Amazon S3 – Object storage

27 July 2025 00:42

🔘 What is Amazon S3?

Amazon S3 (Simple Storage Service) is like a **Google Drive or Dropbox** – but for developers, companies, and apps.

- You can **store anything**: photos, videos, documents, backups, websites, logs, etc.
- You can access these files anytime, from anywhere on the internet.
- It's extremely **secure, reliable**, and can handle **millions of files**.

☑ **Example**:

Imagine you're running a mobile app for online shopping. You can use S3 to:

- Store product images
- Store user invoices
- Store logs of user activity
- Backup your database daily

🔘 Use Cases of Amazon S3

Use Case	Description
Static Website Hosting	Host HTML, CSS, JS files directly from S3 (no server needed).
Backup & Restore	Backup servers, databases, apps securely.
Log Storage	Store logs from servers, applications, or access logs from AWS services like CloudFront.
Data Lake	Store large-scale data to process using analytics tools.
Application File Storage	Store user-uploaded files, like resumes, profile photos, etc.
Software Delivery	Host downloadable files like app installers, patch updates, etc.
Disaster Recovery	Replicate important data to S3 for recovery during failure.

🔘 How to Use Amazon S3 with Terraform

Terraform is a tool that lets you define cloud resources using simple code (in .tf files), so they can be easily created, updated, or destroyed automatically.

☑ Sample Use Case: Create an S3 Bucket to Store Website Logs

Here's a simple .tf code to create an S3 bucket:

```
resource "aws_s3_bucket" "logs_bucket" {
  bucket = "my-application-logs-bucket"
  acl    = "private"
tags = {
    Environment = "Production"
    Purpose     = "App Logs"
  }
}
```

You can use Terraform to:

- Create/delete S3 buckets
- Enable versioning

- Enable encryption
- Set lifecycle rules (e.g., auto-delete old files)
- Set policies (who can access the bucket)

🛠️ Terraform Use Case Examples:

Use Case	Terraform Feature
Backup files daily	Automate bucket creation with lifecycle rules
Store logs from EC2 instances	Create bucket and link EC2 log exports
Host a static website	Enable website block in S3 bucket
Enable encryption	Add server_side_encryption_configuration block

🟢 How to Use Amazon S3 with Ansible

Ansible is an automation tool used for configuration management — like setting up servers or deploying apps.

You can use **Ansible + S3** to:

- Upload backup files
- Download files to EC2 servers
- Sync content (e.g., website files)
- Configure S3 bucket policies

☑️ Sample Ansible Playbook to Upload a File to S3:

```
- name: Upload file to S3
  hosts: localhost
  tasks:
    - name: Upload to S3 bucket
      amazon.aws.aws_s3:
        bucket: my-app-bucket
        object: logs/server.log
        src: /var/log/server.log
        mode: put
        region: ap-south-1
```

🛠️ Ansible Use Case Examples:

Use Case	Ansible Role
Upload nightly database backups	Use cron + aws_s3 module
Sync static site files to S3	Use aws_s3 module with mode sync
Download software binaries for EC2 setup	Use aws_s3 module in get mode
Manage IAM access for S3	Use iam_policy and bucket_policy modules

🧠 Summary for Non-Tech:

Tool	Role	What It Does with S3
Amazon S3	Cloud Storage	Store any kind of file (photos, backups, websites) securely.
Terraform	Infrastructure Setup	Creates S3 buckets automatically with rules and permissions.
Ansible	Automation	Uploads/downloads files to/from S3 as part of deployment tasks.

🚀 Real-World Scenario

Goal: A company wants to store logs from all web servers in one place, clean them every 30 days, and

back them up to another bucket in case of emergency.

Workflow:

1. Use **Terraform** to:
    - Create two S3 buckets: prod-logs, backup-logs
    - Apply lifecycle policy to delete logs after 30 days
    - Enable versioning and encryption
2. Use **Ansible** to:
    - Upload /var/log/nginx/access.log daily to prod-logs
    - Once a week, sync prod-logs to backup-logs

This setup is fully automated and **requires no manual file transfers** — it runs on schedule using infrastructure-as-code and automation playbooks.

◇ What is Amazon S3 (Simple Storage Service)?

Think of it like Google Drive — but for servers, code, logs, backups, and the whole DevOps universe.

- **S3 = Object storage** – it stores **files (objects)** inside **buckets**.
- Each object has:
    - The **data** (file itself),
    - A **unique key** (its name),
    - And **metadata** (info about the file).

🤔 Why Should a DevOps Engineer Care?

Because DevOps = automation + storage + pipelines + logs + backups — and **S3 sits in the middle of it all.**

- **Stores artifacts** from CI/CD pipelines (e.g., WAR files, Docker images).
- **Backup solution** for databases and logs.
- **Static website hosting** (yes, a full website on S3, no server needed).
- **Storing Terraform state files** (very important).
- **Central location** for config files, build scripts, etc.
- **Staging uploads/downloads** during automation.

🧳 Common Use Cases (with Real Examples)

Use Case	Real-World DevOps Example
Backup logs	Send Apache/Nginx logs from EC2 to S3 daily
CI/CD pipelines	Jenkins stores build artifacts (ZIP, JAR) in S3
Terraform backend	Remote state storage to track infra changes
Static website	Host an HTML/CSS site directly from S3
File Sharing	Store and share PDFs, reports via presigned URLs
Versioning	Track and roll back changes in files
CloudFront Origin	Serve files globally via CDN
Lambda triggers	Auto-process files uploaded to S3 (e.g., image resize)
Disaster Recovery	S3 + cross-region replication for resilience
Ansible + Playbooks	Use S3 as the source for config/scripts

🛠 S3 + Terraform Automation (Infra as Code)

"I want to create S3 buckets automatically with security and structure."

Sample Use: Create an S3 bucket with Terraform

```
resource "aws_s3_bucket" "devops_bucket" {
  bucket = "my-devops-bucket-123"
  acl    = "private"
  versioning {
    enabled = true
  }
  tags = {
    Environment = "DevOps"
    Team        = "Infra"
  }
}
```

🪄 What it does:

- Creates the bucket
- Enables versioning
- Adds tags for management

You can also automate **bucket policies**, **encryption**, **replication**, and **lifecycle rules** using Terraform.

🔧 S3 + Ansible Automation (Config Management)

"I want to upload/download files from S3 during a playbook run."

Use Case: Pull config files from S3 and apply them to EC2

```
- name: Download config from S3
  amazon.aws.aws_s3:
    bucket: my-devops-bucket-123
    object: app-config.yml
    dest: /etc/myapp/config.yml
    mode: get
```

Other Ansible tricks with S3:

- Use S3 as the **inventory source**
- Store **playbooks or vars** in S3
- **Push artifacts** from local machine to S3
- Use **lookup plugins** to read files from S3 on the fly

🔐 Security Tips

Task	What to Do
Encrypt data	Use **SSE-S3** or **SSE-KMS**
Limit access	IAM policies + Bucket policies
Public access	**Block it by default** unless hosting website
Audit access	Enable **S3 access logs + CloudTrail**

🎓 For the Non-Tech Mindset – Learn Like This:

1. **Understand what object storage is** (YouTube S3 analogy videos)
2. **Create a bucket manually on AWS Console**
3. **Use AWS CLI to upload/download files**
4. **Write basic Terraform to create/delete bucket**
5. **Automate S3 file copy using Ansible**

6. **Try hosting a static website using S3**
7. **Enable versioning, lifecycle rules, bucket policy**
8. **Integrate it in your pipeline (CI/CD or backup)**

Real DevOps Mastery (Advanced S3 Use)

- Cross-account access using IAM roles
- S3 Event triggers (Lambda or SNS)
- Multipart upload for huge files
- Integrate with CodePipeline (build → store in S3 → deploy)
- Log shipping from CloudWatch to S3

Conclusion – TL;DR

- **S3 = universal, scalable, cheap, reliable storage**
- DevOps folks use it like a Swiss Army knife — pipelines, backups, logs, websites
- Automate creation and use via **Terraform + Ansible**
- Make it secure, organized, and cost-optimized

In the world of clouds, when bits need a home,
S3 is the vault, where all data is known.

Sample Terraform Project: Create and Configure S3 Bucket

Project structure:

```
terraform-s3/
├── main.tf
├── variables.tf
└── terraform.tfvars
```

main.tf

```
provider "aws" {
  region = var.aws_region
}
resource "aws_s3_bucket" "devops_bucket" {
  bucket = var.bucket_name
  acl    = "private"
versioning {
    enabled = true
  }
lifecycle_rule {
    id      = "expire-older-files"
    enabled = true
expiration {
      days = 30
    }
  }
tags = {
    Environment = "Dev"
    Owner       = "DevOps Engineer"
  }
```

```
}
```

variables.tf

```
variable "aws_region" {
  default = "us-east-1"
}
variable "bucket_name" {
  description = "The name of the S3 bucket"
  type        = string
}
```

terraform.tfvars

```
bucket_name = "my-devops-bucket-1234"
```

To Run the Project:

```
terraform init
terraform plan
terraform apply
```

Boom! Bucket created with:

- Versioning enabled
- Expiry after 30 days
- Tagged and private

Ansible Playbook: S3 Sync (Upload or Download)

Uses aws_s3 module from amazon.aws collection

Prerequisites:

- pip install boto3 botocore
- ansible-galaxy collection install amazon.aws
- IAM user with S3 access (set via AWS credentials)

Project Structure:

```
ansible-s3-sync/
├── playbook.yml
└── files/
    └── my-local-file.txt
```

playbook.yml – Uploading to S3

```
- name: Upload local files to S3 bucket
  hosts: localhost
  connection: local
  gather_facts: false
vars:
    bucket_name: my-devops-bucket-1234
    s3_file_key: configs/my-local-file.txt
    local_file_path: ./files/my-local-file.txt
tasks:
    - name: Upload file to S3
      amazon.aws.aws_s3:
        bucket: "{{ bucket_name }}"
```

```
    object: "{{ s3_file_key }}"
    src: "{{ local_file_path }}"
    mode: put
```

Modify for Download (just change mode: get)

```
 - name: Download file from S3
   amazon.aws.aws_s3:
    bucket: "{{ bucket_name }}"
    object: "{{ s3_file_key }}"
    dest: "./files/downloaded-file.txt"
    mode: get
```

How to Run the Playbook

```
ansible-playbook playbook.yml
```

What You've Achieved

Tool	What It Did
Terraform	Created a secure, versioned S3 bucket with tags and lifecycle
Ansible	Synced files to/from that bucket — like a DevOps automation wizard

S3 Bucket Policy – Restrict Uploads/Downloads

Restrict to a Specific IAM User (upload only, no download)

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowUserToUploadOnly",
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::123456789012:user/s3-upload-user"
   },
   "Action": "s3:PutObject",
   "Resource": "arn:aws:s3:::my-devops-bucket-1234/*"
  }
 ]
}
```

This denies read/download (s3:GetObject) – only PutObject (upload) is allowed

How to apply it in Terraform:

```
resource "aws_s3_bucket_policy" "restrict_uploads" {
 bucket = aws_s3_bucket.devops_bucket.id
policy = jsonencode({
   Version = "2012-10-17",
   Statement = [
    {
     Sid    = "AllowUserToUploadOnly",
```

```
      Effect = "Allow",
      Principal = {
        AWS = "arn:aws:iam::123456789012:user/s3-upload-user"
      },
      Action = "s3:PutObject",
      Resource = "${aws_s3_bucket.devops_bucket.arn}/*"
    }
  ]
 })
}
```

Cost-Saving Lifecycle Rules

S3 can silently clean up your mess while you sip chai .

Rule: Delete non-current versions after 30 days + expire files after 60

```
lifecycle_rule {
 id      = "clean-old-stuff"
 enabled = true
noncurrent_version_expiration {
   days = 30
 }
expiration {
   days = 60
 }
prefix = "" # applies to all objects
}
```

Benefits:

- Auto-cleans stale files
- Controls versioning cost
- You don't pay for old junk you forgot

Terraform + Ansible Combined Pipeline

Goal:

1. Terraform creates bucket
2. Ansible uploads a file to it

Step-by-Step:

1. Create a shell script: deploy.sh

```
#!/bin/bash
echo " Running Terraform..."
cd terraform-s3
terraform init
terraform apply -auto-approve
cd ..
echo " Running Ansible..."
ansible-playbook ansible-s3-sync/playbook.yml
```

Directory Structure:

```
project-root/
```

```
├── terraform-s3/
│   ├── main.tf
│   ├── variables.tf
│   └── terraform.tfvars
├── ansible-s3-sync/
│   ├── playbook.yml
│   └── files/
│       └── my-local-file.txt
└── deploy.sh
```

⚙️ 2. Set AWS credentials (CLI or env vars):

```
export AWS_ACCESS_KEY_ID=XXXX
export AWS_SECRET_ACCESS_KEY=YYYY
```

▶️ 3. Run the Pipeline:

```
chmod +x deploy.sh
./deploy.sh
```

🔚 TL;DR: The Final Power Stack 🦾

Tool	What It Does
Terraform	Creates S3 bucket, lifecycle, bucket policy
Ansible	Uploads config/files to that bucket
Shell	Glues it all into a deployable pipeline