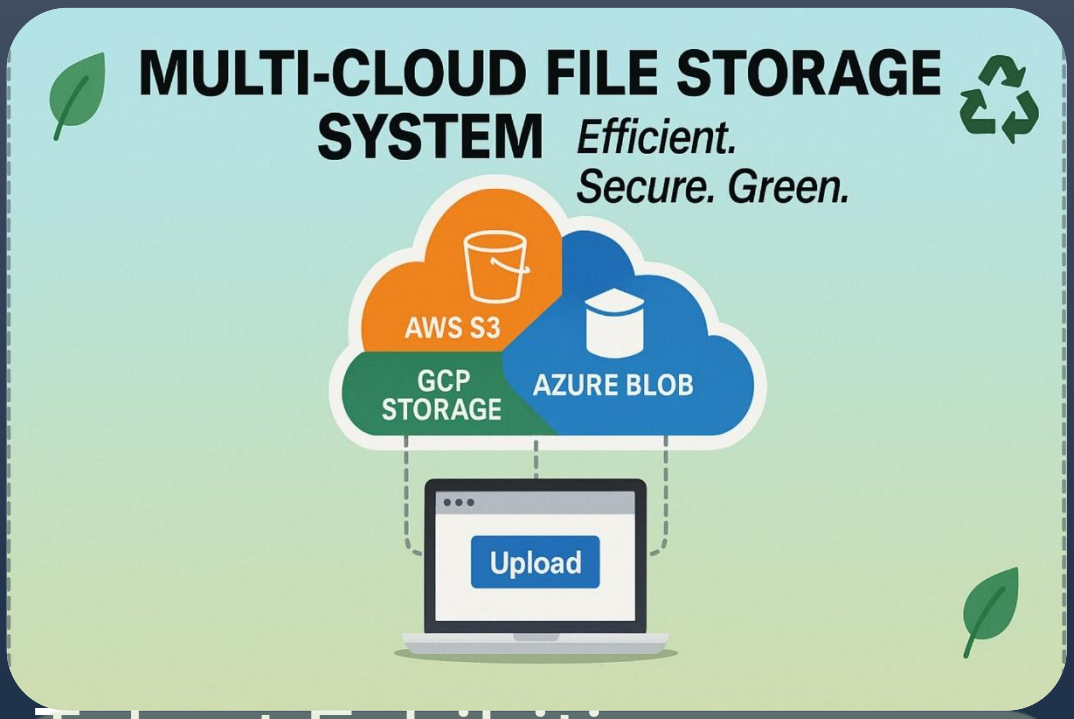
2025



Talent Exhibition Project: Multi-Cloud File Storage

Lycée Guillaume Kroll

ARISTIDE KATAGARUKA

BTS CLOUD COMPUTING

LYCÉE GUILLAME KROLL

SUBMISSION DATE: JULY 2ND 2025

Contents

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,
alerts
│   │   │   ├── StorageInfo.js            # Info and use cases for storage
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+.

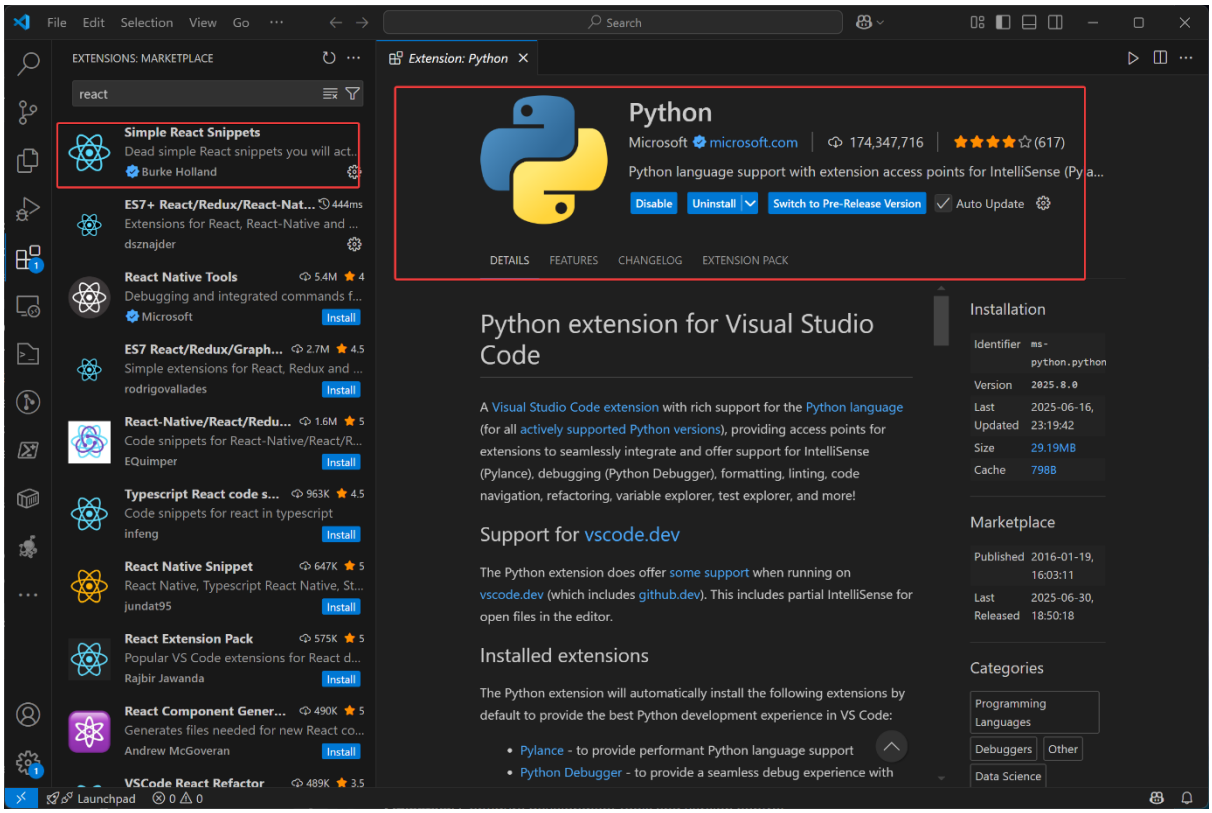


Figure 1:VS Code Extensions panel with installed extensions, Python file open

- **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
```

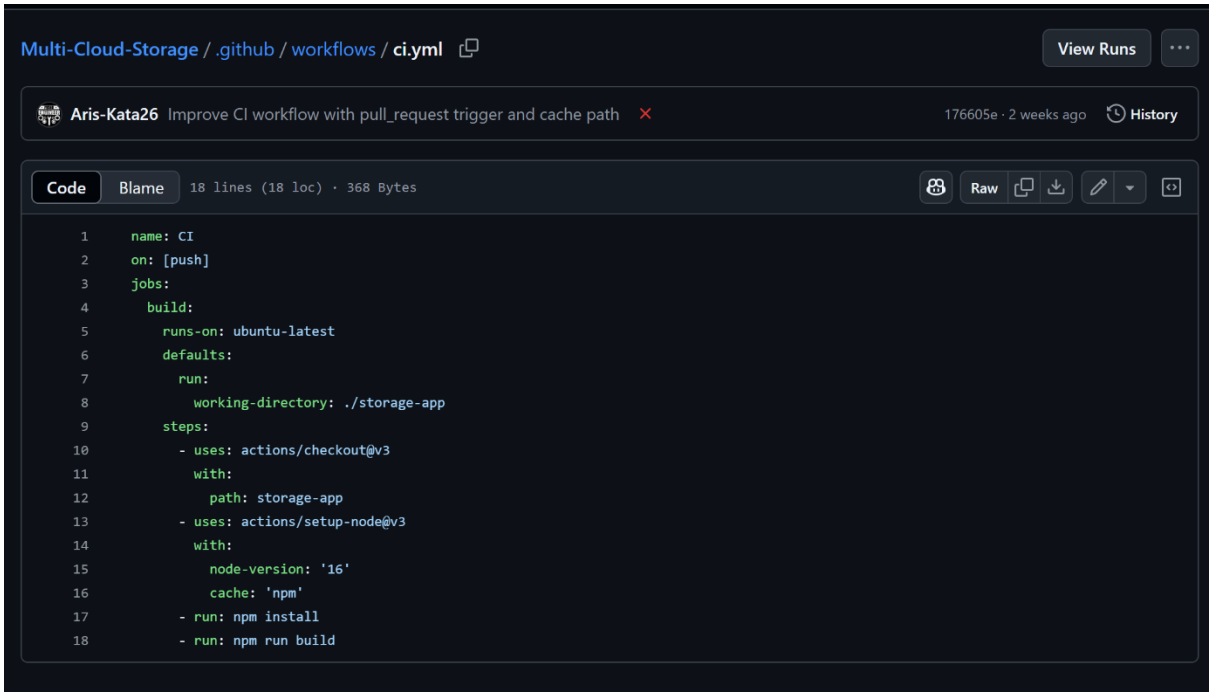


Figure 5: GitHub "Actions" tab showing successful CI run

- **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.

---

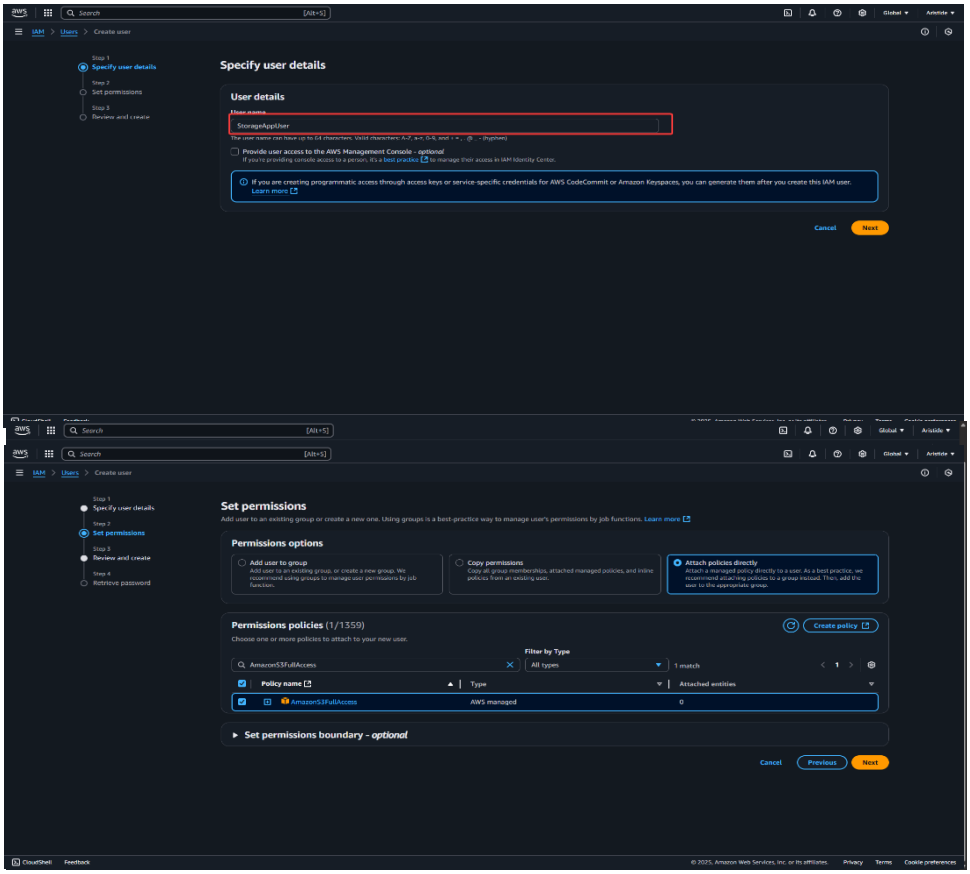
5. Cloud Account Setup (2 hours, June 3, 2025)

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.



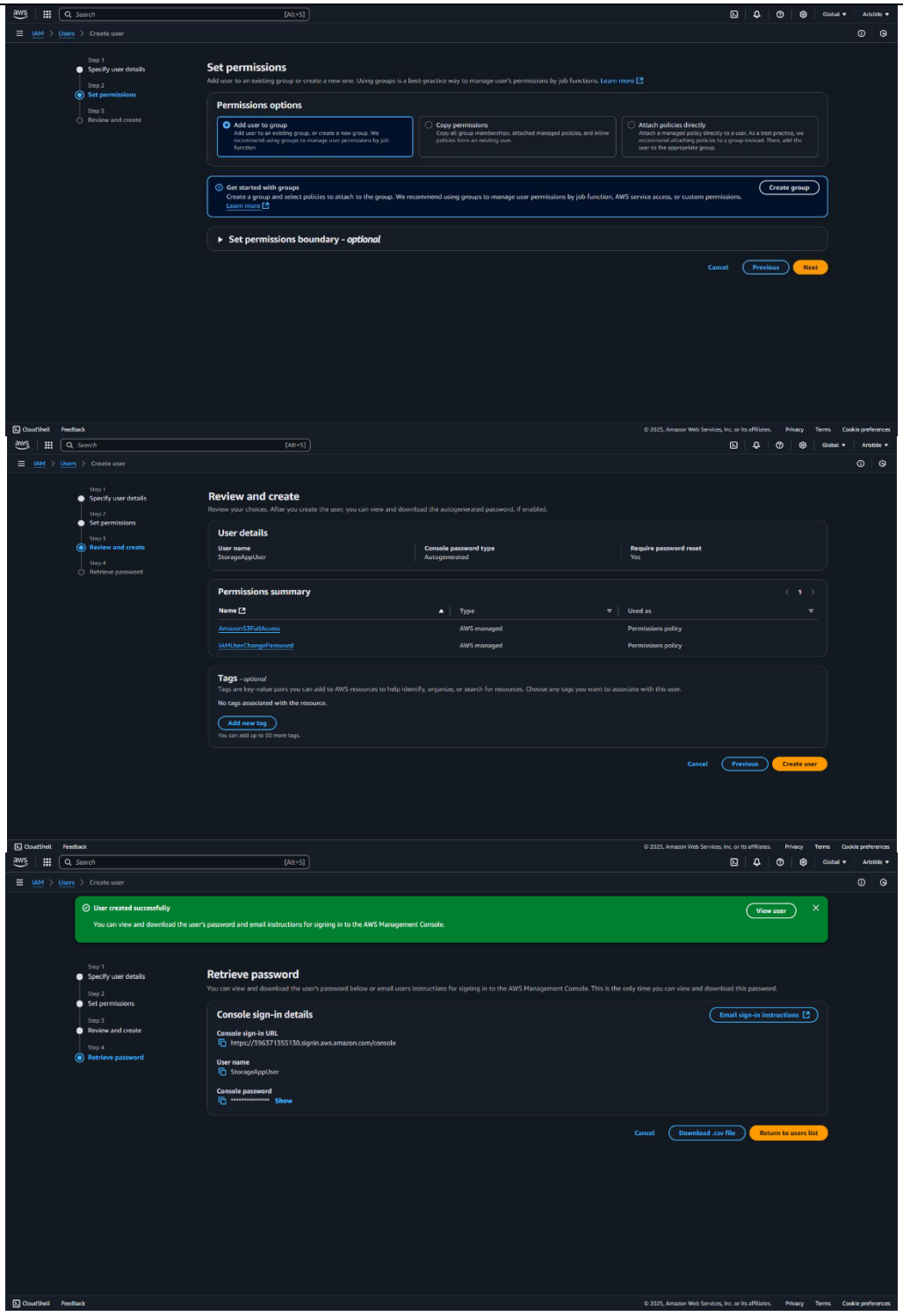


Figure 6: Specify user creation details(AWS)

- Attach AmazonS3FullAccess policy.

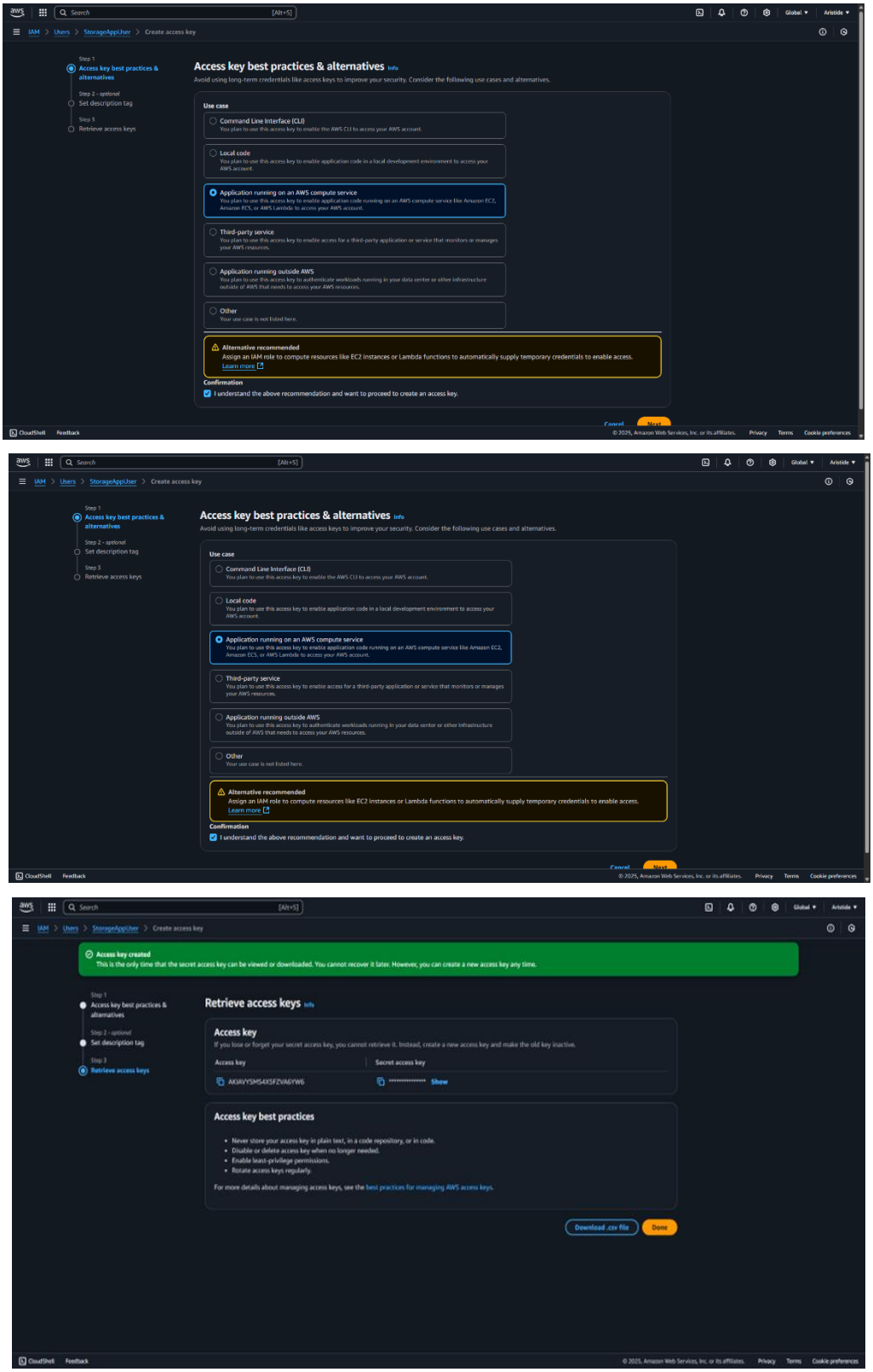


Figure 7: Creating an Access ID and Secret Key

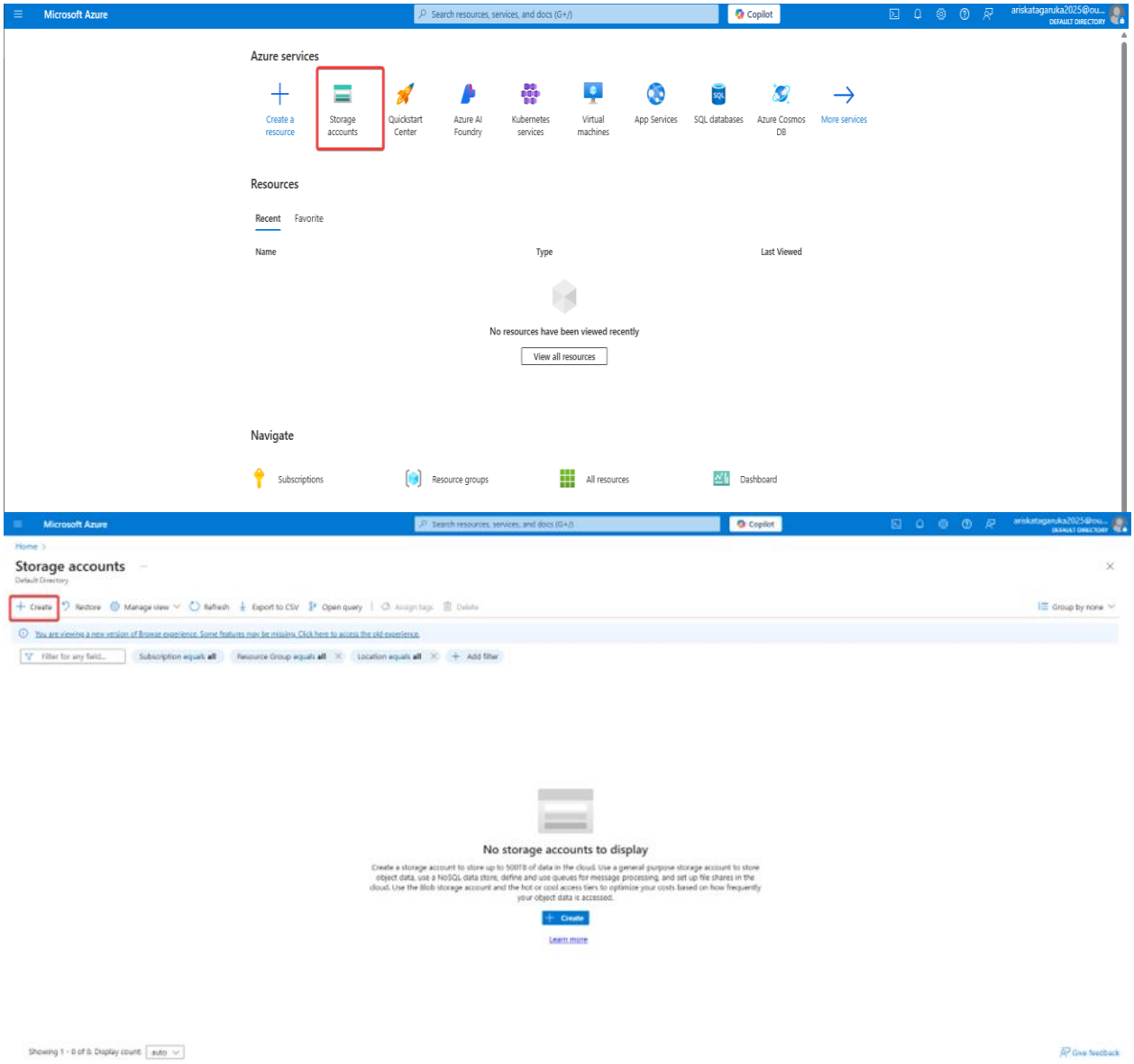
- Save Access Key ID and Secret Access Key in backend/.env:

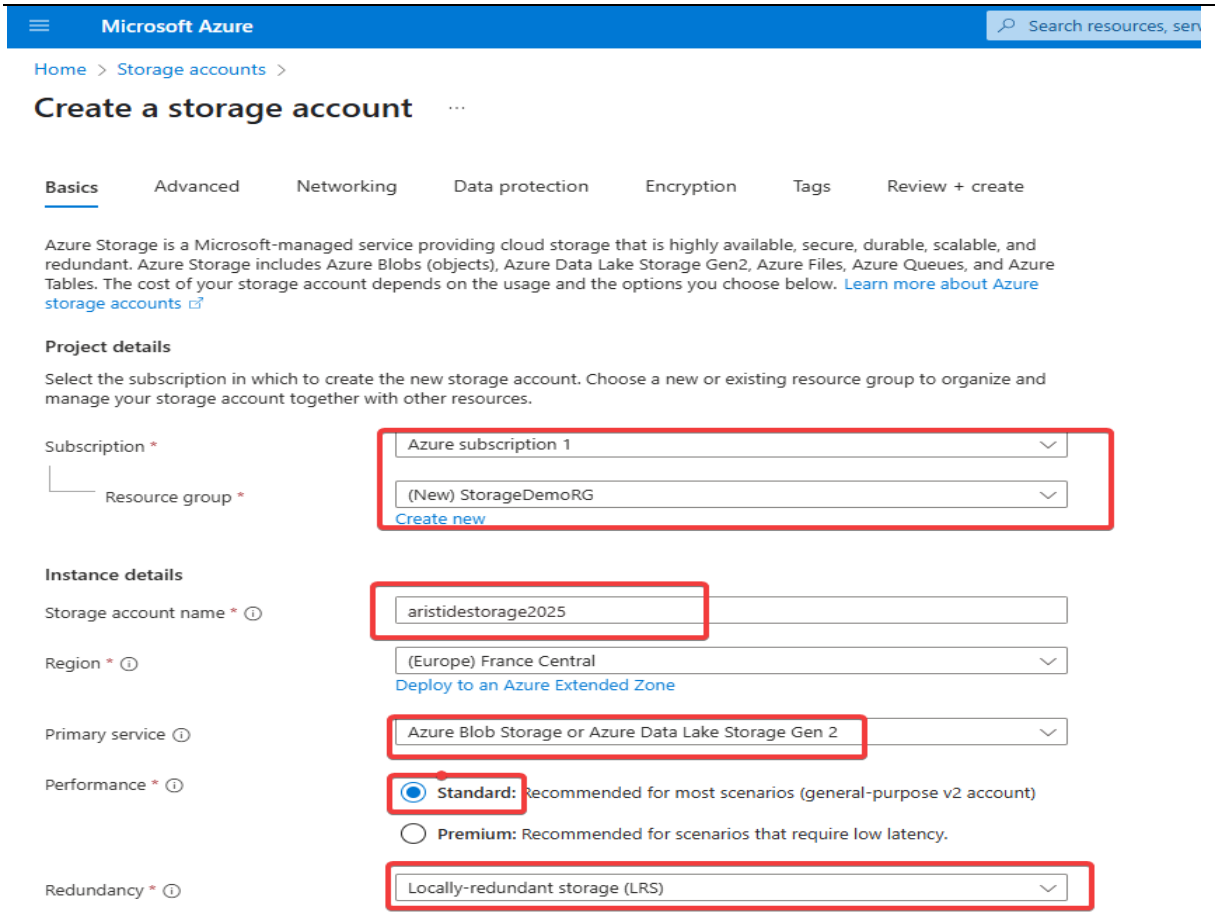
- `AWS_ACCESS_KEY_ID=your-access-key  # AWS access key for S3`
  `AWS_SECRET_ACCESS_KEY=your-secret-key  # AWS secret key for S3`

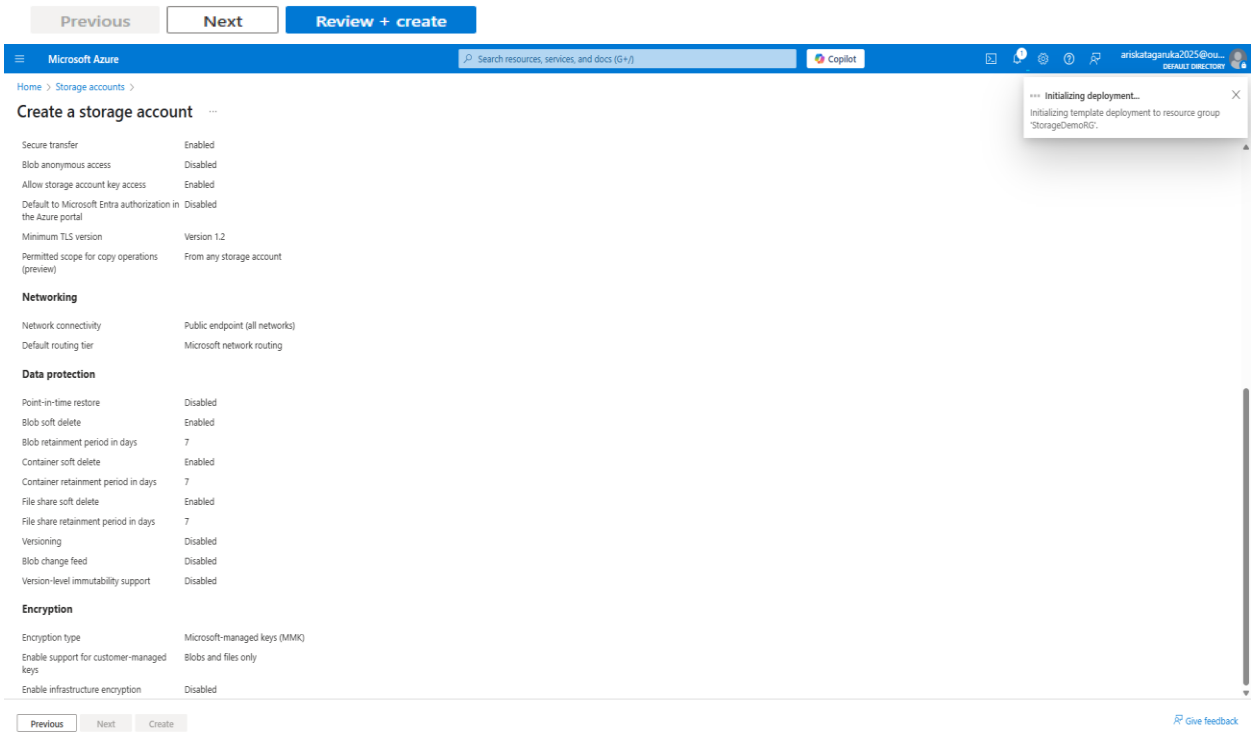
6. *Azure Free Tier:*

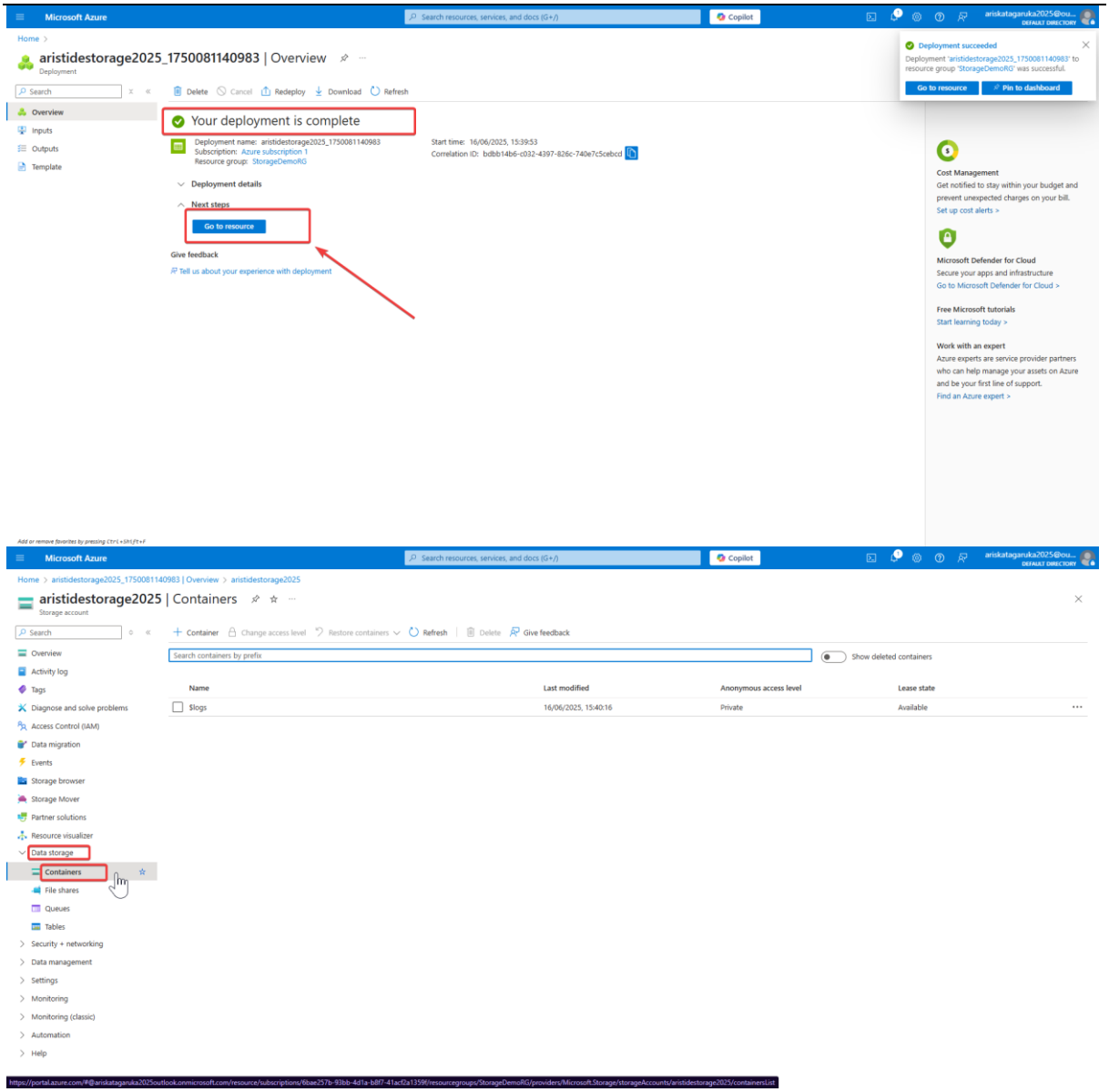
- Visit azure.microsoft.com/free, sign up with Microsoft account.
- Provide payment info for free services (5GB Blob Storage).
- In Azure Portal:
  - Create Storage Account: mystorageaccount, select “Locally-redundant storage” (LRS).
  - Go to Access keys, copy connection string to backend/.env:

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




Microsoft Azure
Search resources, ser
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)


Previous
Next
Review + create
Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
Home > Storage accounts >
Create a storage account
Initializing deployment...
Initializing template deployment to resource group 'StorageDemoRG'.
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 (preview) From any storage account
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 retainment period in days 7
Container soft delete Enabled
Container retainment period in days 7
File share soft delete Enabled
File share retainment 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
Previous
Next
Create
Give feedback


Microsoft Azure
aristidestorage2025_1750081140983 | Overview
Deployment
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: bdbb14b6-c032-4397-826c-740e7c5cebcd
Deployment details
Next steps
Go to resource
Give feedback
Tell us about your experience with deployment
Deployment succeeded
Go to resource
Pin to dashboard
Cost Management
Microsoft Defender for Cloud
Free Microsoft tutorials
Work with an expert
aristidestorage2025 | Containers
Storage account
Container
Change access level
Restore containers
Refresh
Delete
Give feedback
Search containers by prefix
Show deleted containers
Name
Last modified
Anonymous access level
Lease state
$logs
16/06/2025, 15:40:16
Private
Available
Overview
Activity log
Tags
Diagnose and solve problems
Access Control (IAM)
Data migration
Events
Storage browser
Storage Mover
Partner solutions
Resource visualizer
Data storage
Containers
File shares
Queues
Tables
Security + networking
Data management
Settings
Monitoring
Monitoring (classic)
Automation
Help

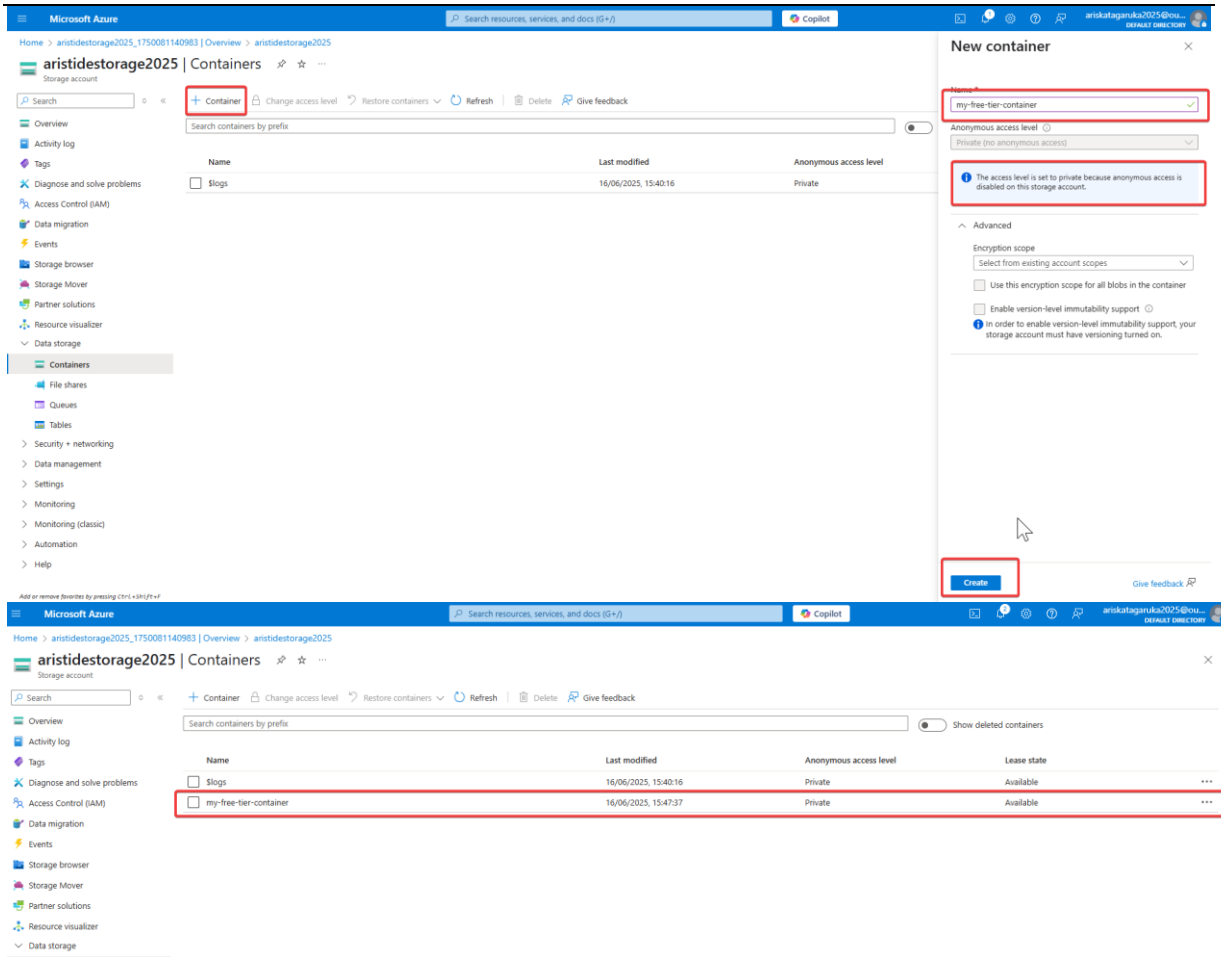
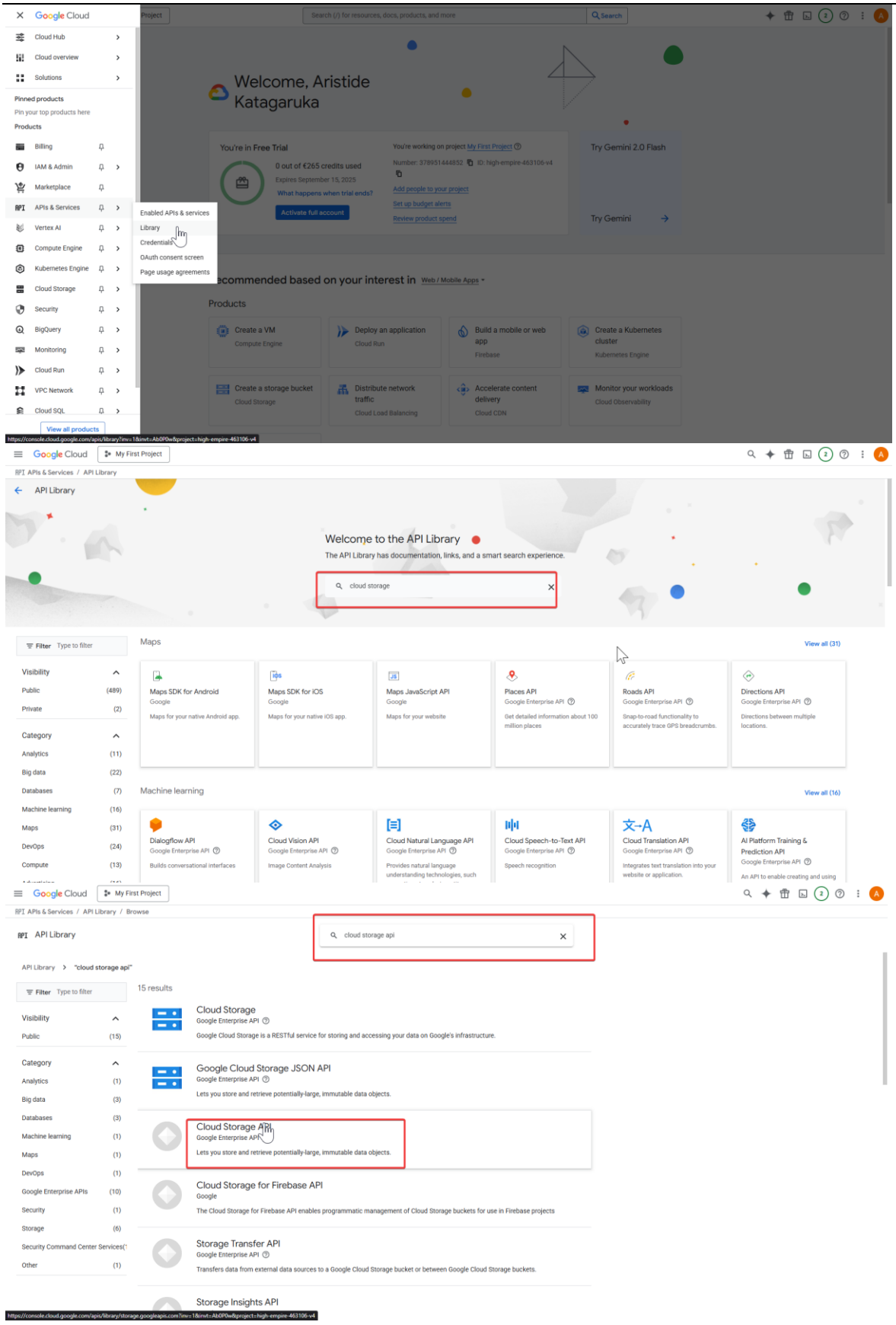


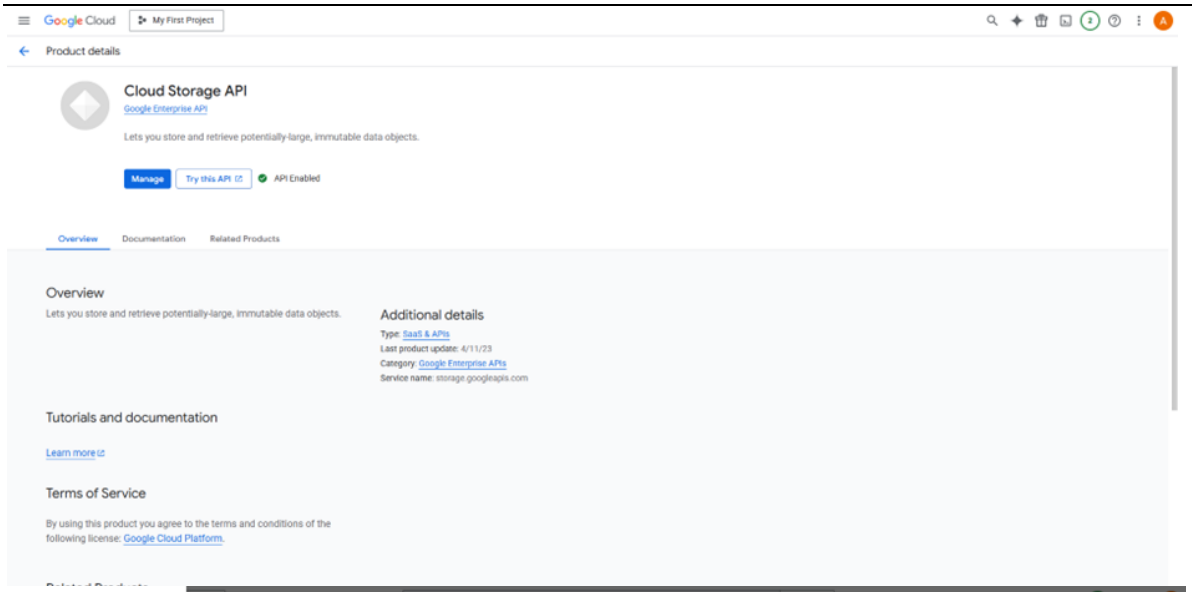
Figure 8:Azure Portal showing connection string

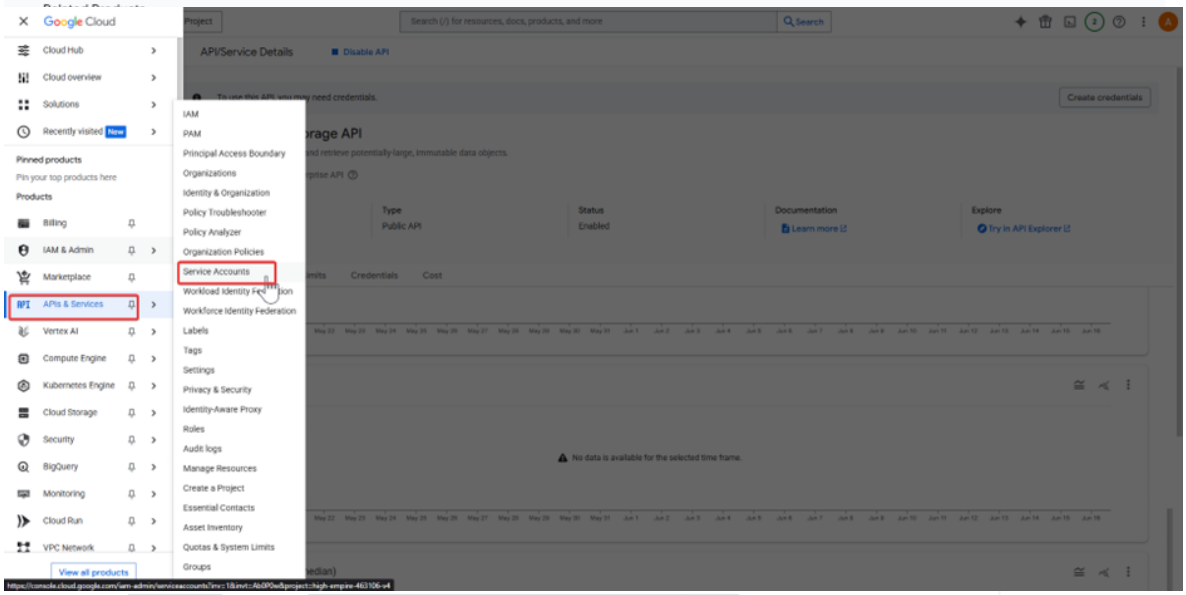
7. GCP Free Tier (0.6 hour):

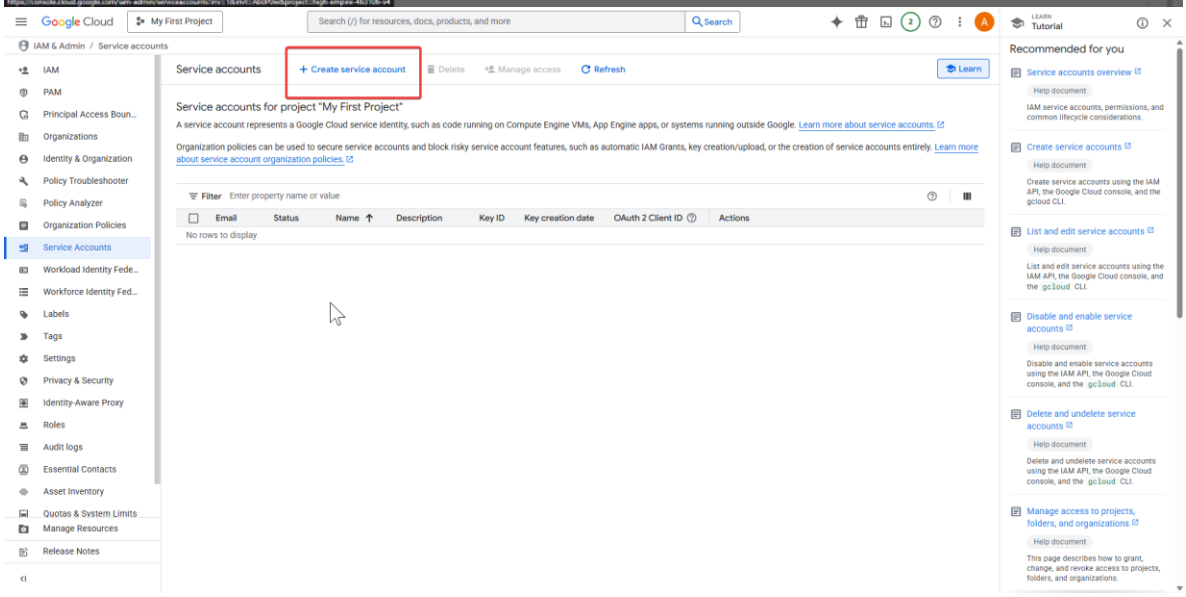
- Sign up at cloud.google.com/free with $300 credit (90 days).
- In Google Cloud Console:
  - Create project: multi-cloud-storage.
  - Enable Cloud Storage API (APIs & Services > Library).
  - Create service account: storage-service-account, assign “Storage Admin” role, download JSON key as backend/key.json.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

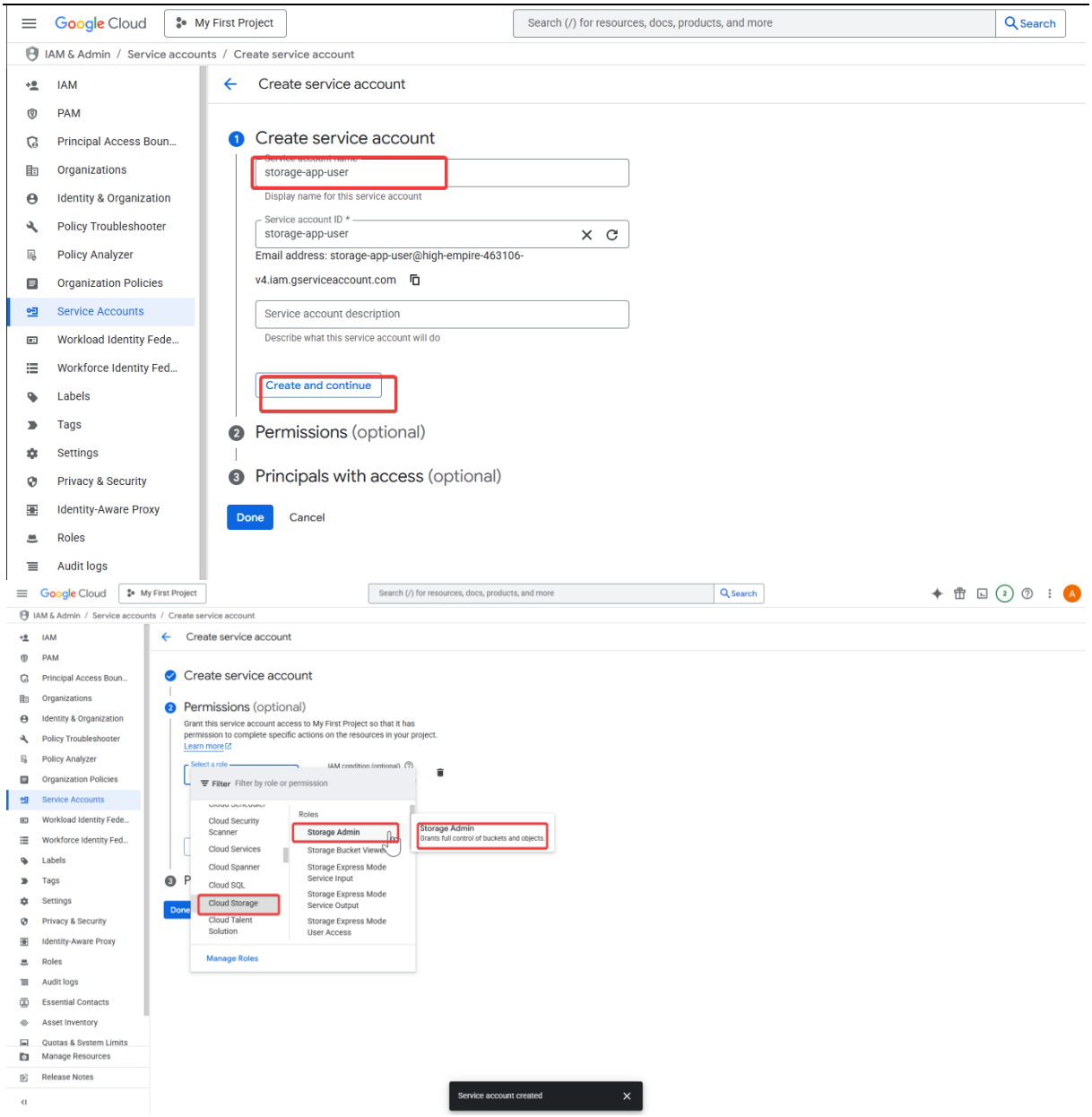
```
GOOGLE_APPLICATION_CREDENTIALS=/app/key.json  # Path to GCP service account key
```

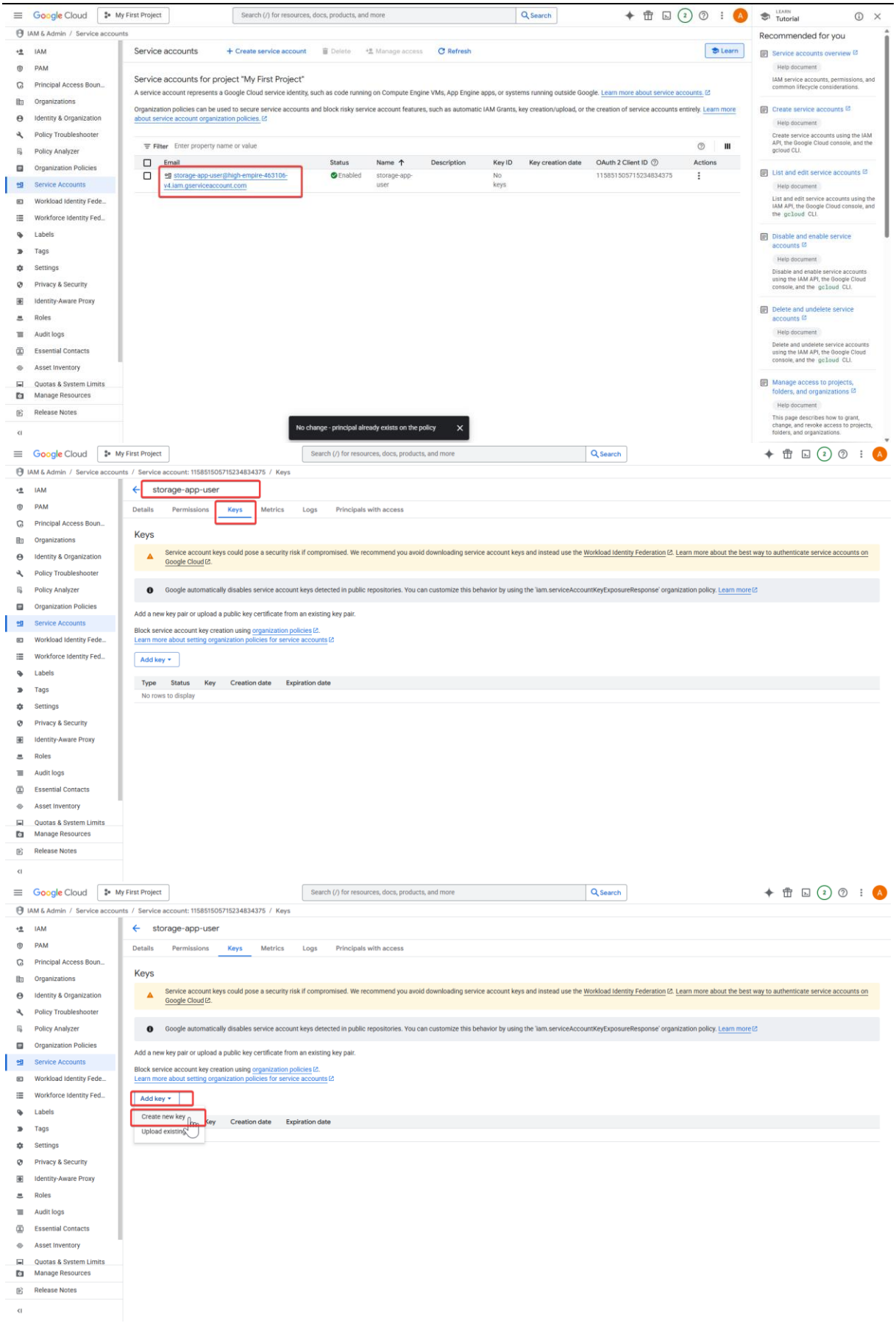

Google Cloud
Welcome, Aristide Katagaruka
Enabled APIs & services
Library
Credentials
OAuth consent screen
Page usage agreements
API Library
Welcome to the API Library
The API Library has documentation, links, and a smart search experience.
cloud storage
cloud storage api
15 results
Cloud Storage
Google Cloud Storage JSON API
Cloud Storage API
Cloud Storage for Firebase API
Storage Transfer API
Storage Insights API

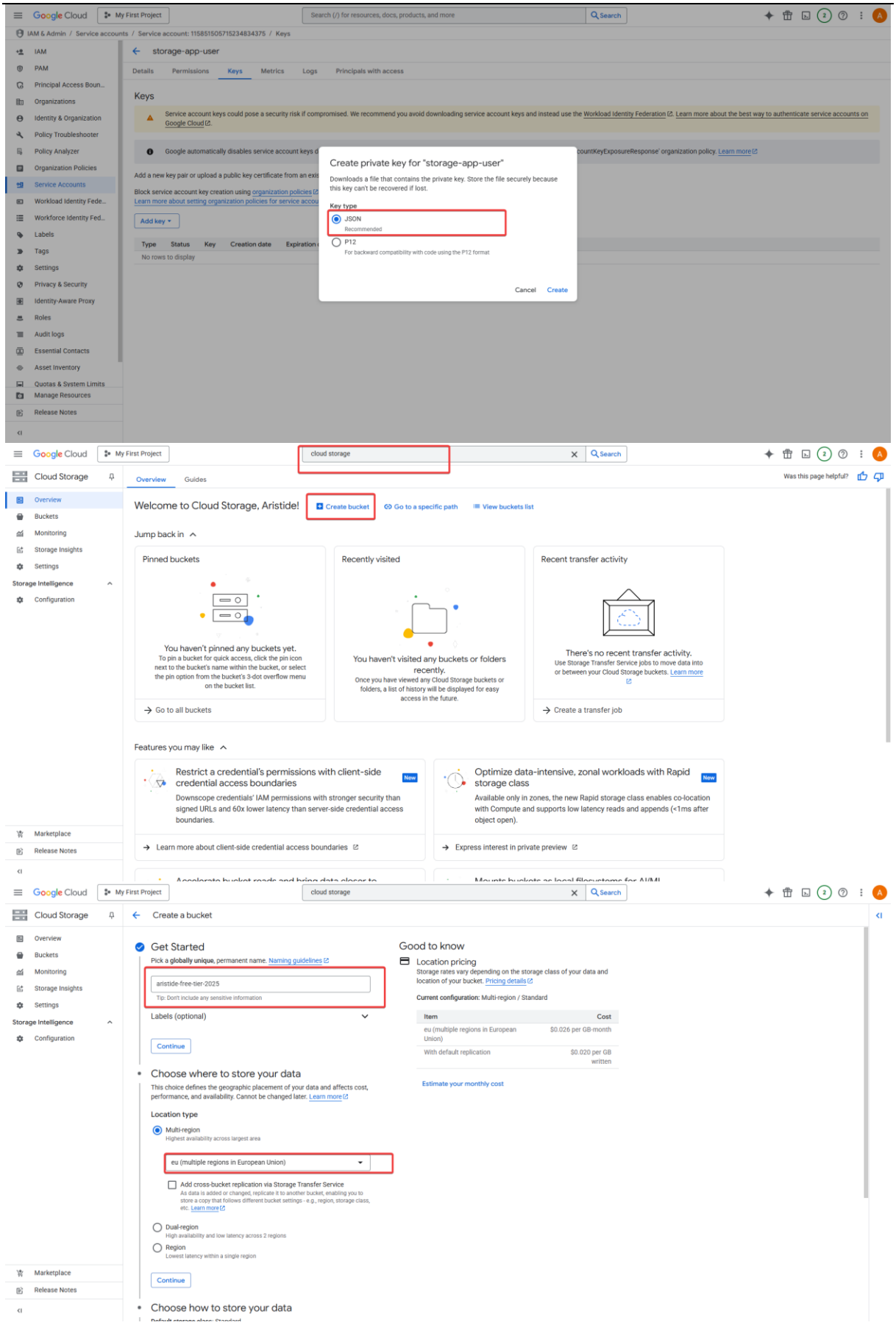

Google Cloud
My First Project
Product details
Cloud Storage API
Google Enterprise API
Lets you store and retrieve potentially-large, immutable data objects.
Manage
Try this API
API Enabled
Overview
Documentation
Related Products
Overview
Lets you store and retrieve potentially-large, immutable data objects.
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.


Google Cloud
API/Service Details
Disable API
Create credentials
Cloud Hub
Cloud overview
Solutions
Recently visited
Pinned products
Products
Billing
IAM & Admin
Marketplace
APIs & Services
Vertex AI
Compute Engine
Kubernetes Engine
Cloud Storage
Security
BigQuery
Monitoring
Cloud Run
VPC Network
View all products
IAM
PAM
Principal Access Boundary
Organizations
Identity & Organization
Policy Troubleshooter
Policy Analyzer
Organization Policies
Service Accounts
Workload Identity Federation
Workforce Identity Federation
Labels
Tags
Settings
Privacy & Security
Identity-Aware Proxy
Roles
Audit logs
Manage Resources
Create a Project
Essential Contacts
Asset Inventory
Quotas & System Limits
Groups
Type
Public API
Status
Enabled
Documentation
Learn more
Explore
Try in API Explorer
Credentials
Cost
No data is available for the selected time frame.


Google Cloud
My First Project
Search (/) for resources, docs, products, and more
Search
IAM & Admin / Service accounts
Service accounts
+ Create service account
Delete
Manage access
Refresh
Learn
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
Tutorial
Recommended for you
Service accounts overview
Create service accounts
List and edit service accounts
Disable and enable service accounts
Delete and undelete service accounts
Manage access to projects, folders, and organizations


Google Cloud
My First Project
Search (/) for resources, docs, products, and more
Search
IAM & Admin / Service accounts / Create service account
IAM
PAM
Principal Access Boun...
Organizations
Identity & Organization
Policy Troubleshooter
Policy Analyzer
Organization Policies
Service Accounts
Workload Identity Fede...
Workforce Identity Fed...
Labels
Tags
Settings
Privacy & Security
Identity-Aware Proxy
Roles
Audit logs
Create service account
1 Create service account
storage-app-user
Display name for this service account
Service account ID *
storage-app-user
Email address: storage-app-user@high-empire-463106-
v4.iam.gserviceaccount.com
Service account description
Describe what this service account will do
Create and continue
2 Permissions (optional)
3 Principals with access (optional)
Done
Cancel
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.
Learn more
Filter Filter by role or permission
Roles
Cloud Security Scanner
Cloud Services
Cloud Spanner
Cloud SQL
Cloud Storage
Cloud Talent Solution
Storage Admin
Storage Bucket Viewer
Storage Express Mode Service Input
Storage Express Mode Service Output
Storage Express Mode User Access
Storage Admin
Grants full control of buckets and objects.
Manage Roles
Service account created


Service accounts
Create service account
Service accounts for project "My First Project"
storage-app-user@high-empire-463106-v4.iam.gserviceaccount.com
No change - principal already exists on the policy
storage-app-user
Details
Permissions
Keys
Metrics
Logs
Principals with access
Keys
Add key
Create new key
Upload existing
Type
Status
Key
Creation date
Expiration date
No rows to display


Google Cloud
My First Project
Search (/) for resources, docs, products, and more
Search
IAM & Admin / Service accounts / Service account: 115851505715234834375 / Keys
storage-app-user
Details
Permissions
Keys
Metrics
Logs
Principals with access
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.
Add key
Type
Status
Key
Creation date
Expiration
No rows to display
Create private key for "storage-app-user"
Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.
Key type
JSON
Recommended
P12
For backward compatibility with code using the P12 format
Cancel
Create
IAM
PAM
Principal Access Boun...
Organizations
Identity & Organization
Policy Troubleshooter
Policy Analyzer
Organization Policies
Service Accounts
Workload Identity Fede...
Workforce Identity Fed...
Labels
Tags
Settings
Privacy & Security
Identity-Aware Proxy
Roles
Audit logs
Essential Contacts
Asset Inventory
Quotas & System Limits
Manage Resources
Release Notes
Google Cloud
My First Project
cloud storage
Search
Cloud Storage
Overview
Guides
Was this page helpful?
Overview
Buckets
Monitoring
Storage Insights
Settings
Storage Intelligence
Configuration
Welcome to Cloud Storage, Aristide!
Create bucket
Go to a specific path
View buckets list
Jump back in
Pinned buckets
You haven't pinned any buckets yet.
To pin a bucket for quick access, click the pin icon next to the bucket's name within the bucket, or select the pin option from the bucket's 3-dot overflow menu on the bucket list.
Go to all buckets
Recently visited
You haven't visited any buckets or folders recently.
Once you have viewed any Cloud Storage buckets or folders, a list of history will be displayed for easy access in the future.
Recent transfer activity
There's no recent transfer activity.
Use Storage Transfer Service jobs to move data into or between your Cloud Storage buckets. Learn more
Create a transfer job
Features you may like
Restrict a credential's permissions with client-side credential access boundaries
New
Downscope credentials' IAM permissions with stronger security than signed URLs and 60x lower latency than server-side credential access boundaries.
Learn more about client-side credential access boundaries
Optimize data-intensive, zonal workloads with Rapid storage class
New
Available only in zones, the new Rapid storage class enables co-location with Compute and supports low latency reads and appends (<1ms after object open).
Express interest in private preview
Marketplace
Release Notes
Google Cloud
My First Project
cloud storage
Search
Cloud Storage
Create a bucket
Overview
Buckets
Monitoring
Storage Insights
Settings
Storage Intelligence
Configuration
Get Started
Pick a globally unique, permanent name. Naming guidelines
aristide-free-tier-2025
Tip: Don't include any sensitive information
Labels (optional)
Continue
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
Location type
Multi-region
Highest availability across largest area
eu (multiple regions in 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
Dual-region
High availability and low latency across 2 regions
Region
Lowest latency within a single region
Continue
Choose how to store your data
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
Marketplace
Release Notes

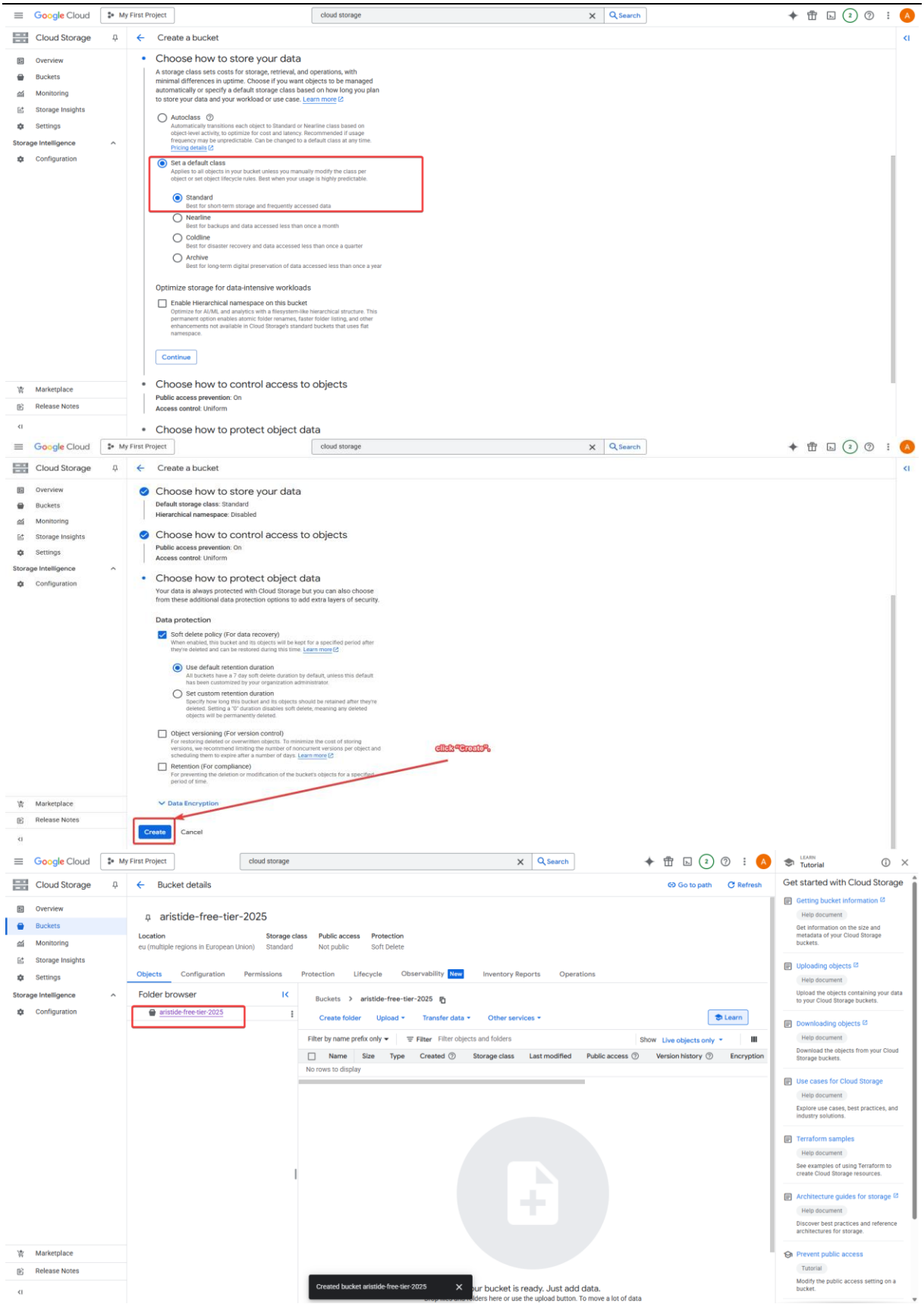


Figure 9:: GCP Console showing enabled API and downloaded key

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
           // App container
           <Home />  // Render Home component

       );
     }
     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:

- Create storage-app/src/components/FileUpload.js:
- `import React, { useState } from 'react';  // Import React and useState hook`
- `import { uploadFile } from '../services/api';  // Import API utility for file upload`
- `function FileUpload() {  // Define FileUpload component`
- `  const [file, setFile] = useState(null);  // State for selected file`
- `  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`
- `    if (e.target.files.length > 0) {  // Check if file is selected`
- `      setFile(e.target.files[0]);  // Update file state`
- `    }`
- `  };`
- `  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`

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

- Create storage-app/src/services/api.js:

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

10. Implement Storage Dashboard:

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

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

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

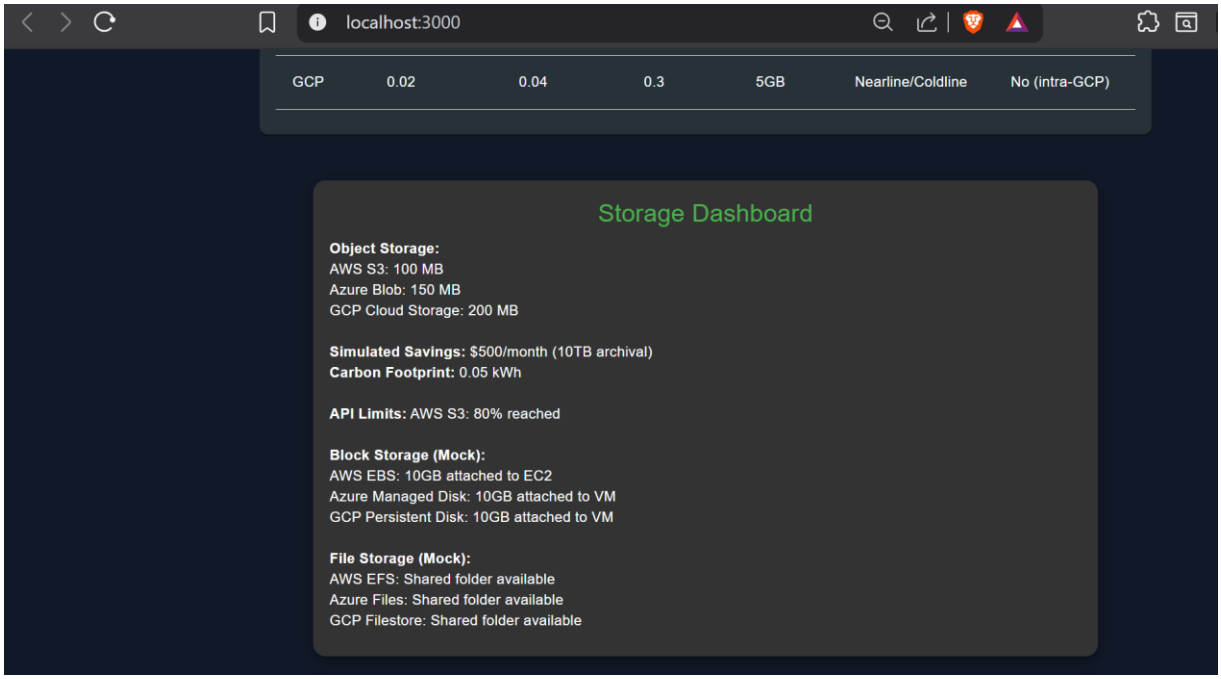


Figure 11:Browser showing dashboard with mock data

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`

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

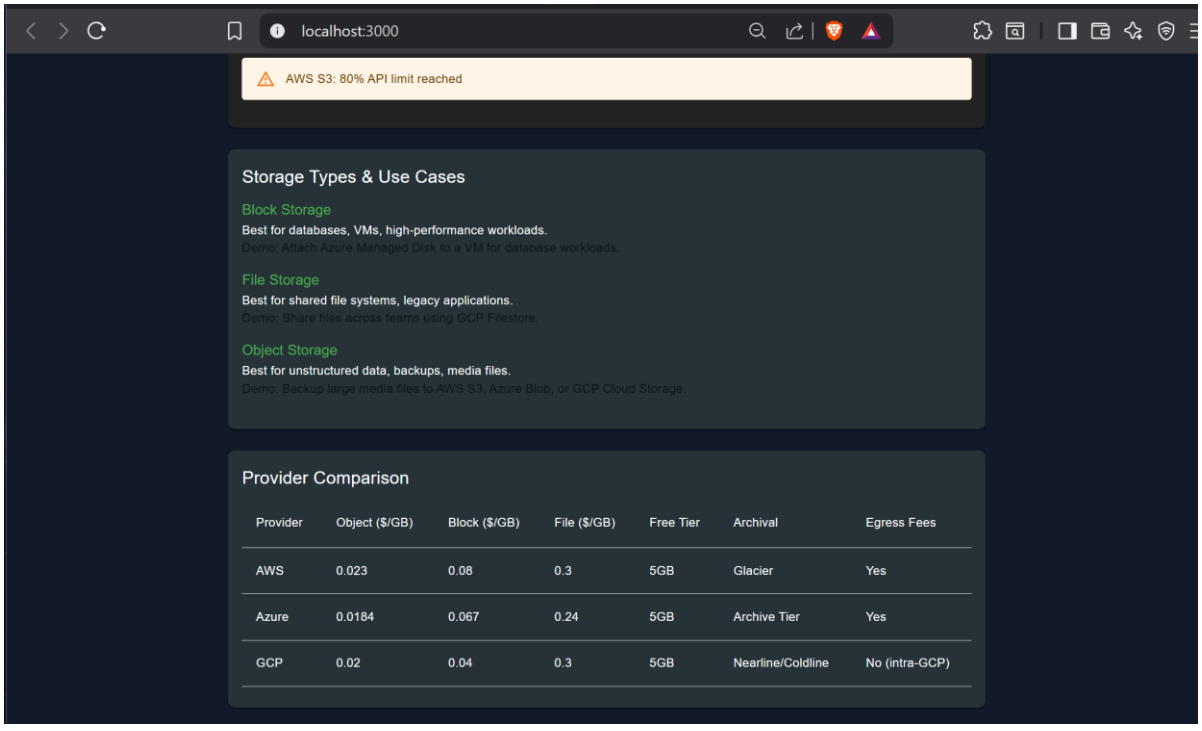


Figure 12:Browser showing storage types and use cases

12. Implement Provider Comparison:

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

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

- Update Home.js to include all components:

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

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

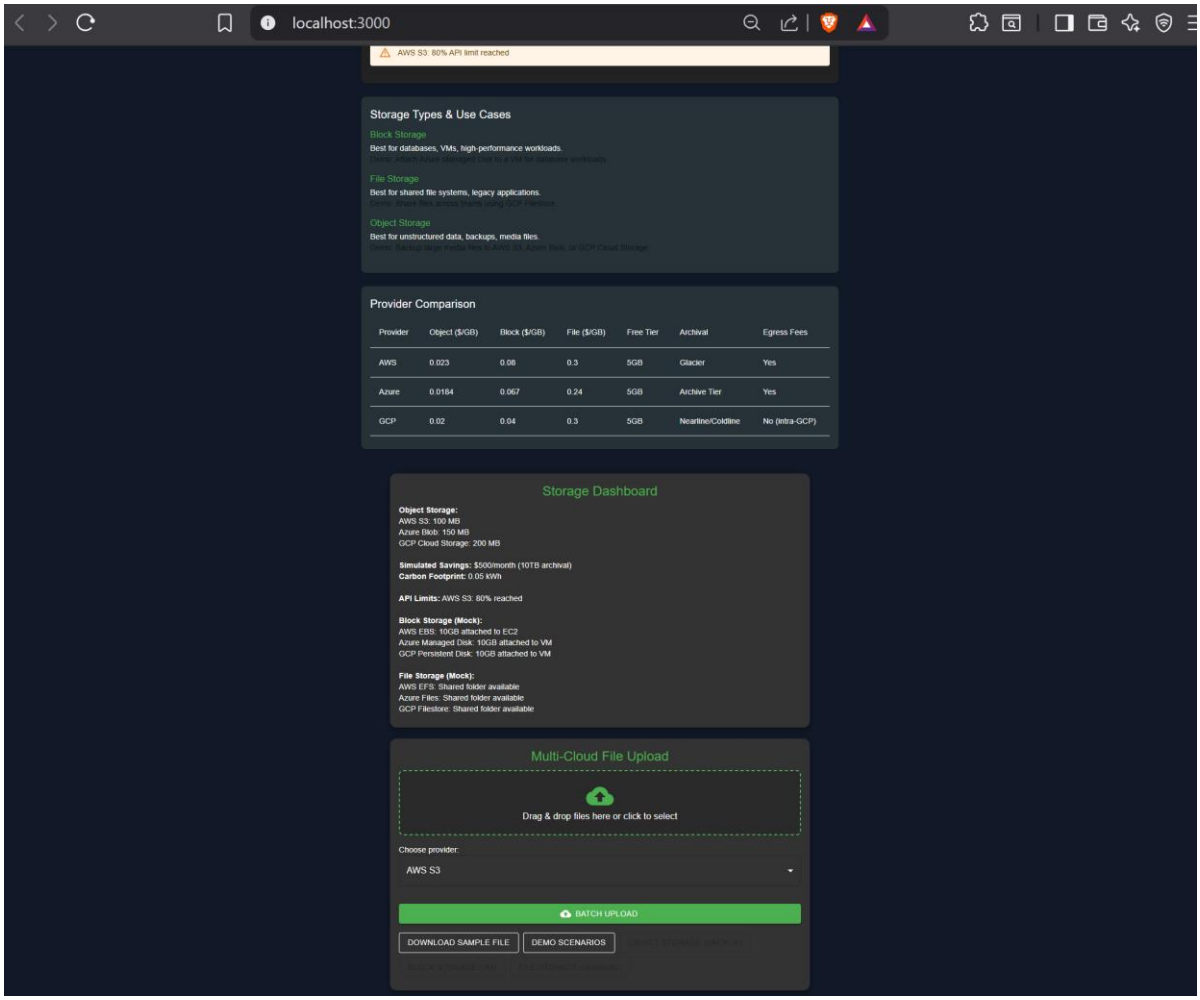


Figure 13:Browser showing all components (upload, dashboard, info, comparison).

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.

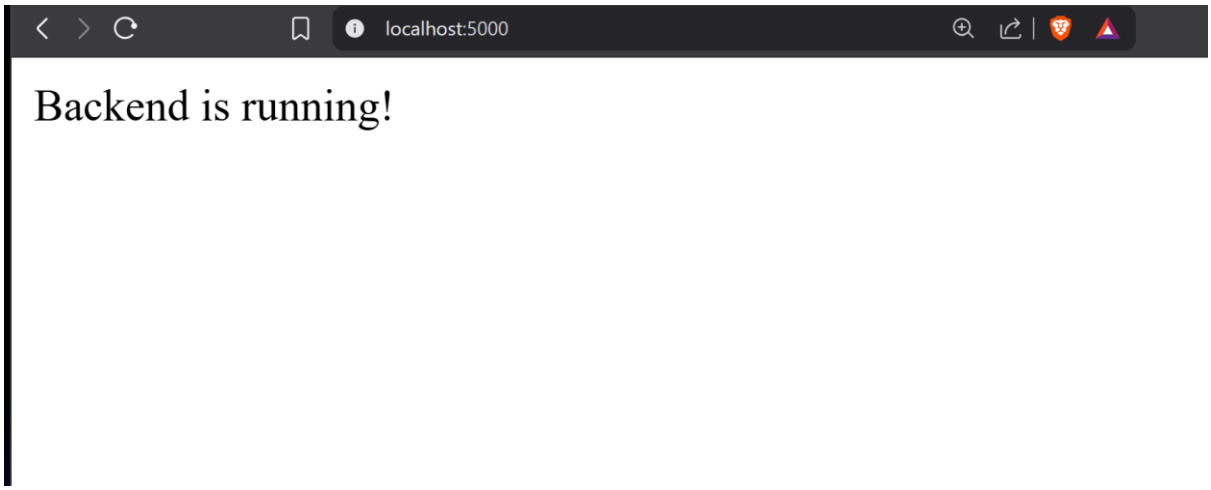


Figure 14:Browser showing "Backend is running!".

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`

```
    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
        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 message
```

- In AWS Console: Create bucket my-free-tier-bucket (us-east-1, disable public access block).
- Test:

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

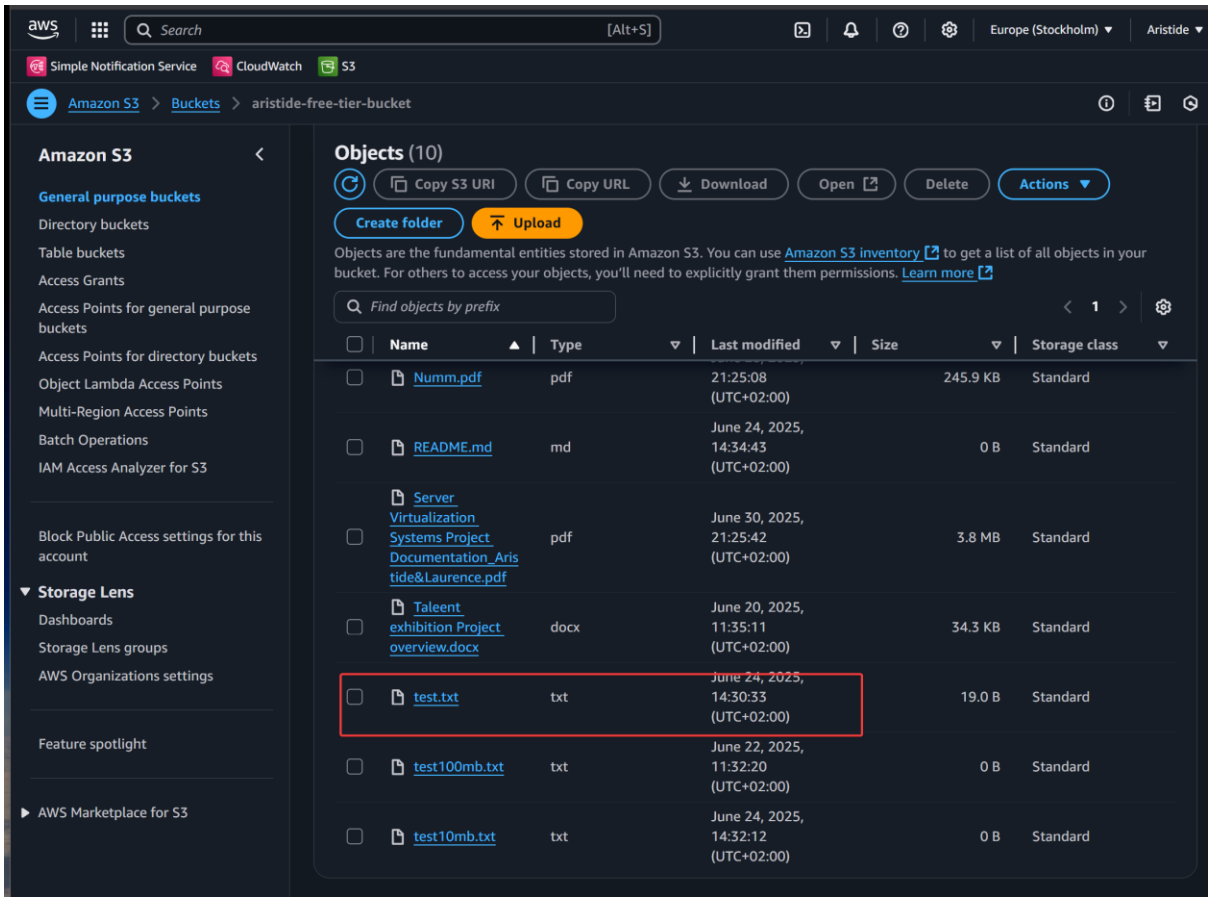


Figure 15:AWS S3 Console showing my-free-tier-bucket with test.txt

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
```

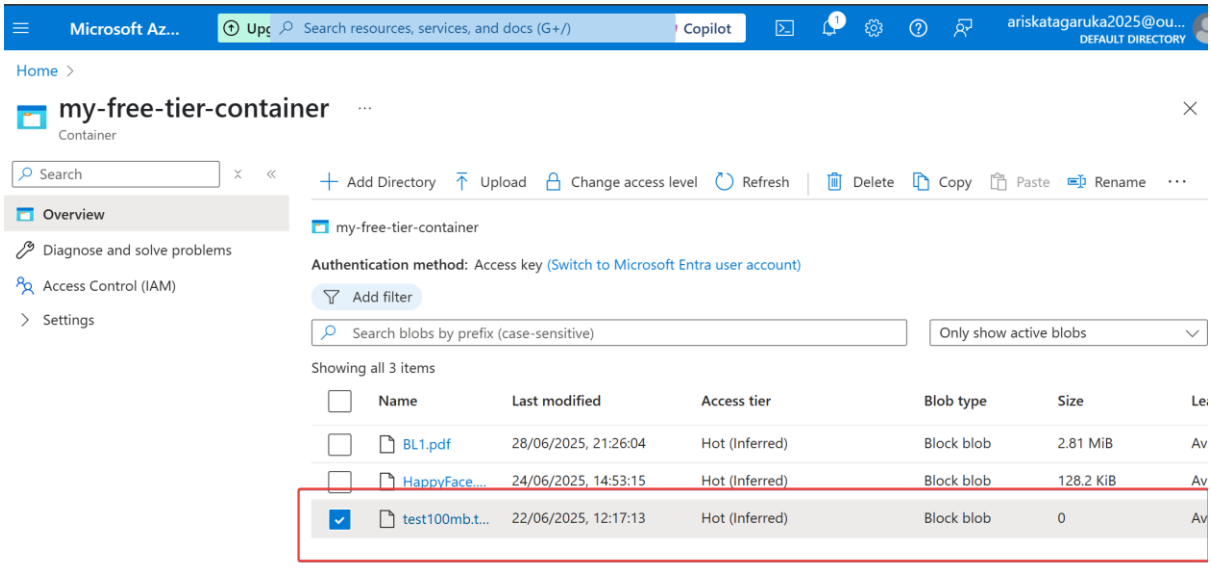


Figure 16: Azure Portal showing my-free-tier-container with test100mb.txt.
```

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
```
```

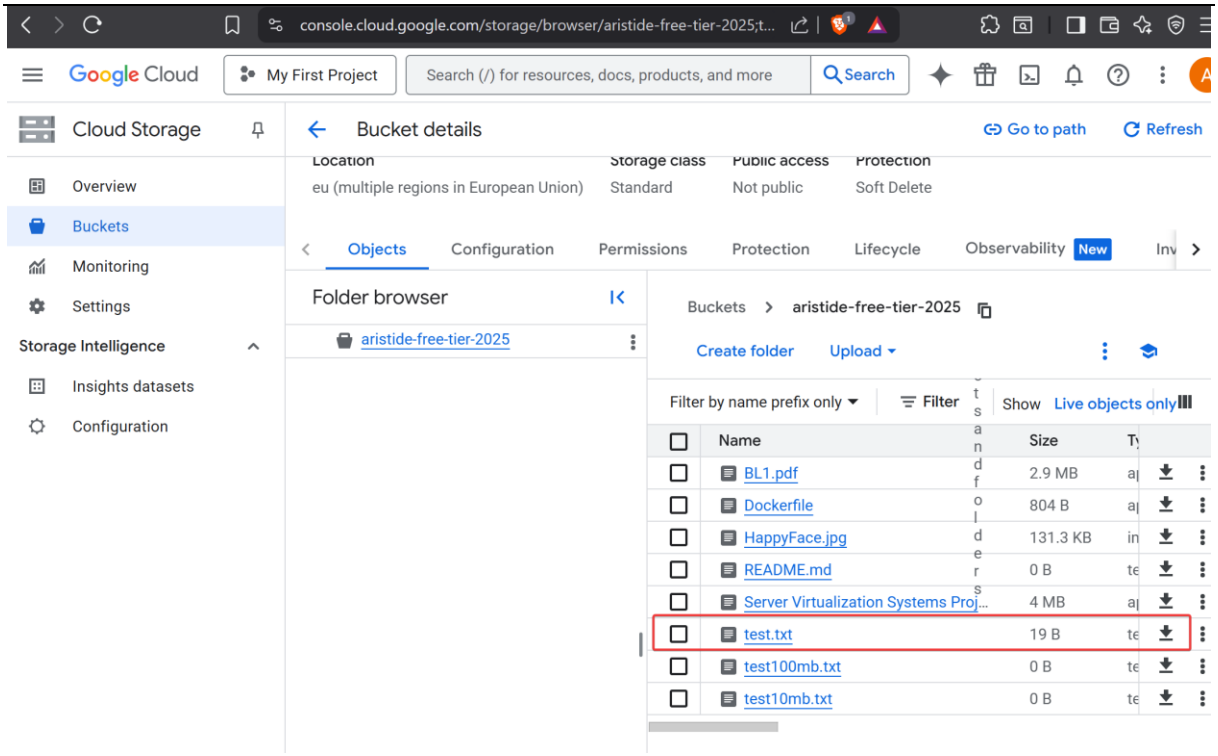


Figure 17: GCP Console showing my-free-tier-bucket with test.txt.

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
```

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

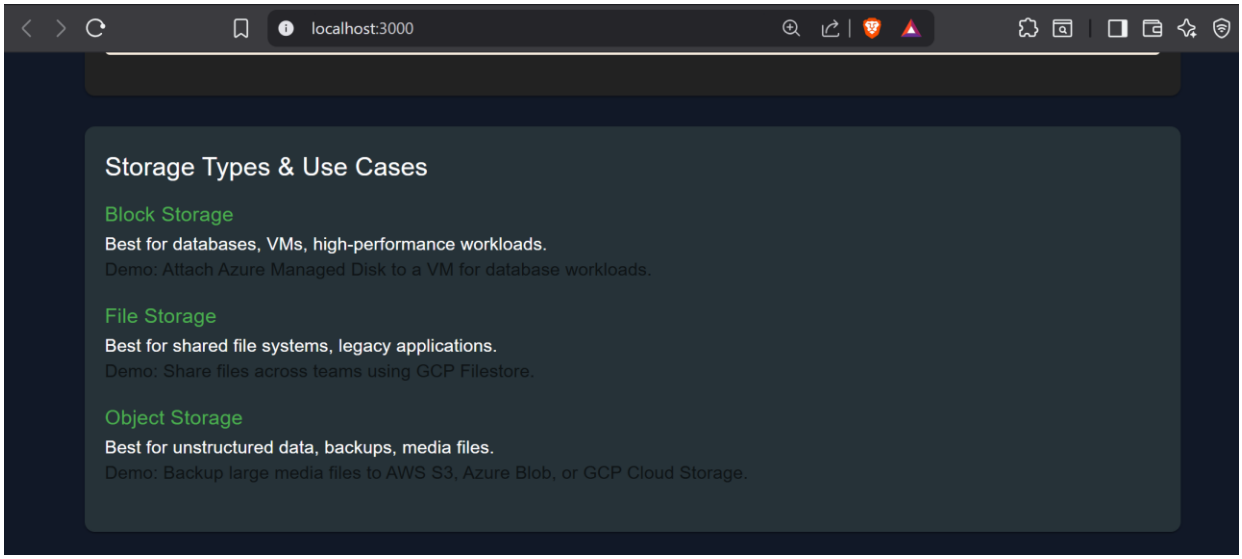


Figure 18: Browser showing mock endpoint responses in StorageInfo.js

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
```

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.

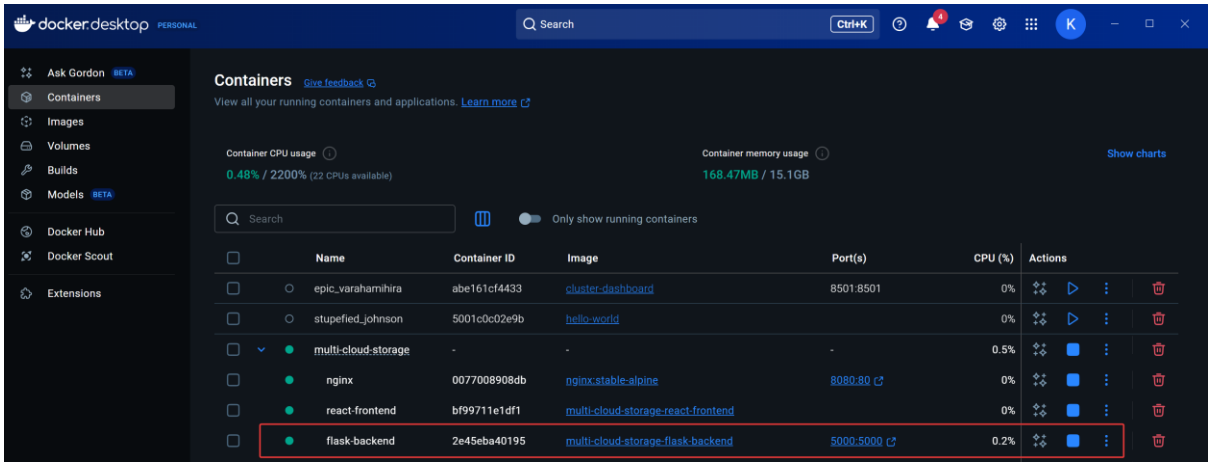


Figure 20:Docker desktop showing Flask backend on port:5000:5000

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.

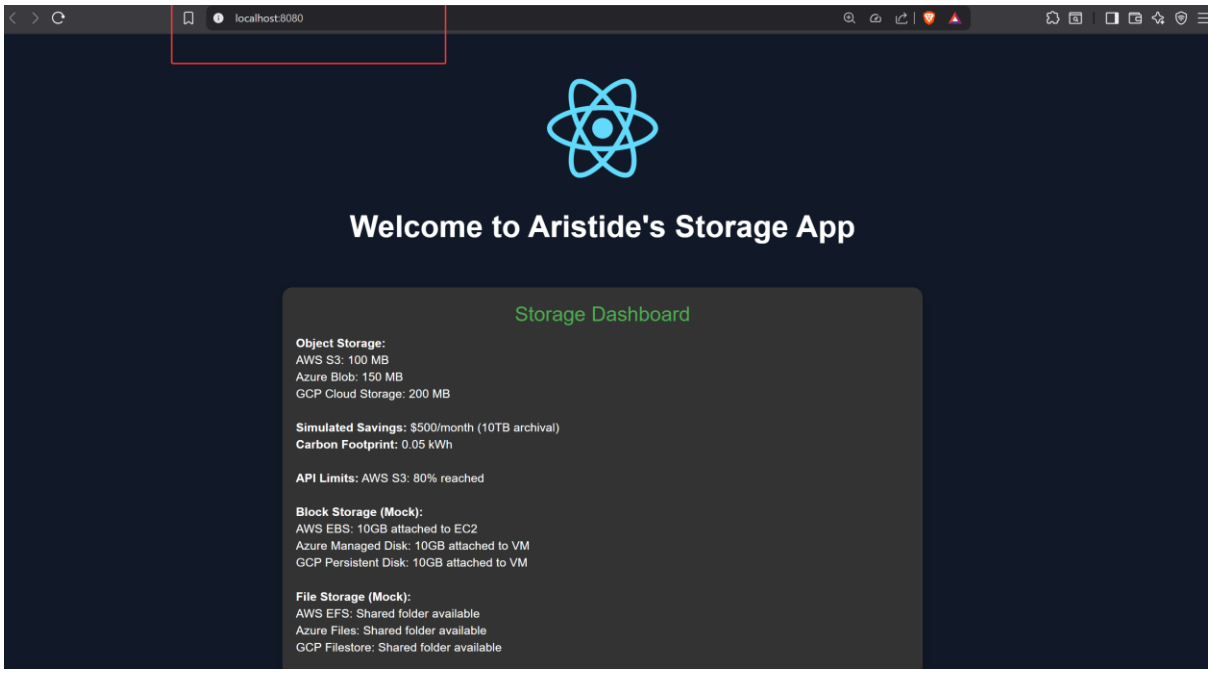


Figure 21: WebUI showing browser at http://localhost:80.

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:  # Define services
  flask-backend:  # Backend service
    build: ./backend  # Build from backend directory
    ports:  # Map ports
      - "5000:5000"  # Map host port 5000 to container port 5000
    environment:  # Set environment variables
      - 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:  # Mount 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:  # Map ports
      - "80:80"  # Map host port 80 to container port 80
  redis:  # Redis service
    image: redis:latest  # Use latest Redis image
    ports:  # Map ports
      - "6379:6379"  # Map host port 6379 to container port 6379
  nginx:  # Nginx service
    image: nginx:latest  # Use latest Nginx image
    ports:  # Map ports
      - "8080:80"  # Map host port 8080 to container port 80
    volumes:  # 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.

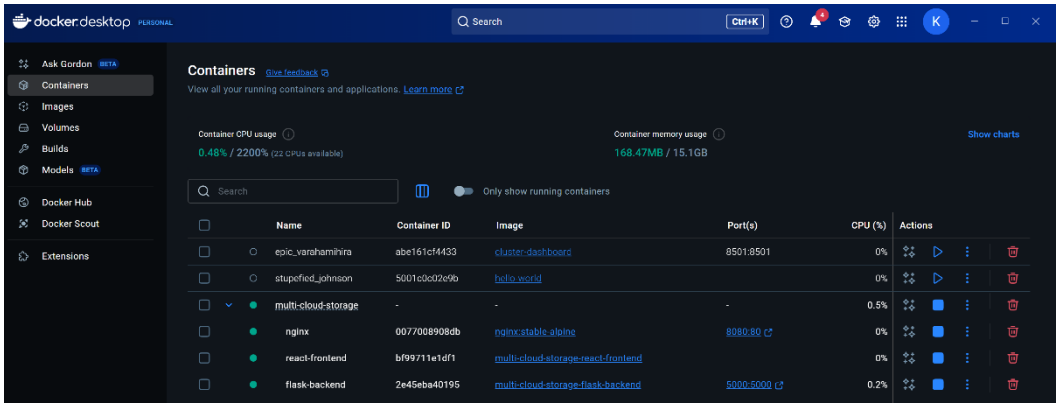


Figure 22: Docker desktop showing docker-compose up at http://localhost:8080.

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`

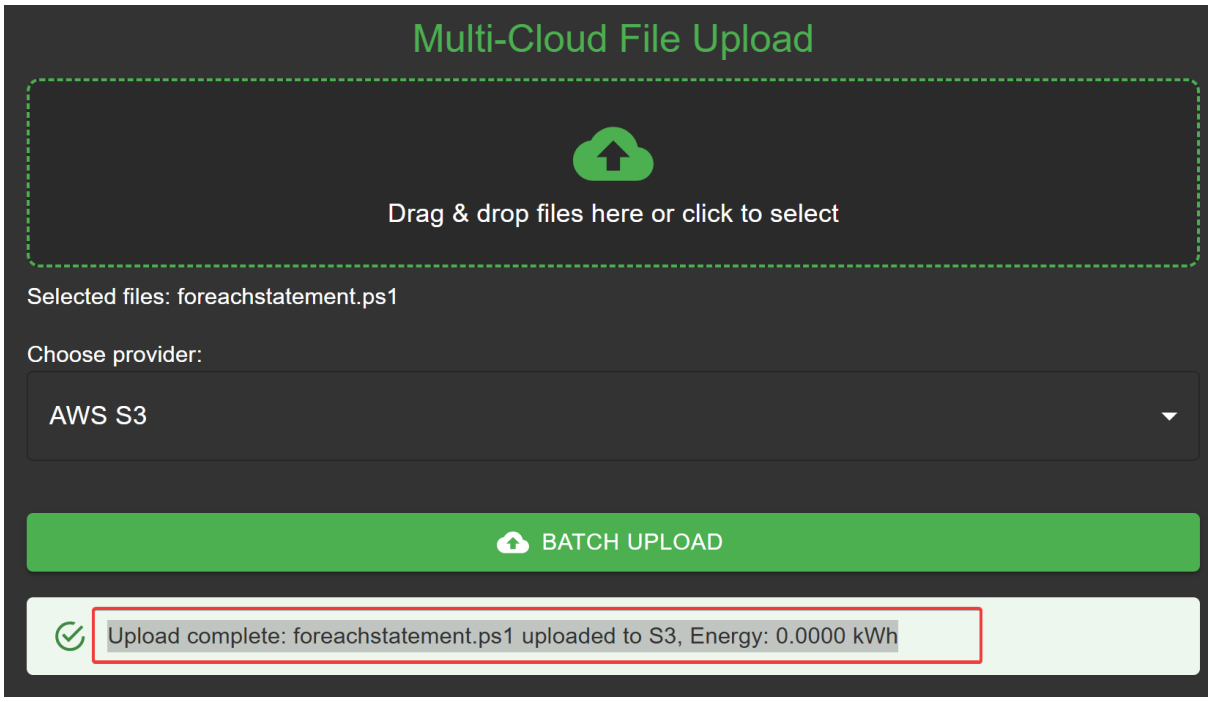
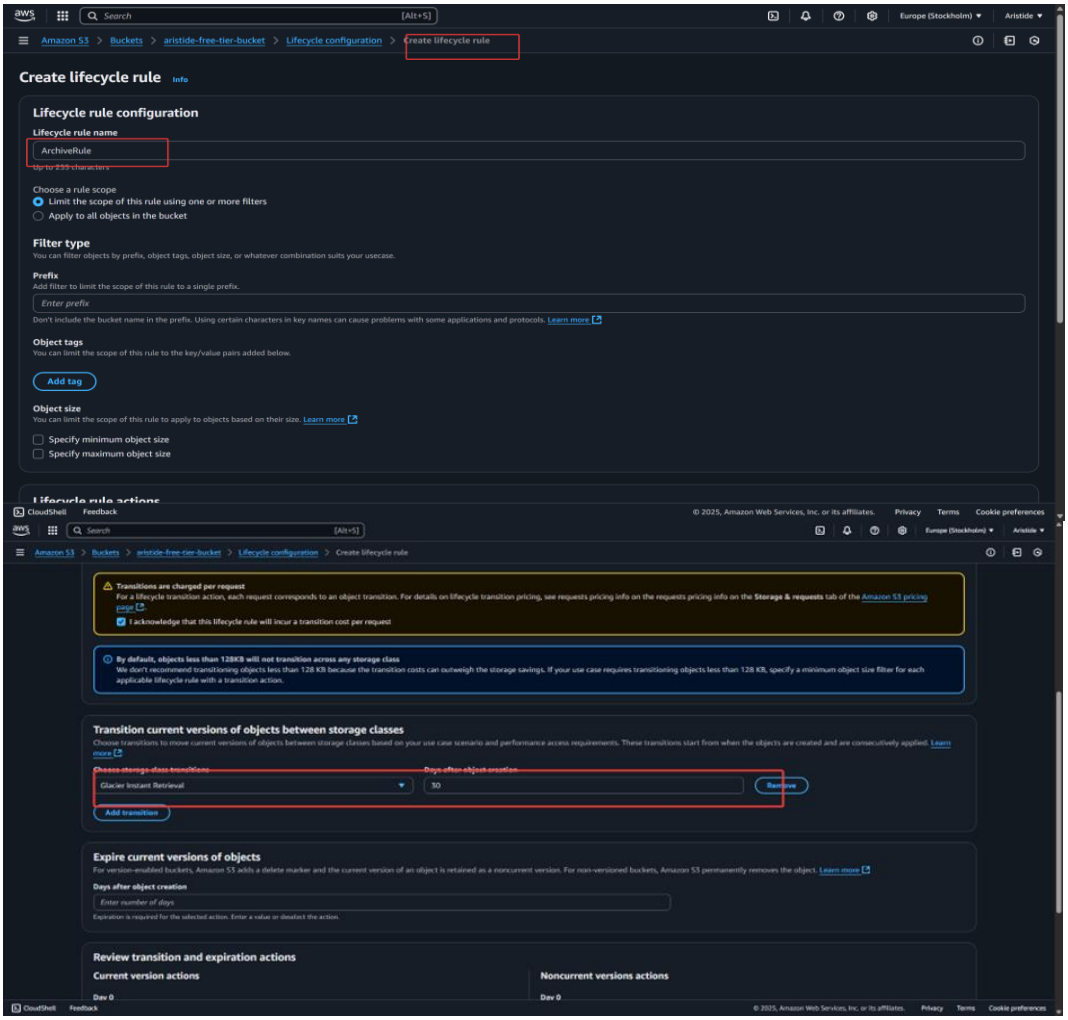


Figure 23:Browser showing upload response with energy estimation.

25. Lifecycle Policies:

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



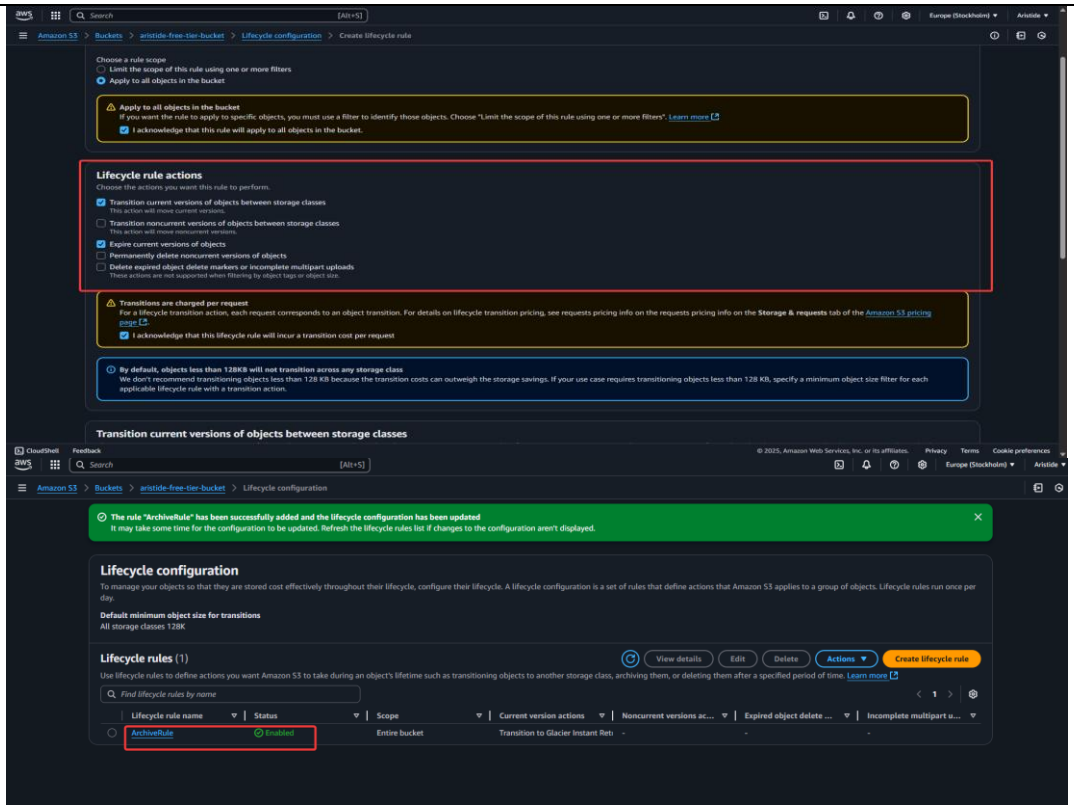


Figure 24:AWS Console showing lifecycle rule.

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

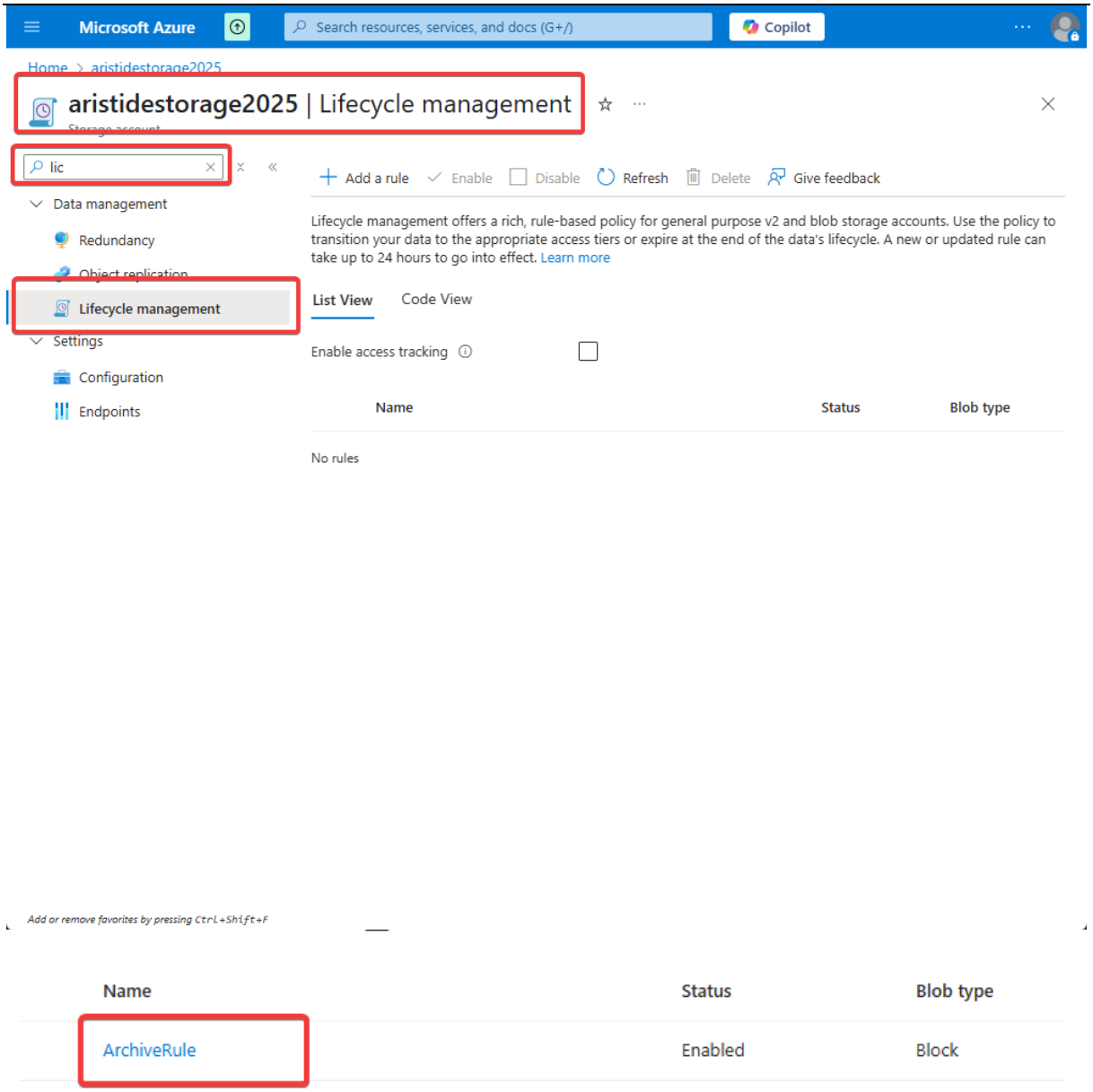


Figure 25: Azure Portal showing lifecycle rule.

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

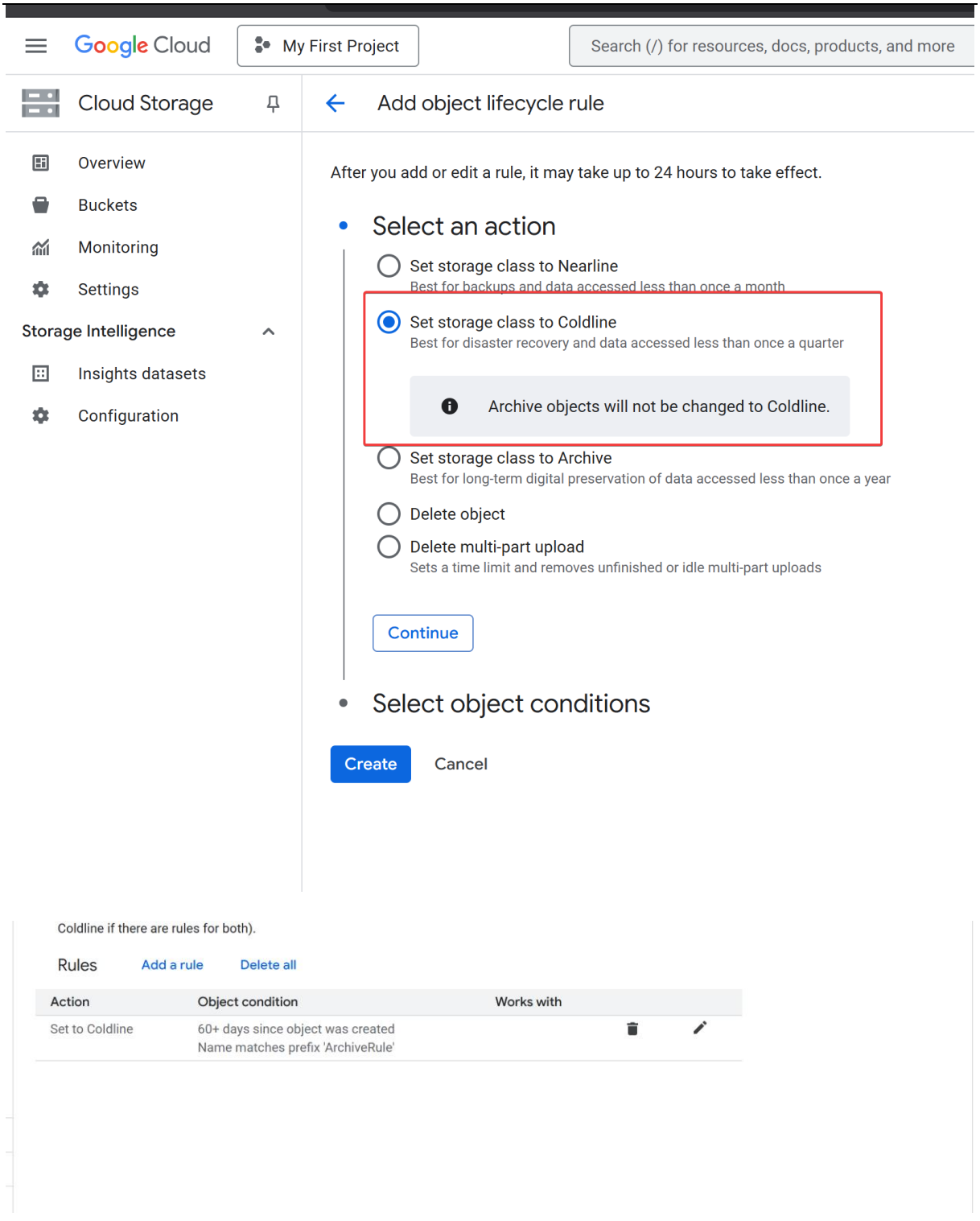
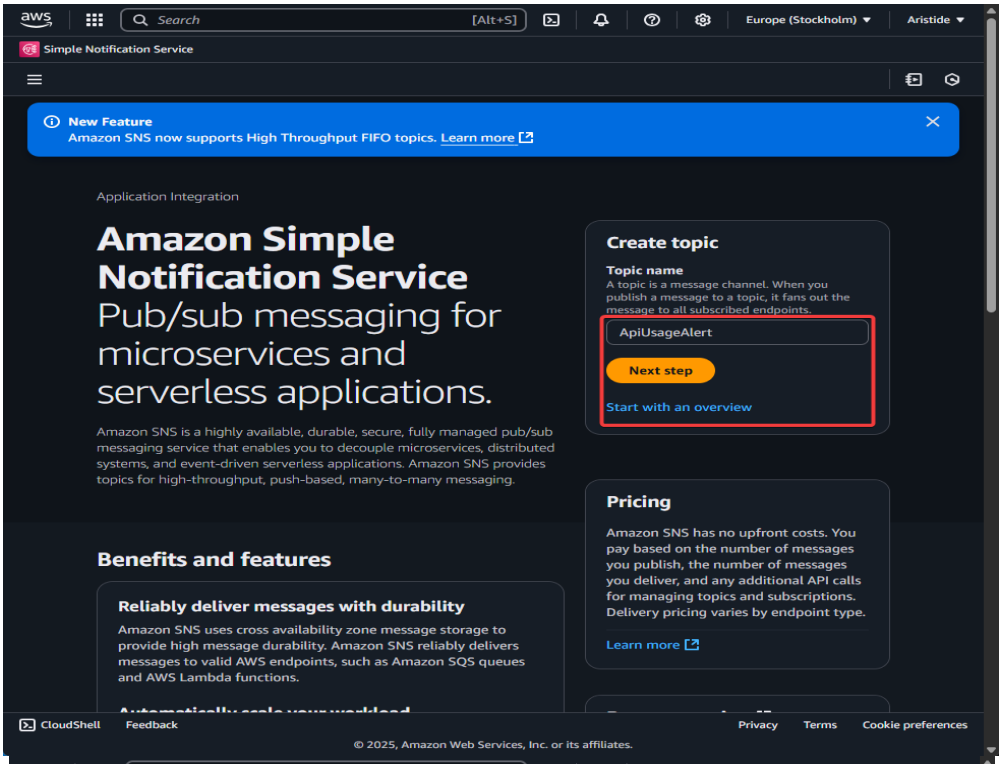


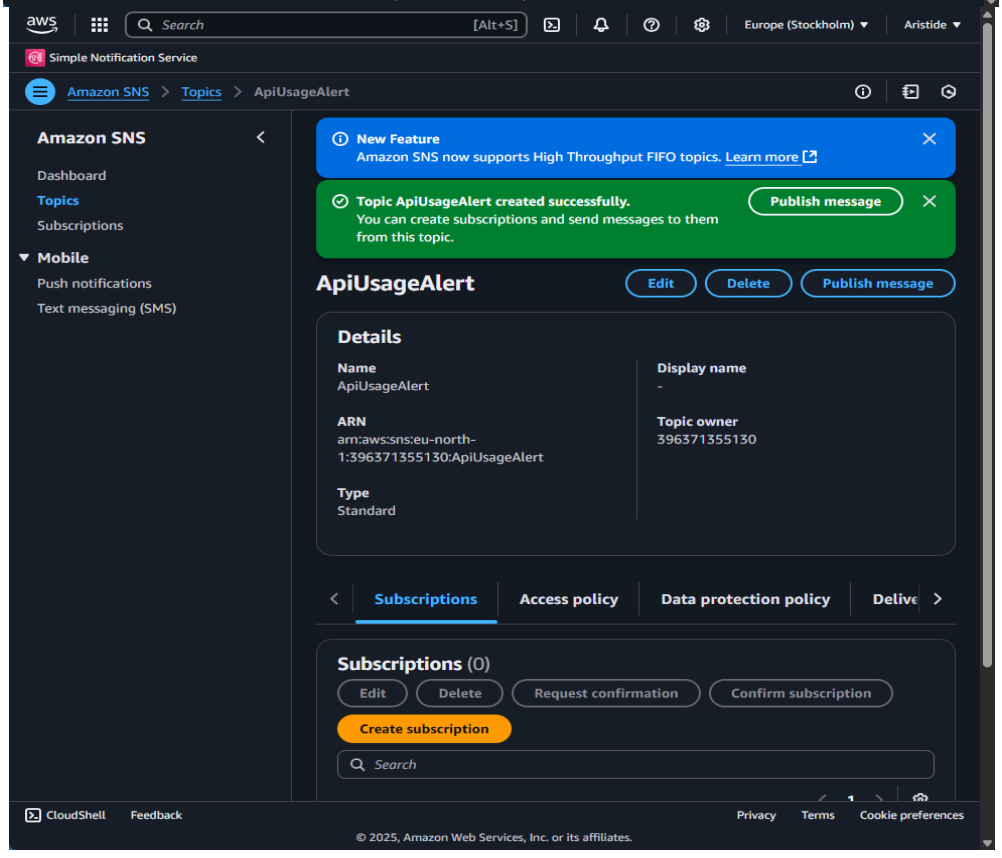
Figure 26: GCP Console showing lifecycle rule.

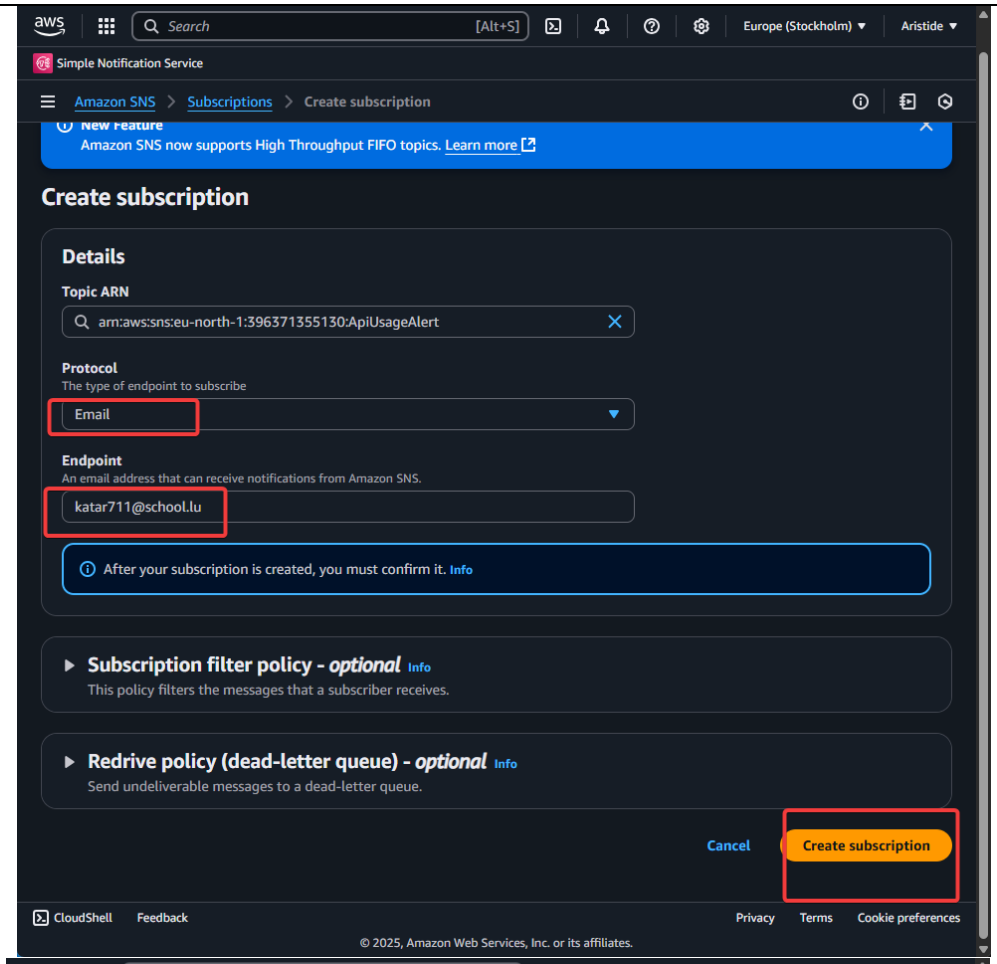
26. API Usage Alerts:

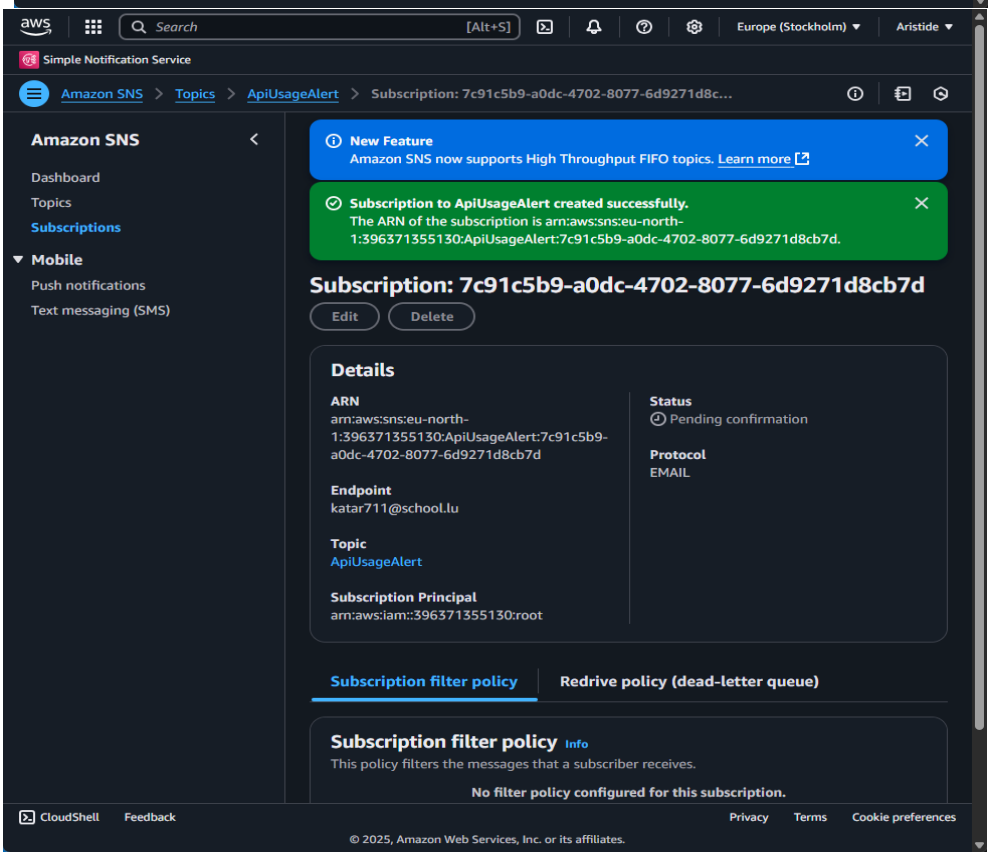
- **AWS SNS/CloudWatch**:
  - In AWS Console: SNS > Topics > Create Topic: ApiUsageAlert, add email subscription.

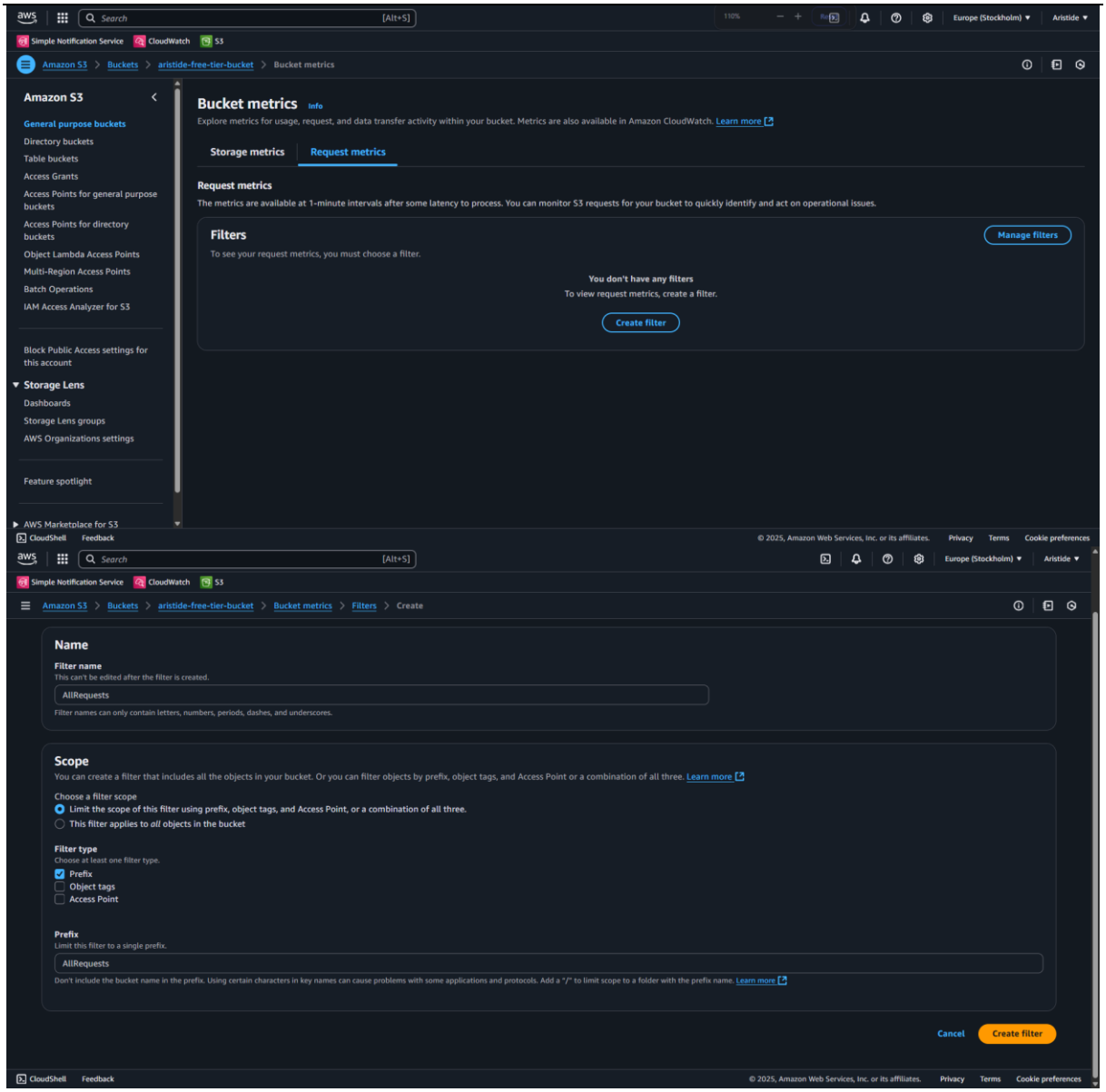
- In CloudWatch: Create alarm for S3 NumberOfRequests > 16,000 (80% of 20,000 GET limit), notify ApiUsageAlert.

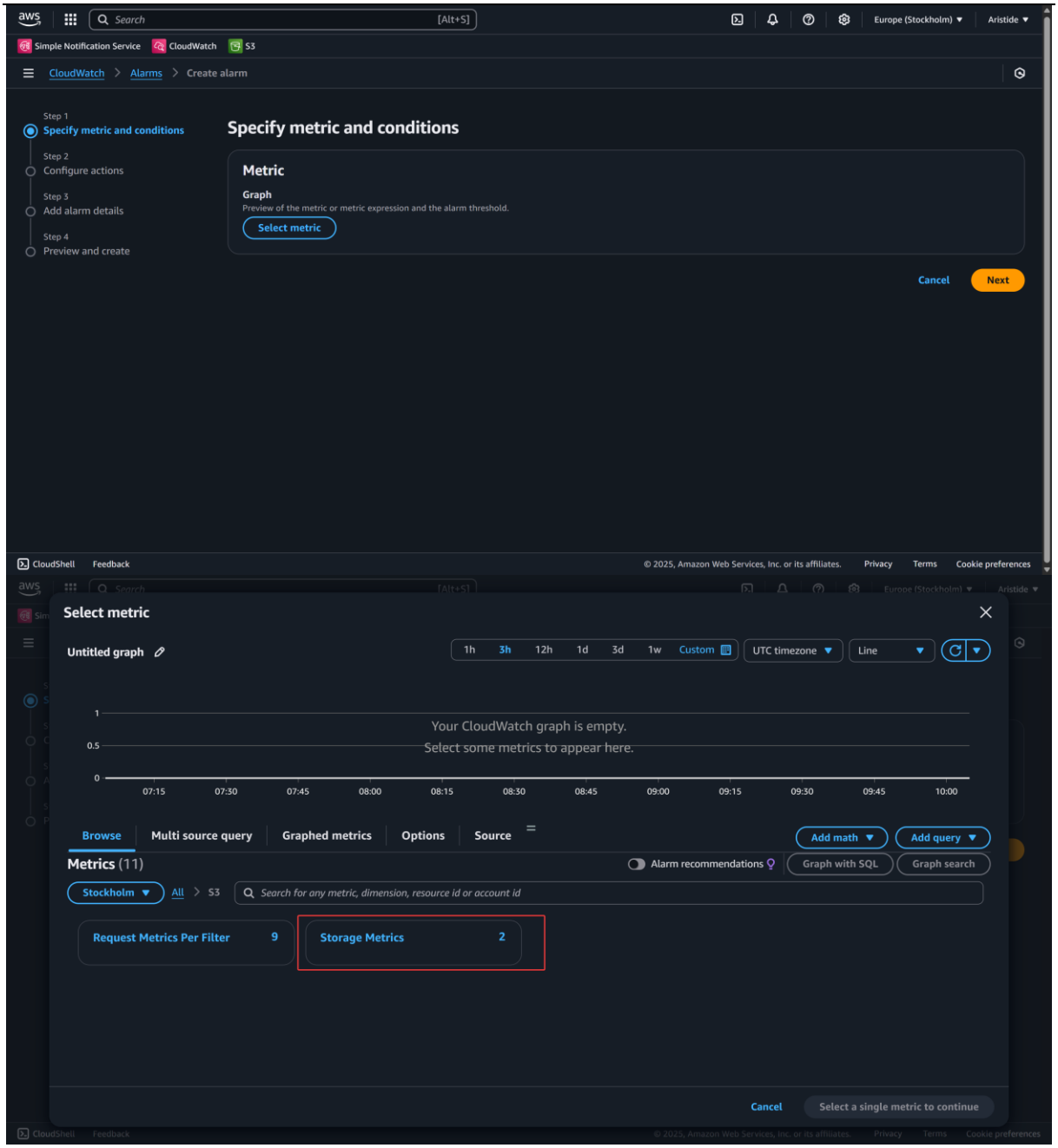


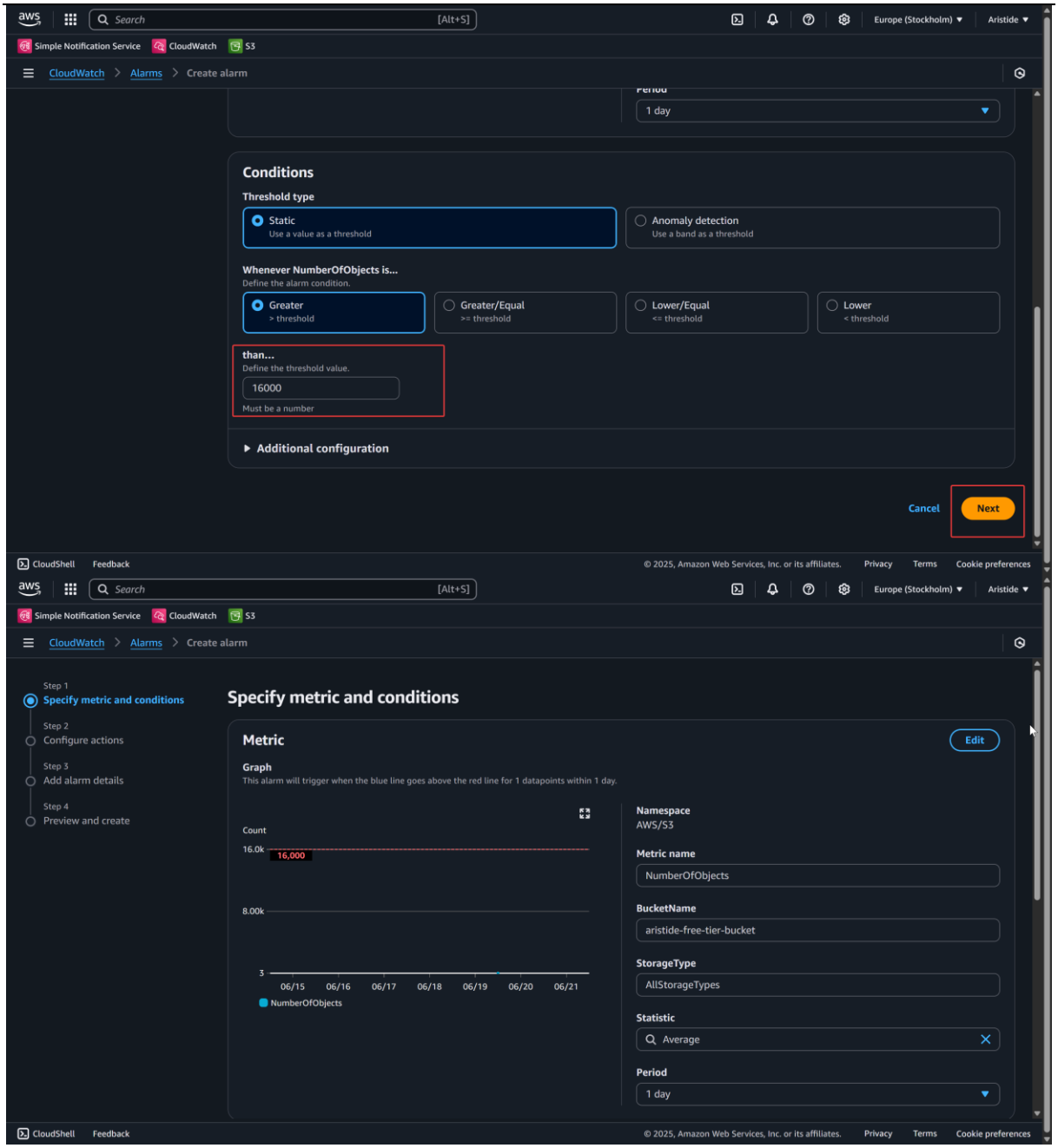


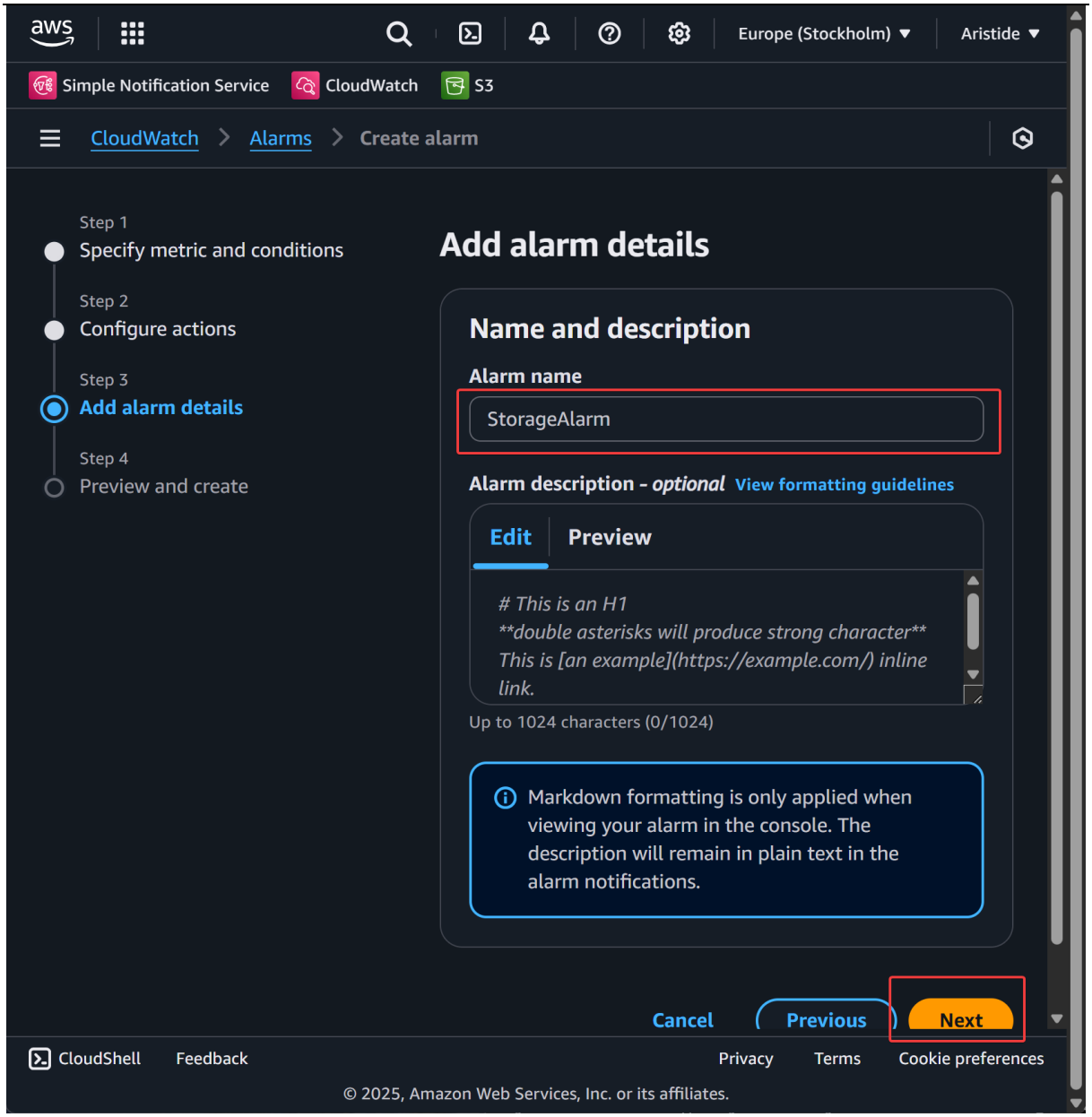

Simple Notification Service
Amazon SNS > Subscriptions > Create subscription
New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more
Create subscription
Details
Topic ARN
arn:aws:sns:eu-north-1:396371355130:ApiUsageAlert
Protocol
The type of endpoint to subscribe
Email
Endpoint
An 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.
Cancel
Create subscription
CloudShell
Feedback
Privacy
Terms
Cookie preferences
© 2025, Amazon Web Services, Inc. or its affiliates.


Simple Notification Service
Amazon SNS > Topics > ApiUsageAlert > Subscription: 7c91c5b9-a0dc-4702-8077-6d9271d8c...
Amazon SNS
Dashboard
Topics
Subscriptions
Mobile
Push notifications
Text messaging (SMS)
New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more
Subscription to ApiUsageAlert created successfully.
The ARN of the subscription is arn:aws:sns:eu-north-1:396371355130:ApiUsageAlert:7c91c5b9-a0dc-4702-8077-6d9271d8cb7d.
Subscription: 7c91c5b9-a0dc-4702-8077-6d9271d8cb7d
Edit
Delete
Details
ARN
arn:aws:sns:eu-north-1:396371355130:ApiUsageAlert:7c91c5b9-a0dc-4702-8077-6d9271d8cb7d
Status
Pending confirmation
Protocol
EMAIL
Endpoint
katar711@school.lu
Topic
ApiUsageAlert
Subscription Principal
arn:aws:iam::396371355130:root
Subscription filter policy
Redrive policy (dead-letter queue)
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.


Amazon S3
Bucket metrics
Explore metrics for usage, request, and data transfer activity within your bucket. Metrics are also available in Amazon CloudWatch.
Storage metrics
Request 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
Manage 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
Name
Filter name
This can't be edited after the filter is created.
AllRequests
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.
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.
AllRequests
Cancel
Create filter


Search [Alt+S]
Europe (Stockholm)
Aristide
Simple Notification Service
CloudWatch
S3
CloudWatch > Alarms > Create alarm
Step 1
Specify metric and conditions
Step 2
Configure actions
Step 3
Add alarm details
Step 4
Preview and create
Specify metric and conditions
Metric
Graph
Preview of the metric or metric expression and the alarm threshold.
Select metric
Cancel
Next
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Select metric
Untitled graph
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
Line
Your CloudWatch graph is empty.
Select some metrics to appear here.
07:15
07:30
07:45
08:00
08:15
08:30
08:45
09:00
09:15
09:30
09:45
10:00
Browse
Multi source query
Graphed metrics
Options
Source
Add math
Add query
Metrics (11)
Alarm recommendations
Graph with SQL
Graph search
Stockholm
All > S3
Search for any metric, dimension, resource id or account id
Request Metrics Per Filter 9
Storage Metrics 2
Cancel
Select a single metric to continue


Search [Alt+S]
Europe (Stockholm)
Aristide
Simple Notification Service
CloudWatch
S3
CloudWatch > Alarms > Create alarm
1 day
Conditions
Threshold type
Static
Use a value as a threshold
Anomaly detection
Use a band as a threshold
Whenever NumberOfObjects is...
Define the alarm condition.
Greater
> threshold
Greater/Equal
>= threshold
Lower/Equal
<= threshold
Lower
< threshold
than...
Define the threshold value.
16000
Must be a number
Additional configuration
Cancel
Next
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Step 1
Specify metric and conditions
Step 2
Configure actions
Step 3
Add alarm details
Step 4
Preview and create
Specify metric and conditions
Metric
Edit
Graph
This alarm will trigger when the blue line goes above the red line for 1 datapoints within 1 day.
Count
16.0k
16,000
8.00k
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


aws
Europe (Stockholm)
Aristide
Simple Notification Service
CloudWatch
S3
CloudWatch > Alarms > Create alarm
Step 1
Specify metric and conditions
Step 2
Configure actions
Step 3
Add alarm details
Step 4
Preview and create
Add alarm details
Name and description
Alarm name
StorageAlarm
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
CloudShell
Feedback
Privacy
Terms
Cookie preferences
© 2025, Amazon Web Services, Inc. or its affiliates.

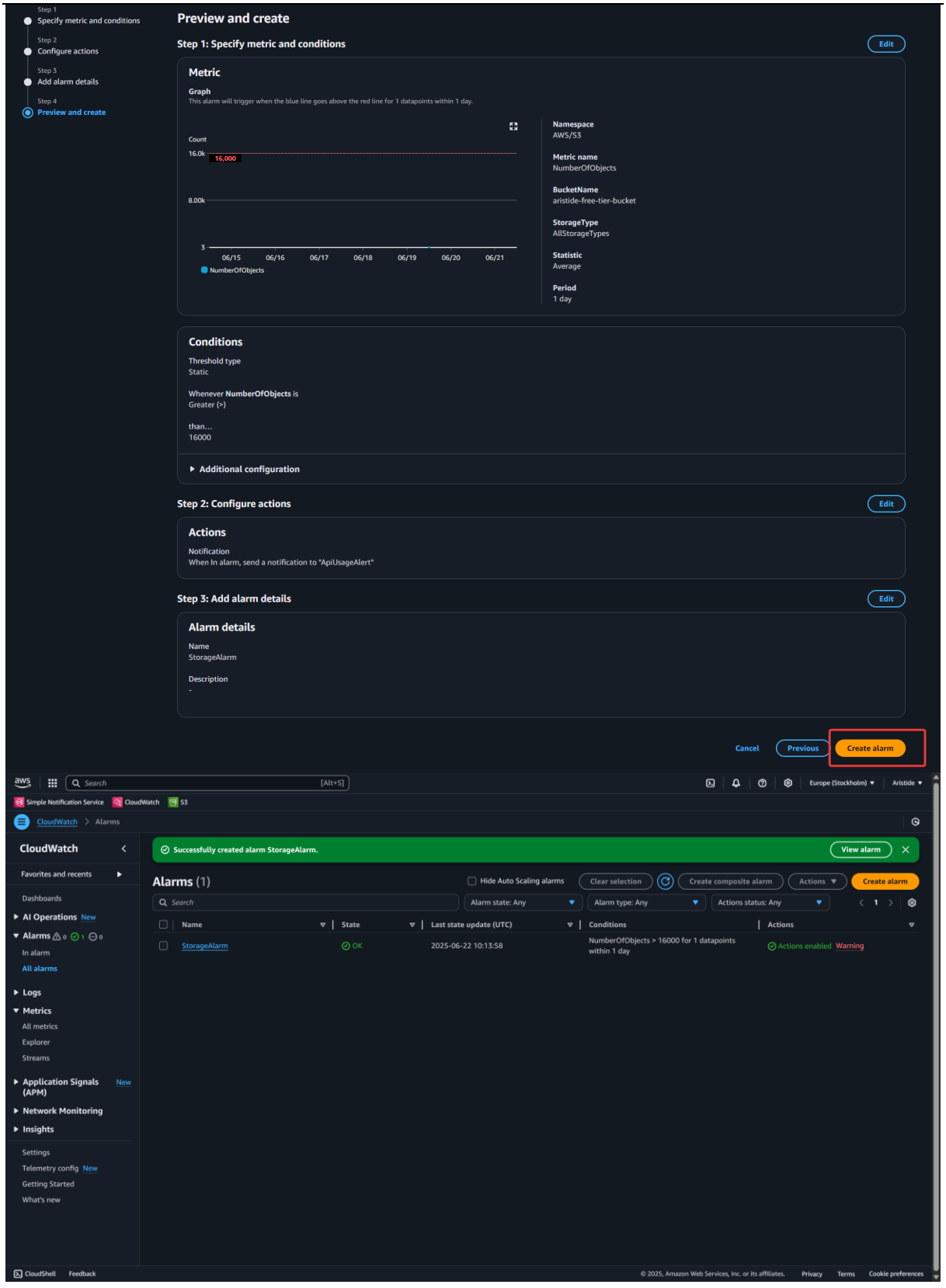
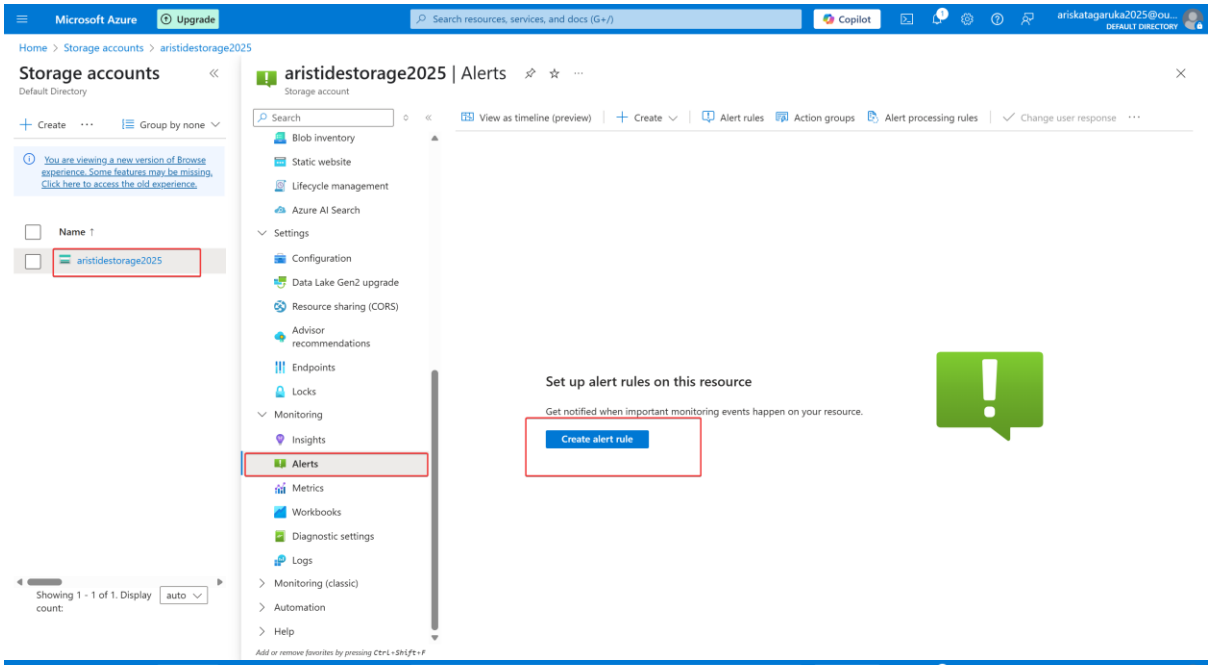
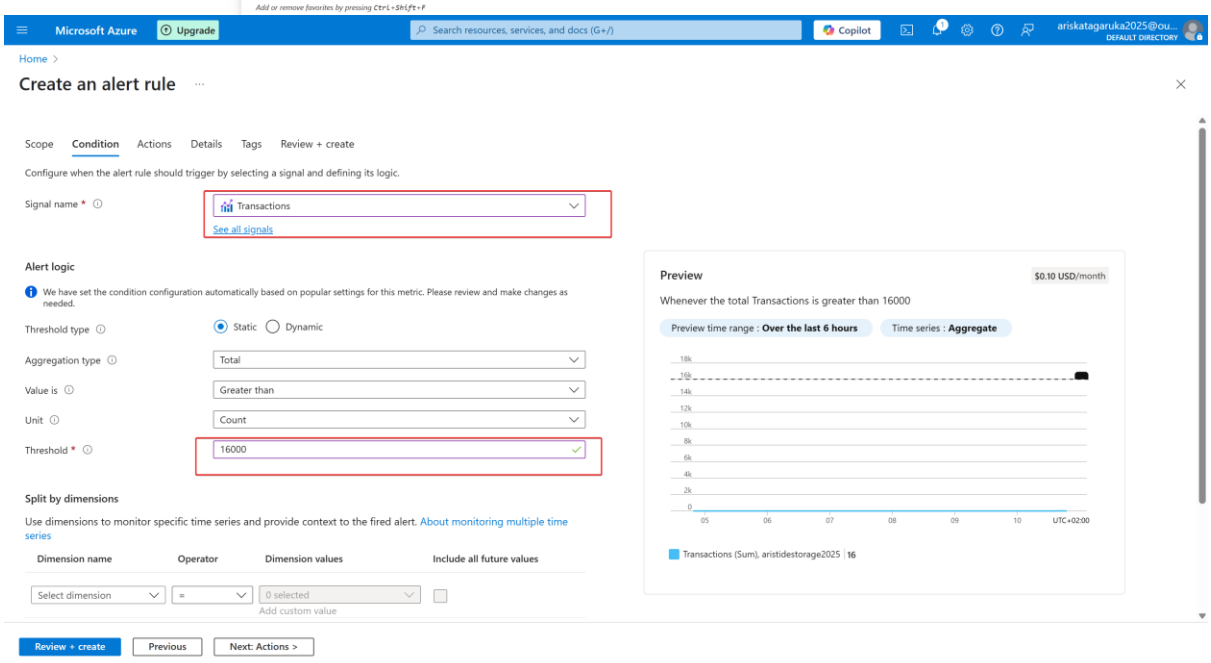


Figure 27: CloudWatch showingthe creation of an alarm.

- **Azure Monitor**:

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





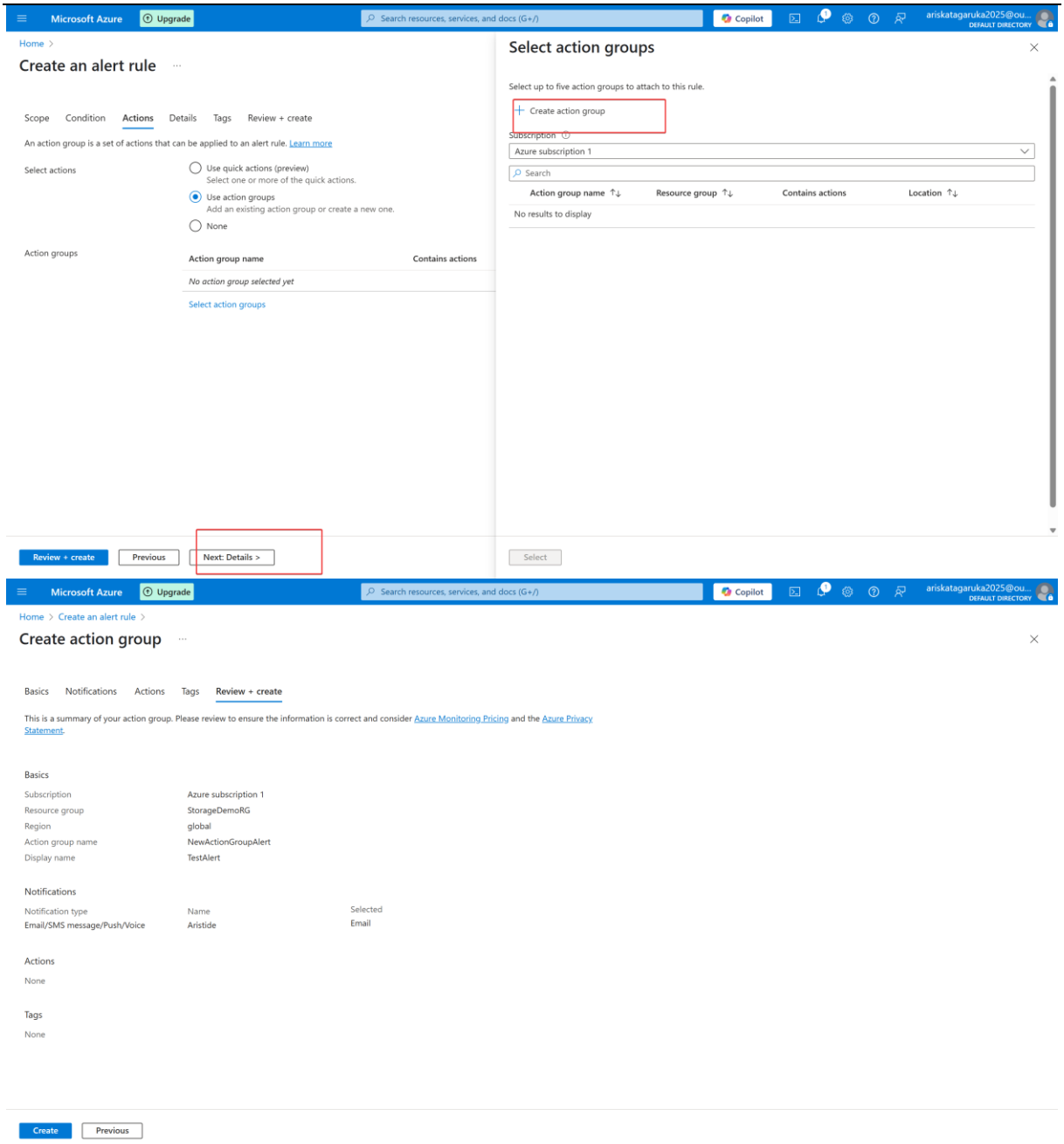
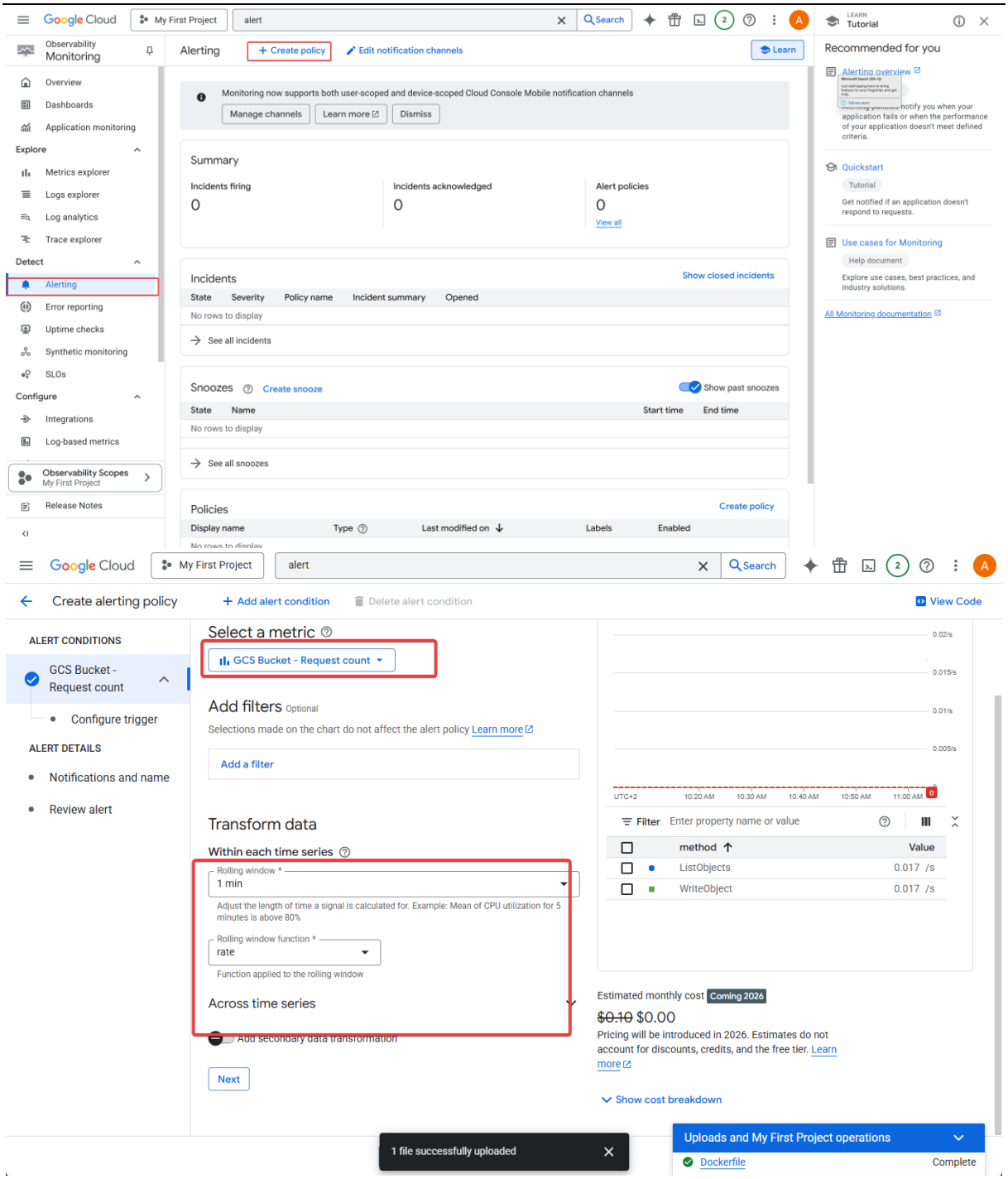
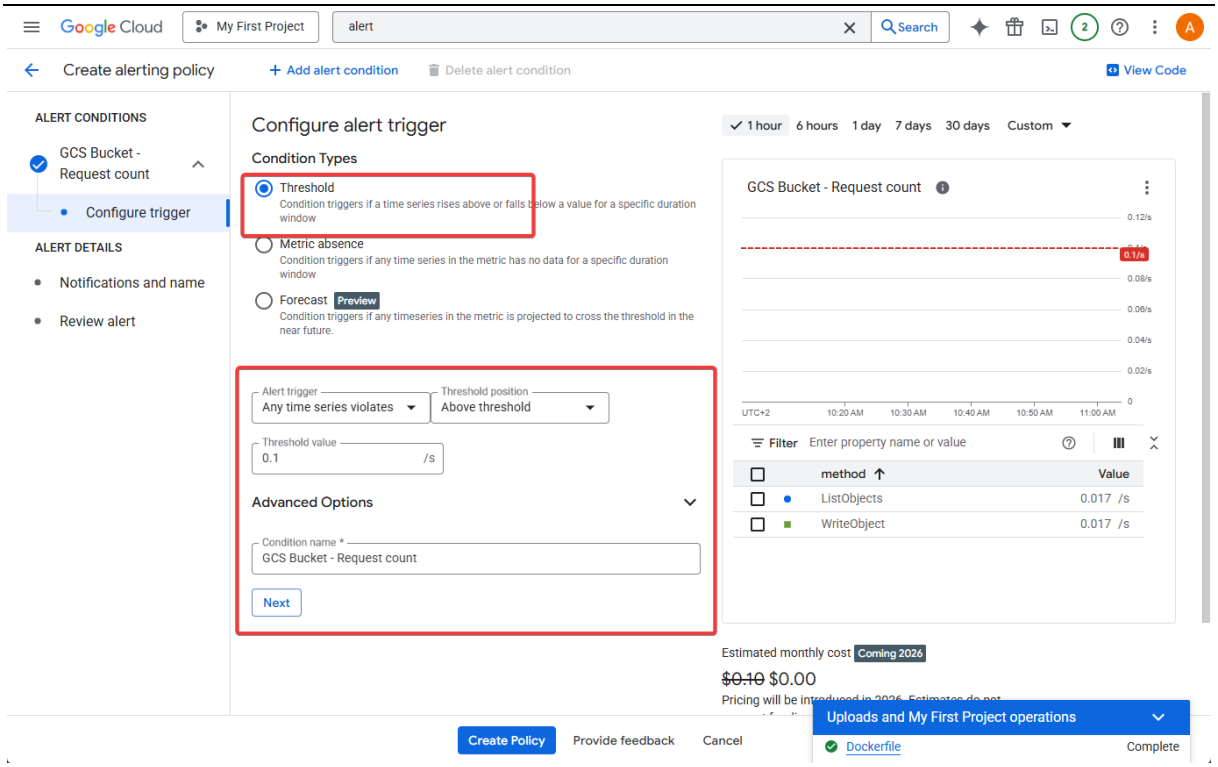
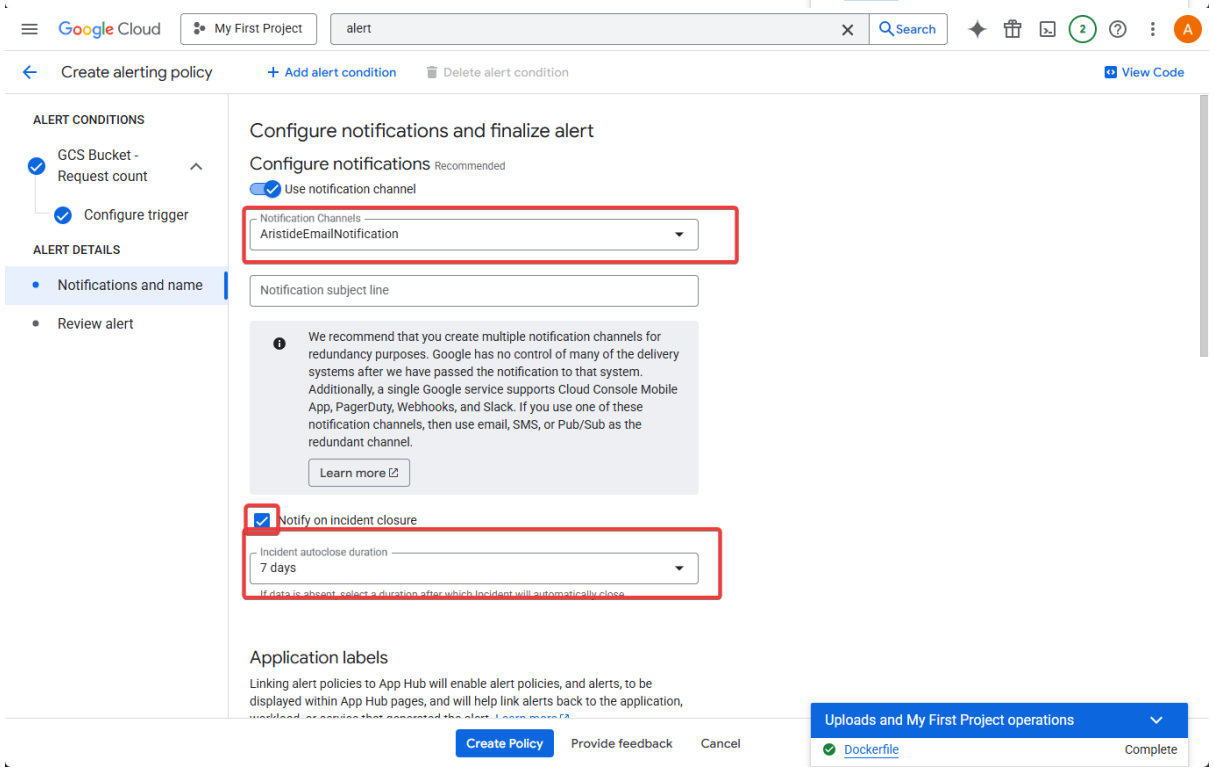


Figure 28: Azure Portal showing the creation alert rule.

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


Google Cloud
My First Project
alert
Search
Observability Monitoring
Alerting
+ Create policy
Edit notification channels
Learn
Overview
Dashboards
Application monitoring
Explore
Metrics explorer
Logs explorer
Log analytics
Trace explorer
Detect
Alerting
Error reporting
Uptime checks
Synthetic monitoring
SLOs
Configure
Integrations
Log-based metrics
Observability Scopes
My First Project
Release Notes
Monitoring now supports both user-scoped and device-scoped Cloud Console Mobile notification channels
Manage channels
Learn more
Dismiss
Summary
Incidents firing
0
Incidents acknowledged
0
Alert policies
0
View all
Incidents
Show closed incidents
State
Severity
Policy name
Incident summary
Opened
No rows to display
See all incidents
Snoozes
Create snooze
Show past snoozes
State
Name
Start time
End time
No rows to display
See all snoozes
Policies
Create policy
Display name
Type
Last modified on
Labels
Enabled
No rows to display
LEARN
Tutorial
Recommended for you
Alerting overview
application fails or when the performance of your application doesn't meet defined criteria.
Quickstart
Tutorial
Get notified if an application doesn't respond to requests.
Use cases for Monitoring
Help document
Explore use cases, best practices, and industry solutions.
All Monitoring documentation
Google Cloud
My First Project
alert
Search
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
Select a metric
GCS Bucket - Request count
Add filters
Optional
Selections made on the chart do not affect the alert policy
Learn more
Add a filter
Transform data
Within each time series
Rolling window *
1 min
Adjust the length of time a signal is calculated for. Example: Mean of CPU utilization for 5 minutes is above 80%
Rolling window function *
rate
Function applied to the rolling window
Across time series
Add secondary data transformation
Next
0.02/s
0.015/s
0.01/s
0.005/s
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 account for discounts, credits, and the free tier.
Learn more
Show cost breakdown
1 file successfully uploaded
Uploads and My First Project operations
Dockerfile
Complete


Google Cloud
My First Project
alert
Search
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
1 hour
6 hours
1 day
7 days
30 days
Custom
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
Create Policy
Provide feedback
Cancel
Uploads and My First Project operations
Dockerfile
Complete


Google Cloud
My First Project
alert
Search
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
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
Uploads and My First Project operations
Dockerfile
Complete

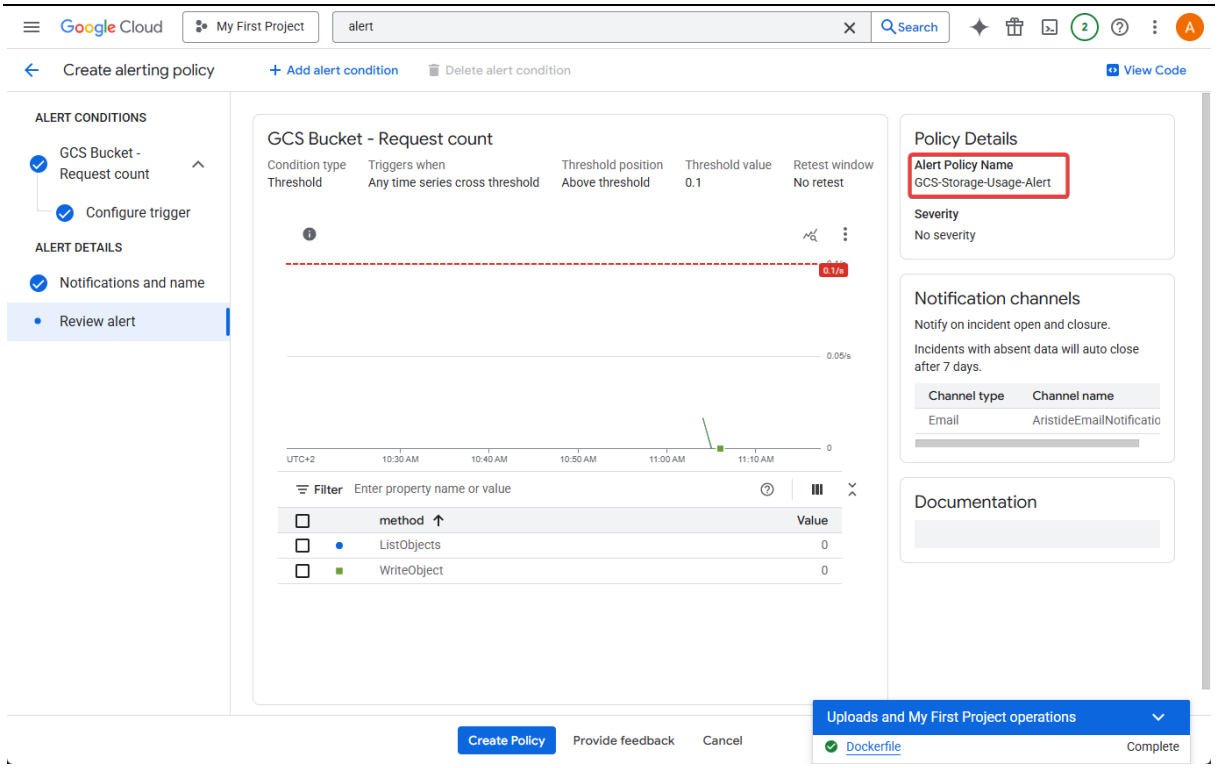


Figure 29: GCP Console showing monitoring policy.

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.

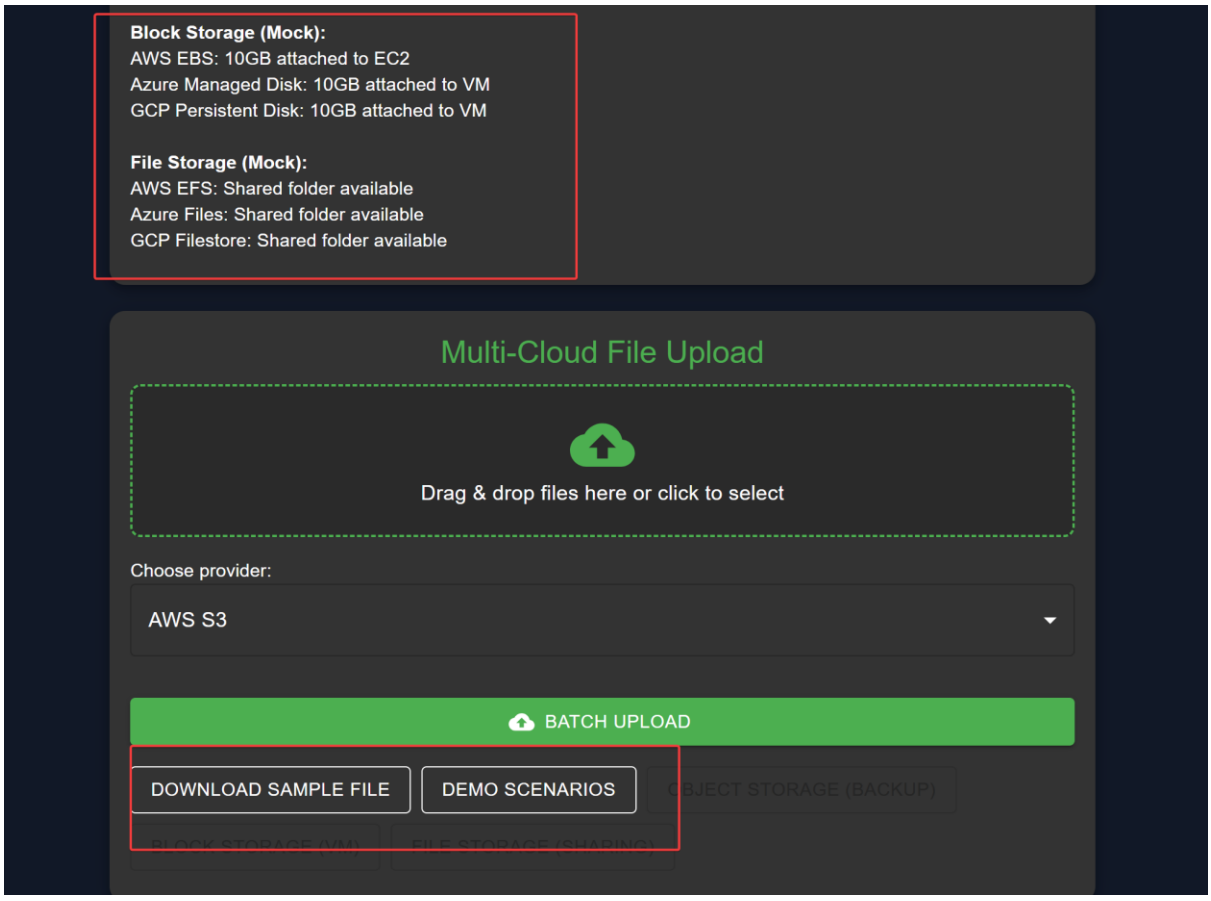


Figure 30: Browser showing upload/download and mock scenario outputs.

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.

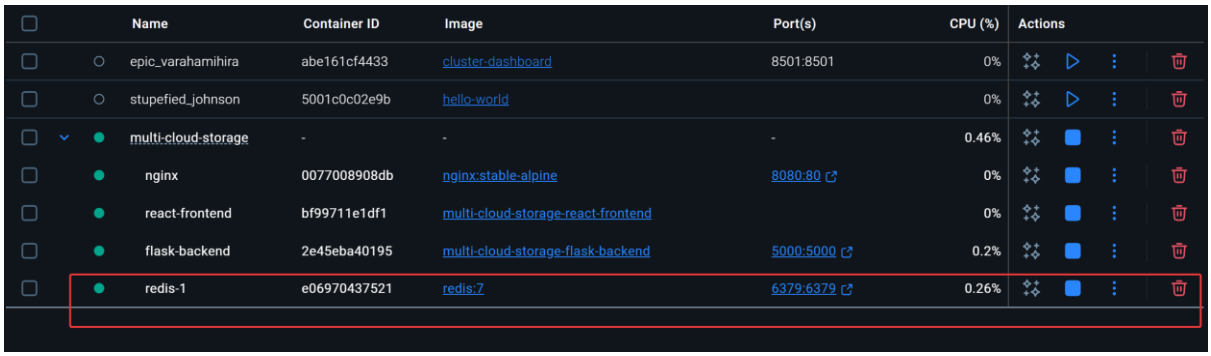


Figure 31: Docker desktop showing Redis container running.

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`

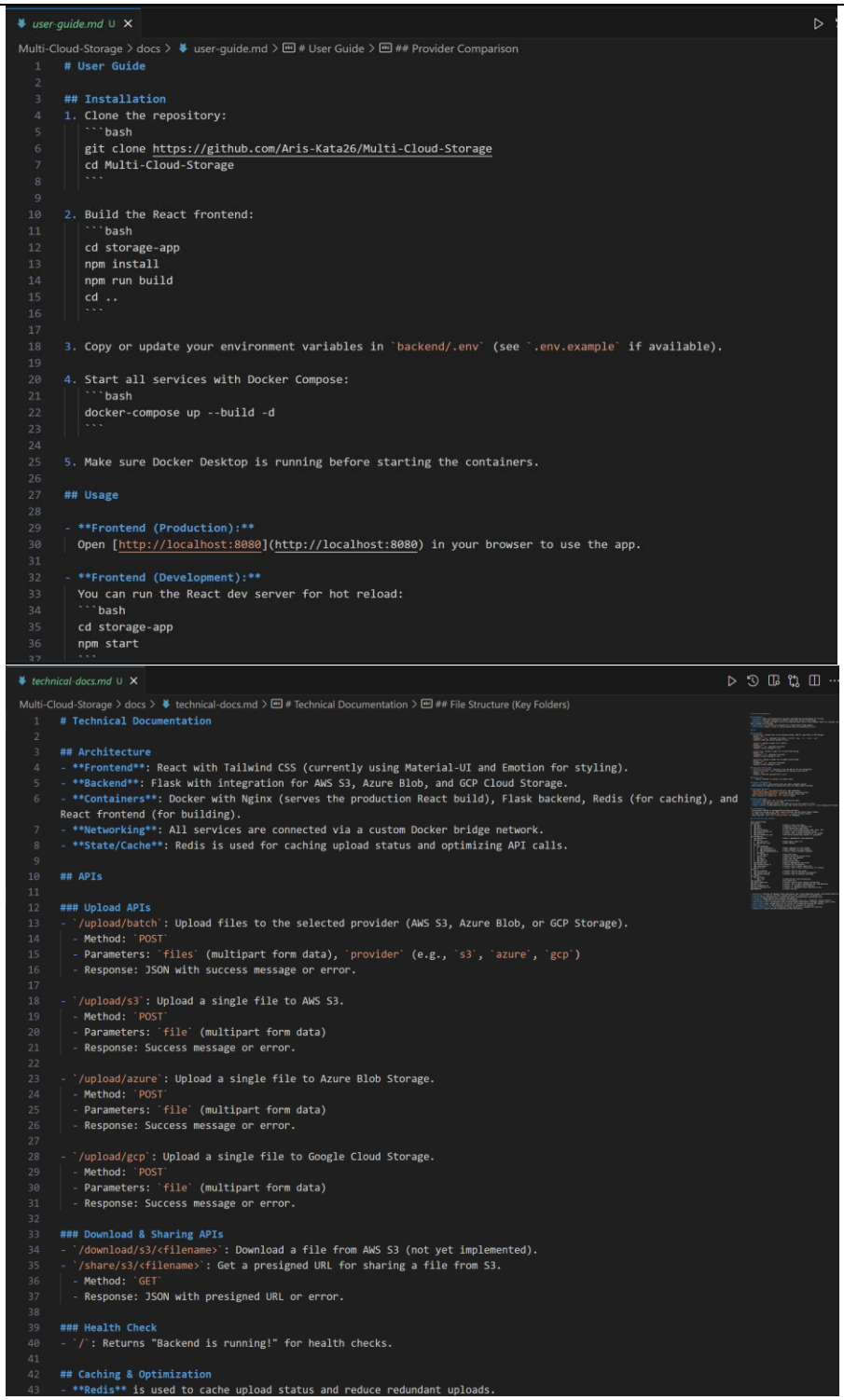


Figure 32: VS Code showing user-guide.md and technical-docs.md

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`

- `- Leveraging GCP's no-egress-fee policy.  # Reduce data transfer costs`
- `- Setting budget alerts at $800 for $1,000/month budget.  # Monitor spending`

`Energy reduction: 20-30% via archival tiers, supporting ESG compliance.  # Sustainability benefits`

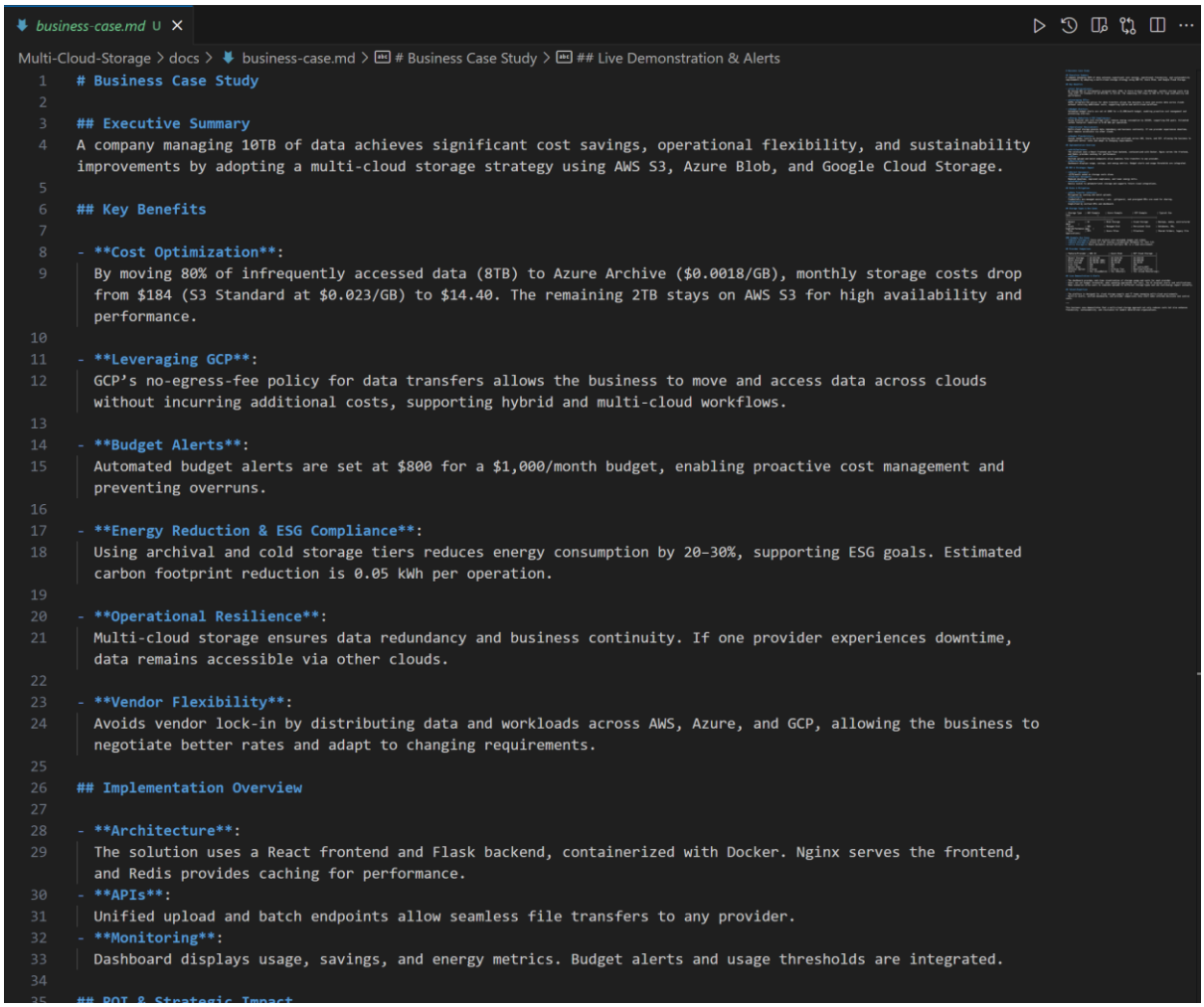


Figure 33: VS code showing business case

14. Technical Architecture

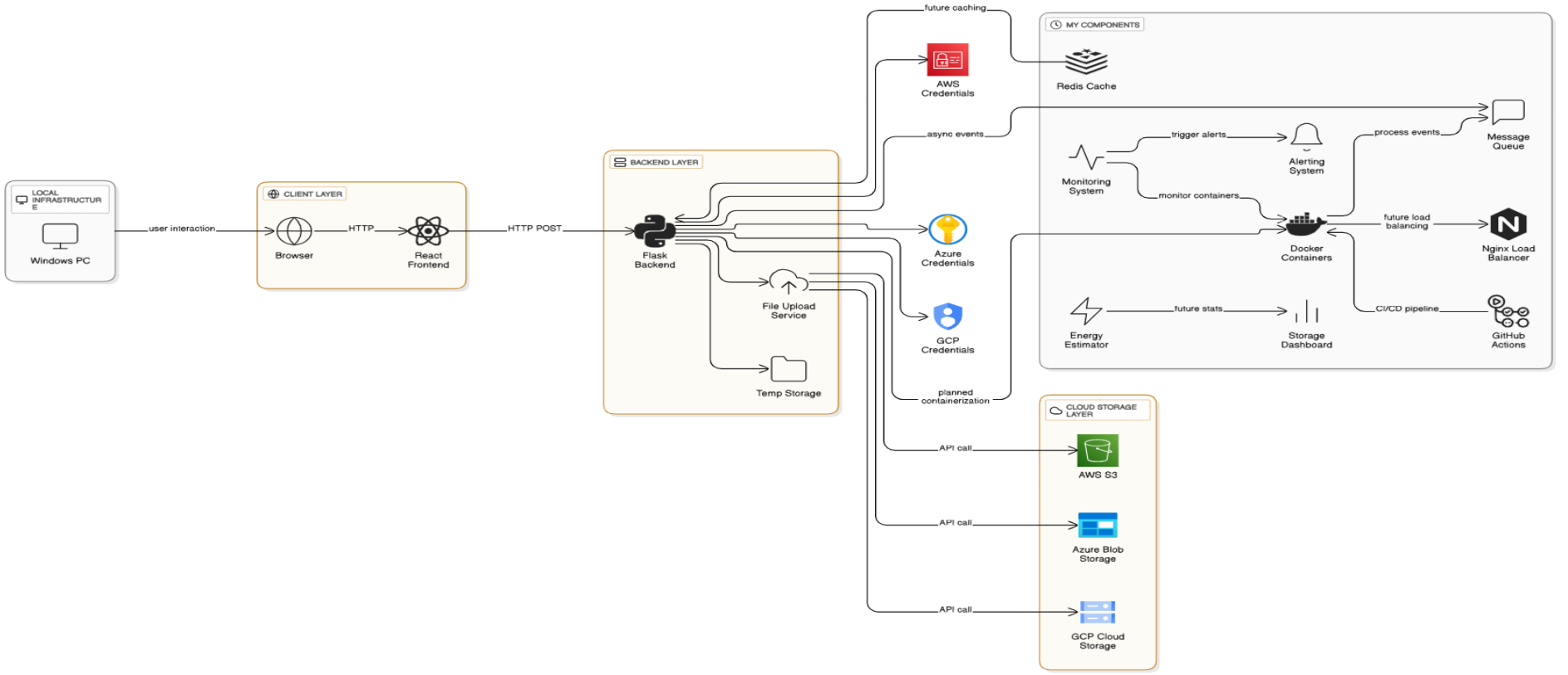


Figure 34: Technical Architecture showing the overview of the project

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

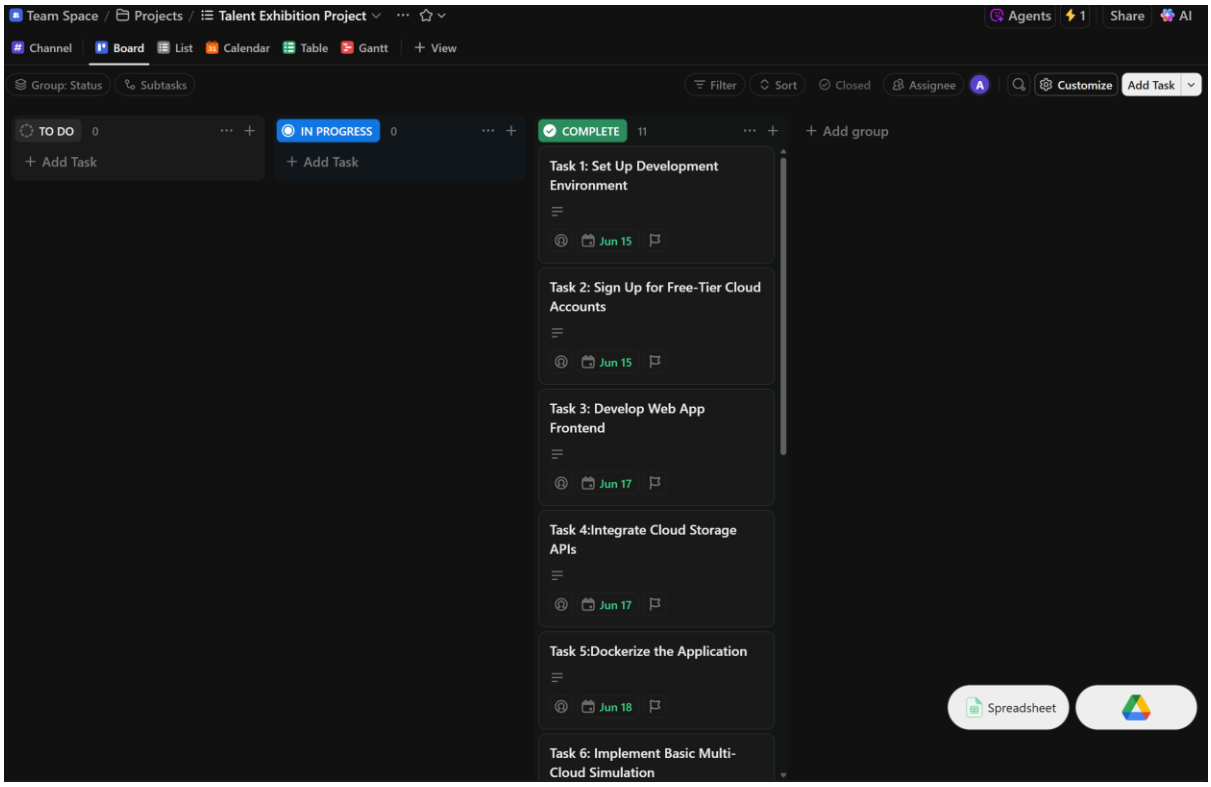


Figure 35: Simple screenshot of the project management tool I used.

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

Thank you for reading