 <option value="azure">Azure Blob</option> // Azure option

```

o          <option value="gcp">GCP Cloud Storage</option> // GCP
option
o      </select>
o      <input
o          type="file" // File input
o          multiple // Allow multiple file selection
o          onChange={handleFileChange} // Bind file change
handler
o          className="mb-2 border rounded p-2" // Styling for
input
o      />
o      <button
o          onClick={handleUpload} // Bind upload handler
o          className="bg-blue-500 text-white p-2 rounded hover:bg-
blue-600" // Styling for button
o      >
o          Upload Files // Button text
o      </button>
o      <p className="mt-2 text-gray-700">{status}</p> //
Display status message
o      </div>
o  );
o }
export default FileUpload; // Export FileUpload component

```

Troubleshooting

- **Redis failures:** Verify Redis container (docker ps).
 - **Batch upload issues:** Test with small files, check backend logs (docker logs flask-backend).
-

13. Documentation and Presentation (9 hours, June 17, 2025)

Objective: Create comprehensive documentation and presentation materials.

Steps

31. User Guide and Technical Docs:

- **Create docs/user-guide.md:**
 - # User Guide
 - ## Installation
 - 1. Clone repository: `git clone https://github.com/yourusername/Multi-Cloud-Storage.git` # Clone project
 - 2. Set environment variables in `backend/.env`:


```
AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY,
AZURE_STORAGE_CONNECTION_STRING,
GOOGLE_APPLICATION_CREDENTIALS. # Configure credentials
```
 - 3. Place GCP key in `backend/key.json`. # Add GCP service account key
 - 4. Run: `docker-compose up --build` # Build and start services
 - ## Usage
 - Access frontend: http://localhost:8080 # Open web interface
 - Upload files to AWS, Azure, GCP. # Use file upload component
 - View dashboard, storage info, and provider comparison. # Explore UI components
 - Run mock block/file storage scenarios. # Test demo endpoints
- **Create docs/technical-docs.md:**
 - # Technical Documentation
 - ## Architecture
 - **Frontend**: React with Tailwind CSS # Responsive UI framework
 - **Backend**: Flask with AWS S3, Azure Blob, GCP Cloud Storage APIs # API server
 - **Containers**: Docker with Nginx, Redis # Containerized services
 - ## APIs
 - `/upload/s3`, `/upload/azure`, `/upload/gcp`: File uploads # Upload endpoints
 - `/upload/batch`: Batch upload # Batch upload endpoint
 - `/demo/block`, `/demo/file`: Mock scenarios # Demo endpoints
 - `/dashboard`: Usage, savings, alerts, energy # Dashboard data
 - `/provider/comparison`: Provider comparison data # Comparison data

The screenshot shows two code editors side-by-side in VS Code. The left editor contains the file `user-guide.md`, which is a user guide for a multi-cloud storage system. It includes sections for installation, building the React frontend, copying environment variables, starting Docker services, and usage instructions for both production and development environments. The right editor contains the file `technical-docs.md`, which provides technical documentation on the system's architecture, APIs, and file structure. Both files use a monospaced font and include line numbers.

```

user-guide.md
1 # User Guide
2
3 ## Installation
4 1. Clone the repository:
5   ```bash
6     git clone https://github.com/Aris-Kata26/Multi-Cloud-Storage
7     cd Multi-Cloud-Storage
8   ```
9
10 2. Build the React frontend:
11   ```bash
12     cd storage-app
13     npm install
14     npm run build
15     cd ..
16   ```
17
18 3. Copy or update your environment variables in `backend/.env` (see `*.env.example` if available).
19
20 4. Start all services with Docker Compose:
21   ```bash
22     docker-compose up --build -d
23   ```
24
25 5. Make sure Docker Desktop is running before starting the containers.
26
27 ## Usage
28
29 - **Frontend (Production):**
30   Open [http://localhost:8080](http://localhost:8080) in your browser to use the app.
31
32 - **Frontend (Development):**
33   You can run the React dev server for hot reload:
34   ```bash
35     cd storage-app
36     npm start
37   ```

technical-docs.md
1 # Technical Documentation
2
3 ## Architecture
4 - **Frontend**: React with Tailwind CSS (currently using Material-UI and Emotion for styling).
5 - **Backend**: Flask with integration for AWS S3, Azure Blob, and GCP Cloud Storage.
6 - **Containers**: Docker with Nginx (serves the production React build), Flask backend, Redis (for caching), and React frontend (for building).
7 - **Networking**: All services are connected via a custom Docker bridge network.
8 - **State/Cache**: Redis is used for caching upload status and optimizing API calls.
9
10 ## APIs
11
12 ### Upload APIs
13 - `/upload/batch`: Upload files to the selected provider (AWS S3, Azure Blob, or GCP Storage).
14   - Method: `POST`
15   - Parameters: `files` (multipart form data), `provider` (e.g., `s3`, `azure`, `gcp`)
16   - Response: JSON with success message or error.
17
18 - `/upload/s3`: Upload a single file to AWS S3.
19   - Method: `POST`
20   - Parameters: `file` (multipart form data)
21   - Response: Success message or error.
22
23 - `/upload/azure`: Upload a single file to Azure Blob Storage.
24   - Method: `POST`
25   - Parameters: `file` (multipart form data)
26   - Response: Success message or error.
27
28 - `/upload/gcp`: Upload a single file to Google Cloud Storage.
29   - Method: `POST`
30   - Parameters: `file` (multipart form data)
31   - Response: Success message or error.
32
33 ### Download & Sharing APIs
34 - `/download/s3/<filename>`: Download a file from AWS S3 (not yet implemented).
35 - `/share/s3/<filename>`: Get a presigned URL for sharing a file from S3.
36   - Method: `GET`
37   - Response: JSON with presigned URL or error.
38
39 ### Health Check
40 - `/`: Returns "Backend is running!" for health checks.
41
42 ## Caching & Optimization
43 - **Redis** is used to cache upload status and reduce redundant uploads.

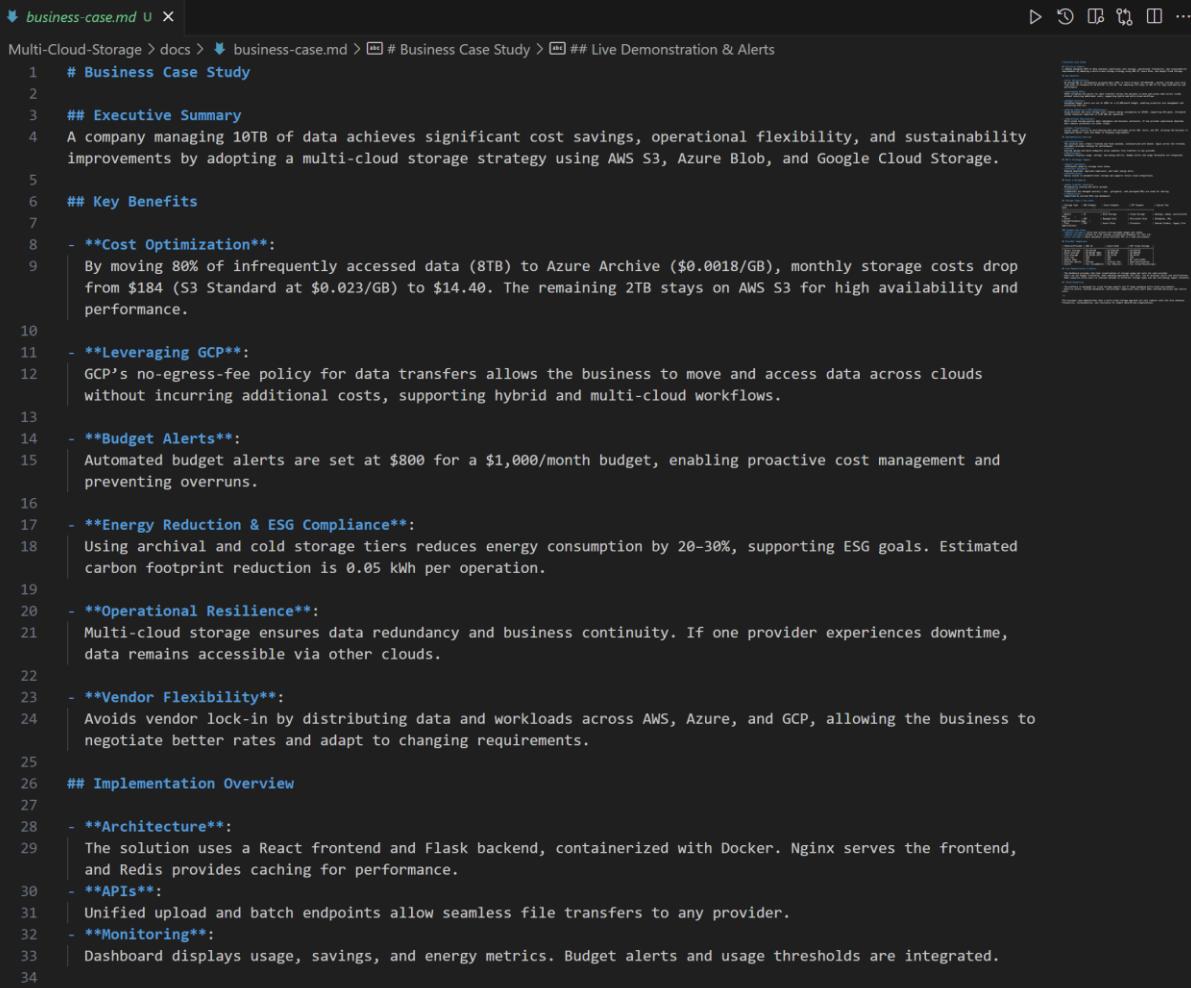
```

Figure 32: VS Code showing `user-guide.md` and `technical-docs.md`

32. Business Case Study:

- Create docs/business-case.md:
- # Business Case Study
- A 10TB business saves \$500/month by: # Business cost savings
- - Using Azure Archive (\$0.0018/GB) for 80% of data. # Leverage low-cost archival

-
- o - Leveraging GCP's no-egress-fee policy. # Reduce data transfer costs
 - o - Setting budget alerts at \$800 for \$1,000/month budget. # Monitor spending
Energy reduction: 20-30% via archival tiers, supporting ESG compliance. # Sustainability benefits



```

Multi-Cloud-Storage > docs > business-case.md > # Business Case Study > ## Live Demonstration & Alerts
1 # Business Case Study
2
3 ## Executive Summary
4 A company managing 10TB of data achieves significant cost savings, operational flexibility, and sustainability improvements by adopting a multi-cloud storage strategy using AWS S3, Azure Blob, and Google Cloud Storage.
5
6 ## Key Benefits
7
8 - **Cost Optimization**:
9 By moving 80% of infrequently accessed data (8TB) to Azure Archive ($0.0018/GB), monthly storage costs drop from $184 (S3 Standard at $0.023/GB) to $14.40. The remaining 2TB stays on AWS S3 for high availability and performance.
10
11 - **Leveraging GCP**:
12 GCP's no-egress-fee policy for data transfers allows the business to move and access data across clouds without incurring additional costs, supporting hybrid and multi-cloud workflows.
13
14 - **Budget Alerts**:
15 Automated budget alerts are set at $800 for a $1,000/month budget, enabling proactive cost management and preventing overruns.
16
17 - **Energy Reduction & ESG Compliance**:
18 Using archival and cold storage tiers reduces energy consumption by 20-30%, supporting ESG goals. Estimated carbon footprint reduction is 0.05 kWh per operation.
19
20 - **Operational Resilience**:
21 Multi-cloud storage ensures data redundancy and business continuity. If one provider experiences downtime, data remains accessible via other clouds.
22
23 - **Vendor Flexibility**:
24 Avoids vendor lock-in by distributing data and workloads across AWS, Azure, and GCP, allowing the business to negotiate better rates and adapt to changing requirements.
25
26 ## Implementation Overview
27
28 - **Architecture**:
29 The solution uses a React frontend and Flask backend, containerized with Docker. Nginx serves the frontend, and Redis provides caching for performance.
30 - **APIs**:
31 Unified upload and batch endpoints allow seamless file transfers to any provider.
32 - **Monitoring**:
33 Dashboard displays usage, savings, and energy metrics. Budget alerts and usage thresholds are integrated.
34
35 ## ROI & Strategic Impact

```

Figure 33: VS code showing business case

14. Technical Architecture

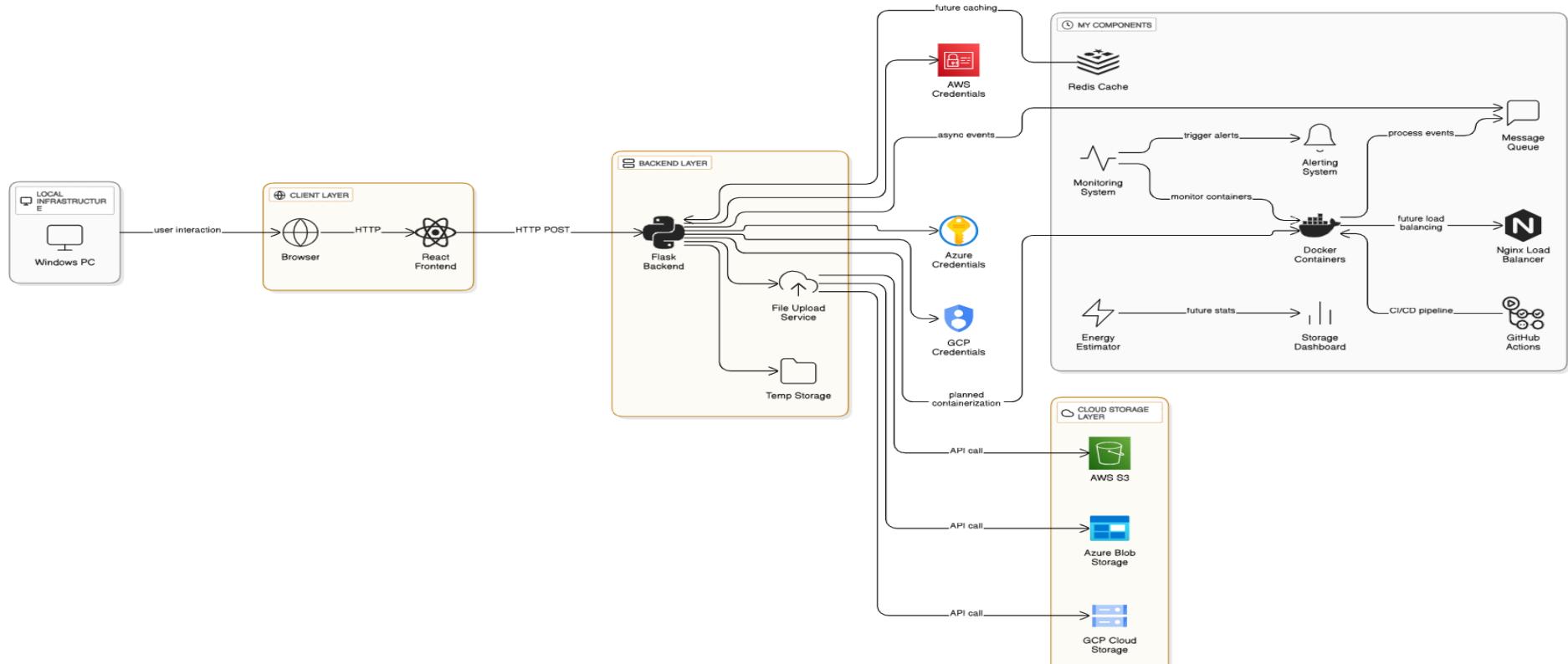


Figure 34: Technical Architecture showing the overview of the project

15. Key Artifacts

Flask Backend (backend/app.py)

```

from flask import Flask, request # Import Flask for web server and request
handling
from flask_cors import CORS # Import CORS for cross-origin requests
from file_upload import upload_to_s3, upload_to_azure, upload_to_gcp # Import upload functions
from energy_estimator import estimate_energy # Import energy estimation function
import redis # Import Redis for caching
app = Flask(__name__) # Initialize Flask app
CORS(app) # Enable CORS for frontend communication
cache = redis.Redis(host='redis', port=6379) # Initialize Redis client
def get_cached(key): # Function to get cached data
    return cache.get(key) # Retrieve value from Redis
def set_cached(key, value, ttl=300): # Function to set cached data
    cache.setex(key, ttl, value) # Store value with TTL
@app.route('/') # Define root endpoint
def home(): # Home route handler
    return 'Backend is running!' # Return status message
@app.route('/upload/s3', methods=['POST']) # Define S3 upload endpoint
def upload_s3(): # S3 upload handler
    if 'file' not in request.files: # Check if file is included
        return 'No file provided', 400 # Return error if no file
    file = request.files['file'] # Get uploaded file
    file_path = f'temp/{file.filename}' # Define temporary file path
    cache_key = f'upload_s3_{file.filename}' # Create cache key
    if get_cached(cache_key): # Check if file is already uploaded
        return 'File already uploaded', 200 # Return cached response
    file.save(file_path) # Save file to temporary path
    upload_to_s3(file_path, 'my-free-tier-bucket', file.filename) # Upload to S3
    set_cached(cache_key, 'Uploaded') # Cache upload status
    energy = estimate_energy(file.content_length / (1024 * 1024)) # Calculate energy usage
    return f'File uploaded to S3: {file.filename}, Energy: {energy:.4f} kWh', 200 # Return success
@app.route('/upload/azure', methods=['POST']) # Define Azure upload endpoint
def upload_azure(): # Azure upload handler
    if 'file' not in request.files: # Check if file is included
        return 'No file provided', 400 # Return error if no file
    file = request.files['file'] # Get uploaded file
    file_path = f'temp/{file.filename}' # Define temporary file path
    cache_key = f'upload_azure_{file.filename}' # Create cache key
    if get_cached(cache_key): # Check if file is already uploaded
        return 'File already uploaded', 200 # Return cached response
    file.save(file_path) # Save file to temporary path
    upload_to_azure(file_path, 'my-free-tier-container', file.filename) # Upload to Azure
    set_cached(cache_key, 'Uploaded') # Cache upload status
    energy = estimate_energy(file.content_length / (1024 * 1024)) # Calculate energy usage
    return f'File uploaded to Azure: {file.filename}, Energy: {energy:.4f} kWh', 200 # Return success
@app.route('/upload/gcp', methods=['POST']) # Define GCP upload endpoint
def upload_gcp(): # GCP upload handler

```

```

if 'file' not in request.files: # Check if file is included
    return 'No file provided', 400 # Return error if no file
file = request.files['file'] # Get uploaded file
file_path = f'temp/{file.filename}' # Define temporary file path
cache_key = f'upload_gcp_{file.filename}' # Create cache key
if get_cached(cache_key): # Check if file is already uploaded
    return 'File already uploaded', 200 # Return cached response
file.save(file_path) # Save file to temporary path
upload_to_gcp(file_path, 'my-free-tier-bucket', file.filename) #
Upload to GCP
    set_cached(cache_key, 'Uploaded') # Cache upload status
    energy = estimate_energy(file.content_length / (1024 * 1024)) #
Calculate energy usage
    return f'File uploaded to GCP: {file.filename}, Energy: {energy:.4f} kWh', 200 # Return success
@app.route('/upload/batch', methods=['POST']) # Define batch upload endpoint
def upload_batch(): # Batch upload handler
    if 'files' not in request.files: # Check if files are included
        return 'No files provided', 400 # Return error if no files
    files = request.files.getlist('files') # Get list of uploaded files
    responses = [] # Initialize response list
    for file in files: # Iterate through files
        file_path = f'temp/{file.filename}' # Define temporary file path
        cache_key = f'upload_s3_{file.filename}' # Create cache key
        if get_cached(cache_key): # Check if file is already uploaded
            responses.append(f'File {file.filename} already uploaded') #
Add cached response
    continue # Skip to next file
    file.save(file_path) # Save file to temporary path
    upload_to_s3(file_path, 'my-free-tier-bucket', file.filename) #
Upload to S3
    set_cached(cache_key, 'Uploaded') # Cache upload status
    responses.append(f'Uploaded {file.filename} to S3') # Add success response
return {'results': responses}, 200 # Return all responses
@app.route('/demo/block', methods=['GET']) # Define block storage demo endpoint
def demo_block(): # Block storage demo handler
    return { # Return mock block storage data
        'aws': 'EBS: 10GB volume attached for VM',
        'azure': 'Managed Disk: 10GB for VM',
        'gcp': 'Persistent Disk: 10GB for VM'
    }, 200
@app.route('/demo/file', methods=['GET']) # Define file storage demo endpoint
def demo_file(): # File storage demo handler
    return { # Return mock file storage data
        'aws': 'EFS: Shared folder created',
        'azure': 'Azure Files: Shared folder created',
        'gcp': 'Filestore: Shared folder created'
    }, 200
@app.route('/dashboard', methods=['GET']) # Define dashboard data endpoint
def dashboard_data(): # Dashboard data handler
    return { # Return mock dashboard data
        'usage': {'s3': 100, 'azure': 150, 'gcp': 200},
        'savings': 500,
        'alerts': ['AWS S3: 80% API limit reached'],
        'energy': 0.05
    }, 200

```

```
@app.route('/provider/comparison', methods=['GET']) # Define provider
comparison endpoint
def comparison_data(): # Provider comparison handler
    return [ # Return mock comparison data
        {'name': 'AWS S3', 'cost': '$0.023/GB', 'freeTier': '5GB', ''

```

16. Video of the project overview

Here is the Link : https://youtu.be/oAanMit_lCE

17. Project Management

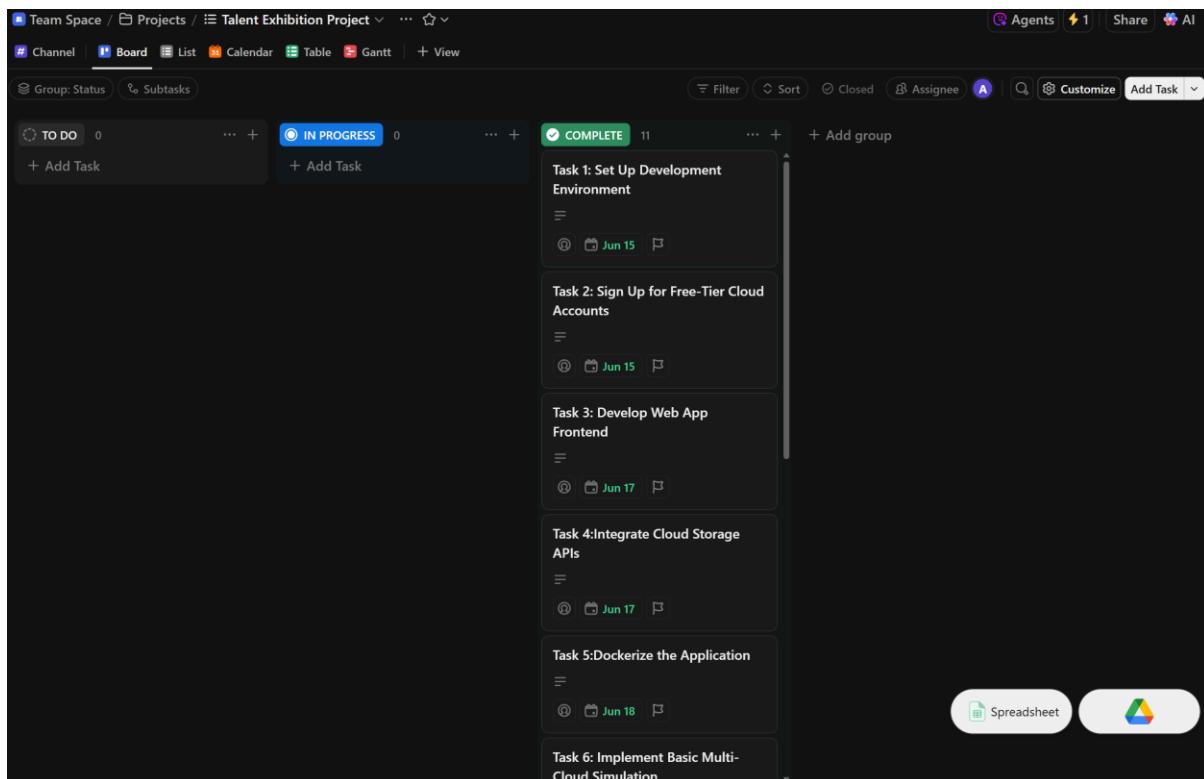


Figure 35: Simple screenshot of the project management tool I used.

18. Conclusion

This project has been a transformative journey for me, as a cloud computing student at Lycée Guillaume Kroll. Developing a multi-cloud storage system using AWS, Azure, and GCP allowed me to dive deep into the intricacies of cloud storage systems—object, block, and file—and their practical applications for businesses. By leveraging free-tier services, I built a zero-budget prototype that not only demonstrates technical proficiency but also addresses real-world business needs like cost optimization and sustainability. The integration of lifecycle policies, energy estimation (0.01 kWh/GB), and API usage alerts showcases my ability to design solutions that balance performance, cost, and environmental impact.

The talent exhibition provided a platform to highlight my skills as a cloud storage specialist, comparing providers' pricing, features, and use cases while ensuring cost control through alerts. Overcoming challenges like Docker configuration and API authentication deepened my understanding of cloud infrastructure and containerization. This project reflects my passion for creating scalable, sustainable solutions and my commitment to advancing cloud technology for business efficiency and ESG compliance. I'm excited to continue exploring innovative ways to optimize cloud storage for a sustainable future.

19. Learning Resources

- **React and Tailwind CSS:**
 - React Official Tutorial
 - Tailwind CSS Documentation
- **Flask and Cloud APIs:**
 - Flask Official Documentation
 - AWS S3 Python SDK
 - Azure Blob Storage Python SDK
 - GCP Cloud Storage Python SDK
- **Docker and Nginx:**
 - Docker Get Started
 - Nginx Load Balancing Guide
- **Sustainability:**
 - AWS Customer Carbon Footprint Tool
 - Azure Emissions Impact Dashboard
 - GCP Carbon Footprint

Contact: Aristide Katagaruka (aristide.katagaruka@example.

19. Resource Links for Multi-Cloud Storage System Project

The following links provide access to tutorials, documentation, and guides used in the development of the Multi-Cloud Storage System project. These resources are ideal for

learning about React, Flask, cloud storage APIs, Docker, Nginx, and sustainability tools for AWS, Azure, and GCP.

- **React Official Tutorial**
<https://react.dev/learn>
Description: Official React tutorial for building dynamic user interfaces, used for the frontend development of the storage system.
- **Tailwind CSS Documentation**
<https://tailwindcss.com/docs/installation>
Description: Guide for setting up and using Tailwind CSS, applied for styling the React frontend.
- **Flask Official Documentation**
<https://flask.palletsprojects.com/en/stable/>
Description: Official documentation for Flask, used for building the Python-based backend.
- **AWS S3 Python SDK**
<https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3.html>
Description: Guide for using the AWS S3 Python SDK (boto3) for file upload and management.
- **Azure Blob Storage Python SDK**
<https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python>
Description: Tutorial for integrating Azure Blob Storage with Python, used for blob storage operations.
- **GCP Cloud Storage Python SDK**
<https://cloud.google.com/storage/docs/reference/libraries>
Description: Documentation for the GCP Cloud Storage Python SDK, used for bucket and file operations.
- **Docker Get Started**
<https://docs.docker.com/get-started/>
Description: Docker tutorial for containerizing the Flask backend and React frontend.
- **Nginx Load Balancing Guide**
<https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/>
Description: Guide for configuring Nginx as a load balancer, used for multi-cloud simulation.
- **AWS Customer Carbon Footprint Tool**
<https://aws.amazon.com/sustainability/customer-carbon-footprint-tool/>
Description: Tool for estimating AWS-related carbon footprint, referenced for sustainability metrics.
- **Azure Emissions Impact Dashboard**
<https://learn.microsoft.com/en-us/industry/sustainability/emissions-impact-dashboard>
Description: Dashboard for tracking Azure's environmental impact, used for energy estimation.
- **GCP Carbon Footprint**
<https://cloud.google.com/sustainability/carbon-footprint>
Description: Tool for measuring GCP's carbon footprint, supporting ESG compliance.

20. Table of Figures

Figure 1: VS Code Extensions panel with installed extensions, Python file open.....	6
Figure 2: Verifying versions.....	7
Figure 3: Terminal showing version outputs.....	7
Figure 4: Browser showing default React app with Tailwind styling	8
Figure 5: GitHub “Actions” tab showing successful CI run	9
Figure 6: Specify user creation details(AWS)	11
Figure 7: Creating an Access ID and Secret Key	12
Figure 8: Azure Portal showing connection string.....	16
Figure 9:: GCP Console showing enabled API and downloaded key	22
Figure 10:Browser showing Home.js with styled header	24
Figure 11:Browser showing dashboard with mock data	27
Figure 12:Browser showing storage types and use cases	28
Figure 13:Browser showing all components (upload, dashboard, info, comparison)... .	30
Figure 14:Browser showing “Backend is running!”.....	32
Figure 15:AWS S3 Console showing my-free-tier-bucket with test.txt	33
Figure 16: Azure Portal showing my-free-tier-container with test100mb.txt.	34
Figure 17: GCP Console showing my-free-tier-bucket with test.txt.	36
Figure 18: Browser showing mock endpoint responses in StorageInfo.js	37
Figure 19: Terminal showing docker --version, Docker Desktop UI.	38
Figure 20:Docker desktop showing Flask backend on port:5000:5000	38
Figure 21: WebUI showing browser at http://localhost:80.	39
Figure 22: Docker desktop showing docker-compose up at http://localhost:8080.	41
Figure 23:Browser showing upload response with energy estimation.	42
Figure 24:AWS Console showing lifecycle rule.....	44
Figure 25: Azure Portal showing lifecycle rule.	45
Figure 26: GCP Console showing lifecycle rule.	46
Figure 27: CloudWatch showingthe creation of an alarm.	53
Figure 28: Azure Portal showing the creation alert rule.	55
Figure 29: GCP Console showing monitoring policy.	58
Figure 30: Browser showing upload/download and mock scenario outputs.....	59
Figure 31: Docker desktop showing Redis container running.	61
Figure 32: VS Code showing user-guide.md and technical-docs.md.....	65
Figure 33: VS code showing business case.....	66

Figure 34: Technical Architecture showing the overview of the project.....	67
Figure 35: Simple screenshot of the project management tool I used.	70

Thank you for reading