

Project: 5 Title: Automated Jenkins Job Triggered by Access Log Size

Objective:

The aim of this project is to create an **automated log monitoring and upload system**. The system continuously checks if the access log file (/var/log/custom/access.log) exceeds **1 GB**. When it does:

1. A Jenkins pipeline is automatically triggered.
2. The log file is **uploaded to an Amazon S3 bucket**.
3. After successful upload, the **original log file is cleared**.
4. An email notification is sent.

This setup helps in **automating log management** and prevents excessive disk usage on the server.

Steps:

STEP 1: Initial Setup

i) Update Your Ubuntu System:

```
sudo apt update && sudo apt upgrade -y
```

ii) Install Java (Required for Jenkins):

```
sudo apt install openjdk-17-jdk -y
```

iii) Install Jenkins:

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```

```
sudo apt update
sudo apt install jenkins -y
```

iv) Start Jenkins:

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

v) Access Jenkins UI:

Go to:

<http://localhost:8080>

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

United States (N. Virginia)

AtharvaCloud

Amazon S3 > Buckets > Create bucket

Info Help Feedback

General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type

Info

☒ General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

☐ Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name

Info

logfileexceeds2004

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn More](#)

Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

Object Ownership

Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☒ ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

☐ ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership

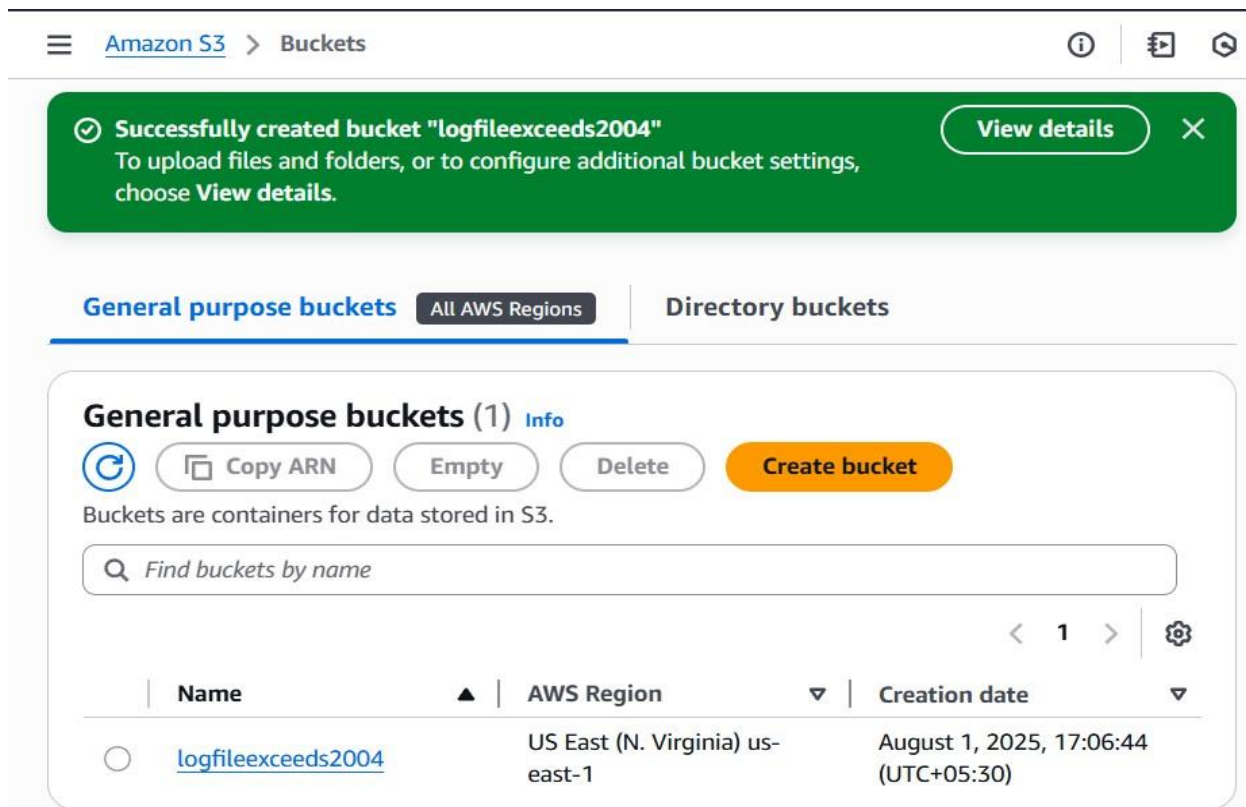
Bucket owner enforced

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**



STEP 4: Create Shell Script to Monitor Log Size:

nano ~/monitor_log.sh

ii)write the script

```
#!/bin/bash
FILE="/var/log/custom/access.log"
MAX_SIZE=$((1024 * 1024 * 1024)) # 1 GB

# Jenkins details
JENKINS_URL="http://localhost:8080"
JENKINS_USER="Atharva"
JENKINS_API_TOKEN="1270...8f32222222"
JOB_NAME="UploadLogToS3"

if [ -f "$FILE" ]; then
    FILE_SIZE=$(stat -c%s "$FILE")

    if [ "$FILE_SIZE" -ge "$MAX_SIZE" ]; then
        echo "✅ File exceeded 1GB. Triggering Jenkins job..."

        # Get Jenkins crumb for CSRF protection
        CRUMB=$(curl -s -u "$JENKINS_USER:$JENKINS_API_TOKEN" \
            "$JENKINS_URL/crumIssuer/api/json" | jq -r '.crumbRequestField + ":" + .crumb')

        # Trigger Jenkins job with file path as parameter
        curl -X POST "$JENKINS_URL/job/$JOB_NAME/buildWithParameters" \
            --user "$JENKINS_USER:$JENKINS_API_TOKEN" \
            -H "$CRUMB" \
            --data-urlencode "LOG_PATH=$FILE"
    else
        echo "📄 File size is below 1GB. No action taken."
    fi
else
    echo "❌ File not found: $FILE"
fi
```

Explanation of Script:

Section 1: File Path and Size Limit Definition

In this section, the path to the log file that needs to be monitored is defined. Additionally, a size threshold is set to 1 GB (in bytes). This value is used to compare against the actual file size later.

Section 2: Jenkins Configuration

This section contains all the necessary Jenkins connection details:

- Jenkins server URL (usually localhost or a remote Jenkins server)
- Username to authenticate with Jenkins
- Jenkins API token for secure access
- Name of the Jenkins job that will be triggered if conditions are met

Section 3: Check if File Exists and Get File Size

Here, the script checks whether the specified log file actually exists on the system. If it does, the script retrieves the file's current size in bytes using system utilities.

Section 4: Check If File Size Exceeds Threshold

This conditional section compares the actual file size with the defined 1 GB limit. If the file size is greater than or equal to 1 GB, it proceeds to trigger a Jenkins job.

Section 6: Get CSRF Protection Crumb from Jenkins

Jenkins uses CSRF protection to prevent malicious requests. To authenticate POST requests, Jenkins issues a special token called a **crumb**. In this section, the script fetches this crumb using Jenkins's API and prepares it for use in the next step.

Section 7: Trigger Jenkins Job with Parameters

Once the crumb is obtained, the script triggers the specified Jenkins job via a POST request. It passes the log file's path as a parameter to the Jenkins job and includes the crumb and authentication token in the request.

Section 8: Log File Size Below Threshold

If the file exists but is **smaller than 1 GB**, the script prints an informational message and exits without triggering the Jenkins job. No further action is taken.

Section 9: File Not Found Handler: If the specified log file does **not exist** at all, the script logs an error message indicating that the file could not be found and terminates without performing any operation.

iii) Make it executable:

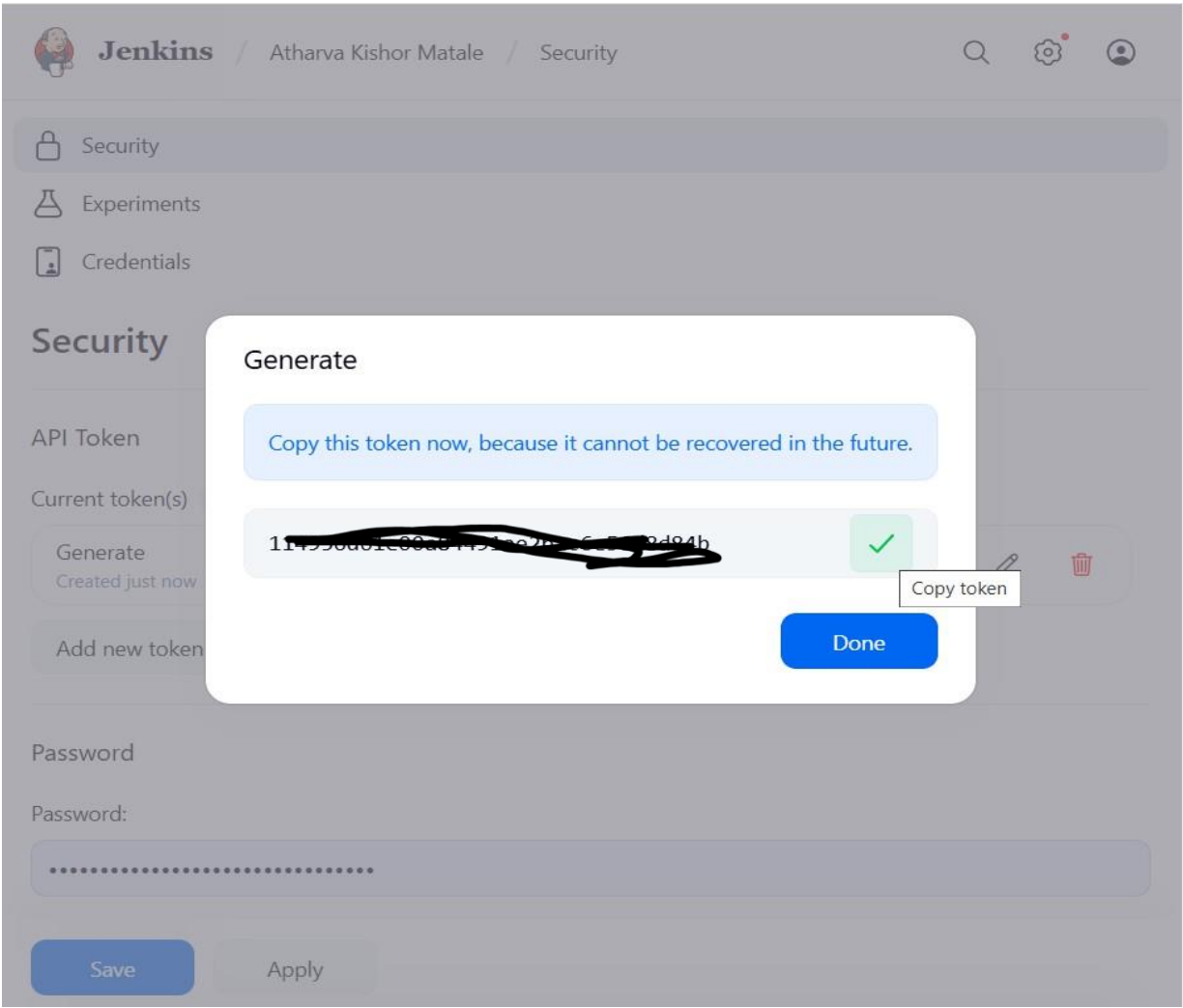
```
chmod +x ~/monitor_log.sh
```

STEP 5: Created Jenkins Api Token

go to security ---- click on jenkins api ---give name--- create

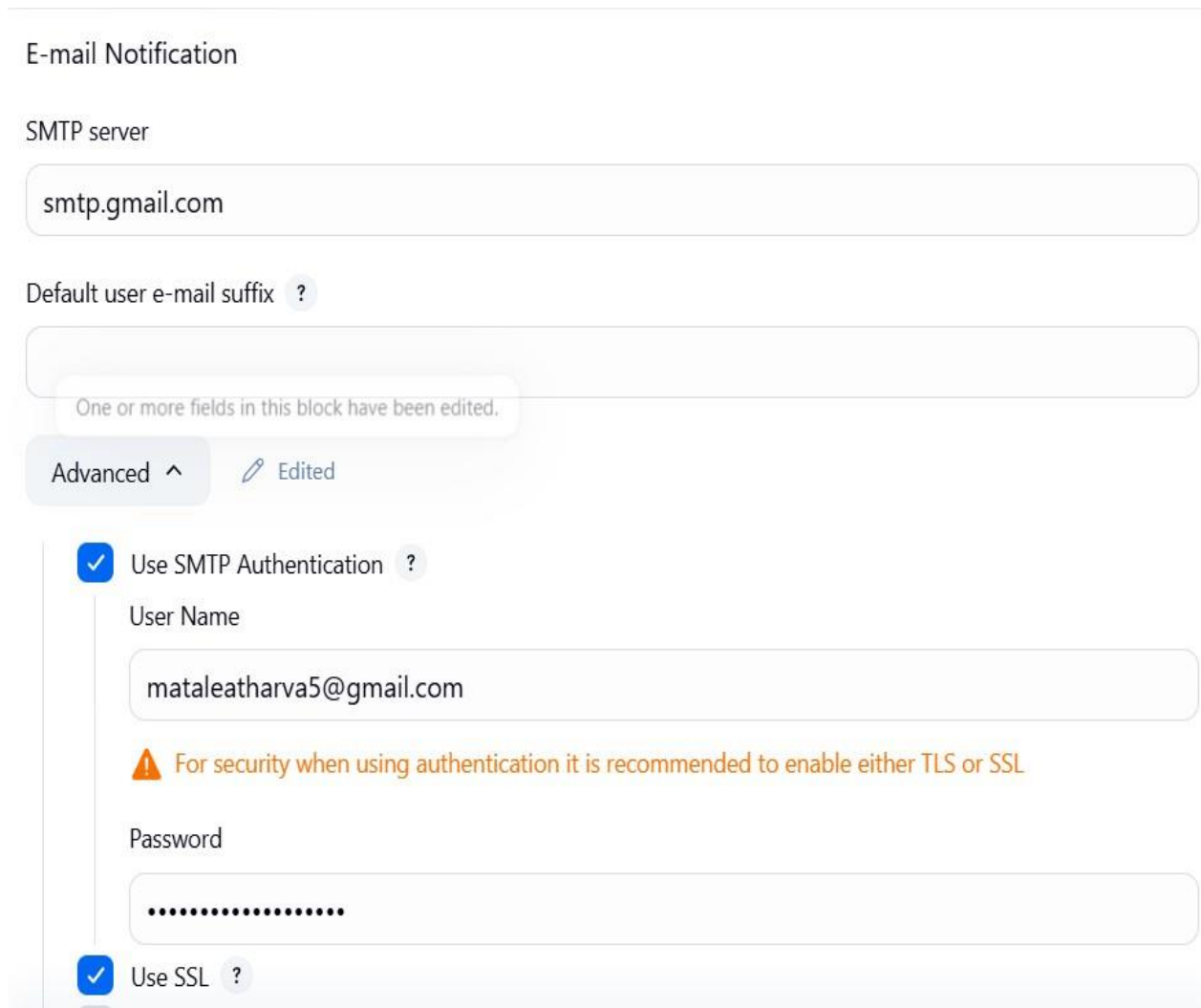
Why: created the Jenkins API token to securely authenticate your script with Jenkins without using your

actual password. It allows the script to trigger Jenkins jobs and fetch CSRF protection crumbs via the REST API.



STEP 6: Setting up email notification:

To set up email notifications in Jenkins, go to Manage Jenkins → Configure System, and configure the E-mail Notification section with your SMTP server details. Then save and test the configuration.



The screenshot shows the 'E-mail Notification' configuration section in Jenkins. It includes a text field for the SMTP server set to 'smtp.gmail.com'. Below it is a section for 'Default user e-mail suffix' which is currently empty. A notification bubble states 'One or more fields in this block have been edited.' There is an 'Advanced' toggle and an 'Edited' indicator. Under the 'Advanced' section, 'Use SMTP Authentication' is checked, with a 'User Name' field containing 'mataleatharva5@gmail.com'. A warning message says 'For security when using authentication it is recommended to enable either TLS or SSL'. The 'Password' field is masked with dots. At the bottom, 'Use SSL' is also checked.

E-mail Notification

SMTP server

smtp.gmail.com

Default user e-mail suffix ?

One or more fields in this block have been edited.

Advanced ^ Edited

☒ Use SMTP Authentication ?

User Name

mataleatharva5@gmail.com

⚠ For security when using authentication it is recommended to enable either TLS or SSL

Password

.....

☒ Use SSL ?

STEP 7: Create New Job

- Go to Jenkins → New Item → "UploadLogToS3"
- Choose **Pipeline**
- Add a string parameter: LOG_PATH

Why:

create a string parameter LOG_PATH in the Jenkins job so the script can pass the log file path to the pipeline during the build trigger. This allows the pipeline to know which specific log file to process or upload.

☒ This project is parameterized ?

String Parameter ?

Name ?

LOG_PATH

Default Value ?

/var/log/custom/access.log

Description ?

Full path of the log file to upload

ii) Added security credentials in Jenkins:



Jenkins



System









Global credentials (unrestricted)



Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 379e872e-6be4-4edb-bb7a-4a0608b7c8b4	mataleatharva5@gmail.com/*****	Username with password	
 91b80839-80a2-4498-9c45-42c71cdcb99a	mataleatharva5@gmail.com/***** (gmail-app)	Username with password	gmail-app 
 aws-access-key	AWS Access Key for S3 upload	Secret text	AWS Access Key for S3 upload 
 aws-secret-key	AWS Secret Key for S3 upload	Secret text	AWS Secret Key for S3 upload 

Icon: S M L

iii)written a pipeline:

i. Agent and Parameters

agent any: Run on any available Jenkins agent.

parameters { LOG_PATH }: Accepts the log file path from the triggering script to make it dynamic.

ii. Environment Variables

Stores AWS credentials, S3 bucket name, region, and email recipient in environment variables for reuse and cleaner code.

iii. Stages

a. Verify Log File

- Checks if the log file at the given path exists.
- Fails the pipeline early if the file is missing.

b. Upload to S3

- Uploads the verified log file to a specific S3 bucket using AWS CLI.
- Captures the status to catch upload failures.

c. Clear Log File

- Empties the log file after upload to prevent duplicate uploads or storage bloat.

iv. Post Actions

On Success

- Sends an email confirming successful upload and cleanup, with job details.

On Failure

- Sends an email alerting that the upload failed, with a link to the Jenkins job for troubleshooting.

STEP 8: Edit Crontab

```
atharva@LAPTOP-A9SSNJEV:~$ crontab -e

crontab: installing new crontab
atharva@LAPTOP-A9SSNJEV:~$ |

GNU nano 7.2 /tmp/crontab.RTeBQh/crontab *
*/5 * * * * /home/atharva/monitor_log.sh >> /home/atharva/monitor_log_output.log 2>&1
# Edit this file to introduce tasks to be run by cron.
```

STEP 9: Testing

i)manually increased the size of log file for testing:

Using:

```
truncate -s 1.1G /var/log/custom/access.log
```


ii)Run the monitor_log.sh script:


```
atharva@LAPTOP-A9SSNJEV:~$ ./monitor_log.sh
✅ File exceeded 1GB. Triggering Jenkins job...
atharva@LAPTOP-A9SSNJEV:~$
```

iii) Automatically builds the pipeline:

← → ↺ 🏠 ⓘ localhost:8080/job/UploadLogToS3/6/console ☆ ⬇️ A ⋮

🗄️ | 📦 All Bookmarks

 **Jenkins** / UploadLogToS3 / #6 🔍 ⚙️ 👤

|| Pause/resume




↶ Replay

☰ Pipeline Steps

📁 Workspaces

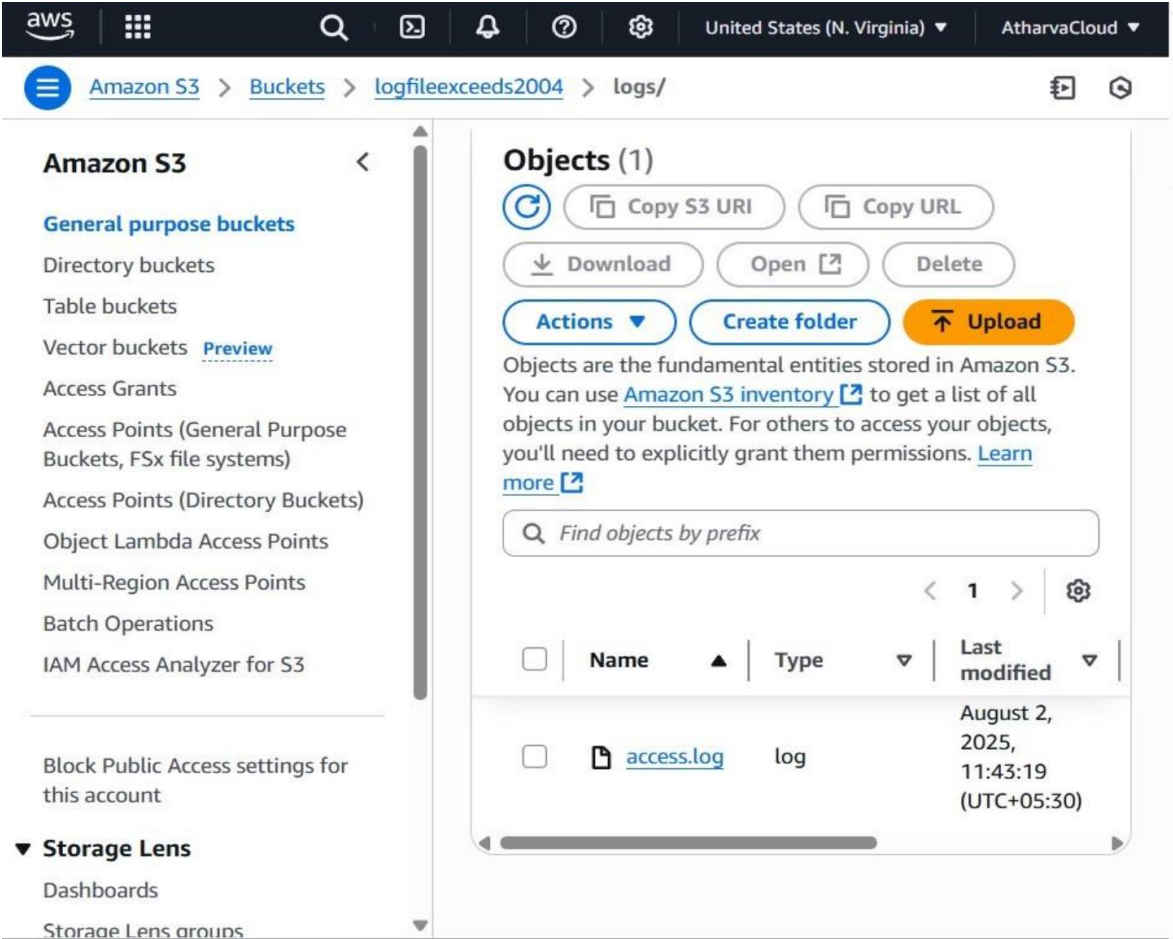
← Previous Build

✅ **Console Output**

Progress:  ❌  Download  Copy View as plain text

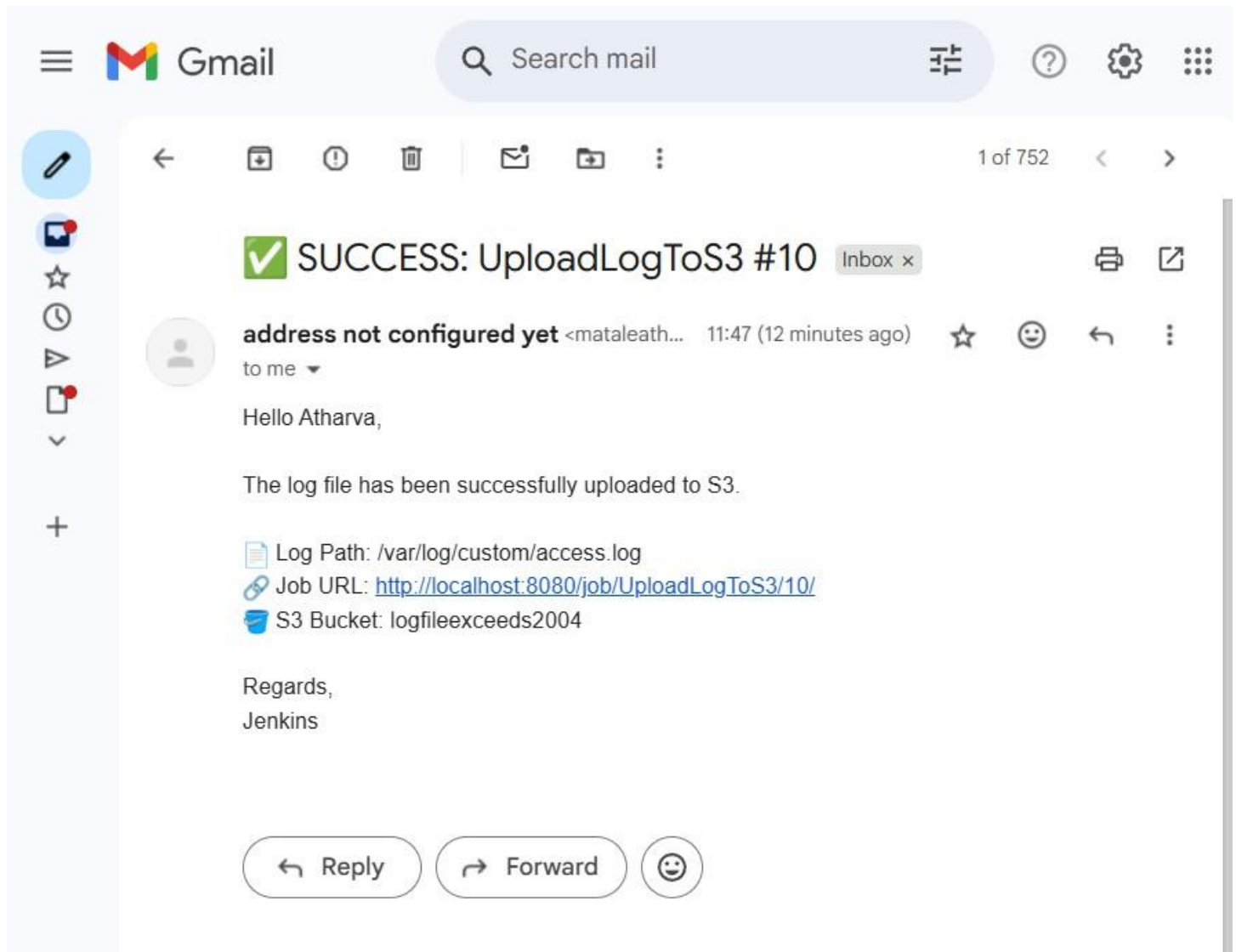
```
Started by user Atharva Kishor Matala
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/UploadLogToS3
[Pipeline] {
[Pipeline] withCredentials
Masking supported pattern matches of $AWS_ACCESS_KEY_ID or $AWS_SECRET_ACCESS_KEY
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
```

iv)Log File uploaded to s3:



```
[Pipeline] sh
+ aws s3 cp /var/log/custom/access.log s3://logfileexceeds2004/logs/ --region us-east-1
--cli-connect-timeout 120 --cli-read-timeout 120 --no-progress
upload: ../../../../log/custom/access.log to s3://logfileexceeds2004/logs/access.log
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
[Pipeline] echo
[Pipeline] mail
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

v)Email notification Received:



SUMMARY:

This project automates the monitoring and uploading of large log files using a Jenkins pipeline. A Bash script checks if the log file exceeds 1 GB and, if so, triggers a Jenkins job with the file path as a parameter. The Jenkins pipeline verifies the file, uploads it to an S3 bucket using AWS CLI, and clears the log afterward. Email notifications are sent on success or failure to keep the user informed. AWS credentials are securely managed, and CSRF protection is handled via Jenkins crumbs. This setup ensures efficient log management and automated cloud storage without manual intervention.

tested email notification

added secret and access key

