SECURE VM WITH AZURITE BLOB STORAGE BACKUP

A step-by-step journey of securing a Linux server and automating backups

Summary

This project shows how I built a secure Linux server on VirtualBox and set up automated backups to Azurite (an Azure Storage emulator). The process included hardening the VM with a firewall and Fail2Ban, configuring SSH for safe access, and using scripts with azcopy to send backups into blob storage. Microsoft Storage Explorer was used to confirm everything worked. Finally, I automated the process with cron and added a dedicated backup user for better security.

Tebogo Matseding

tmatseding@outlook.com

Table of Contents

Test Fail2ban

Logging using PuTTY

Create Script

Script permissions

Script Complete

Verify Creation

Download Microsoft Azure Storage Explorer

Download Nodejs

Install Azurite

Download azcopy

Unzip and move azcopy

Create A Firewall Rule

Create Test Container

Get SAS Token

Verify IP Address

Test script for backup

Script worked

Verify Upload

Create Container

Official Backup Script

Backup is successful

Verify Backup

Create Backupuser

Move script and hand over ownership

Update Script

Verify Backup User can use script

Verify Backup

Crontab and schedule backup

Test Crontab

Crontab worked

Done

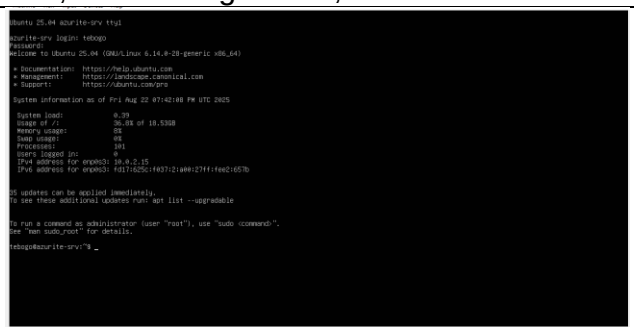
START

Introduction

This guide walks you through creating a secure Linux virtual machine, setting up automated backups using Azurite (an Azure Storage emulator), and ensuring everything is protected with firewall rules, SSH configuration, and Fail2Ban.

First Log on

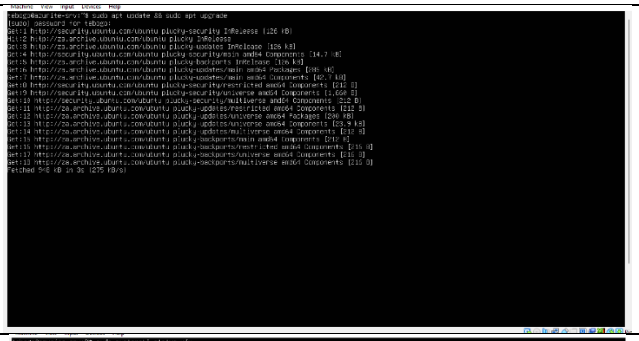
After starting your VM in VirtualBox, log in using the default credentials. You will see a terminal prompt where all commands will be entered.



Update

(using sudo apt update && sudo apt upgrade)

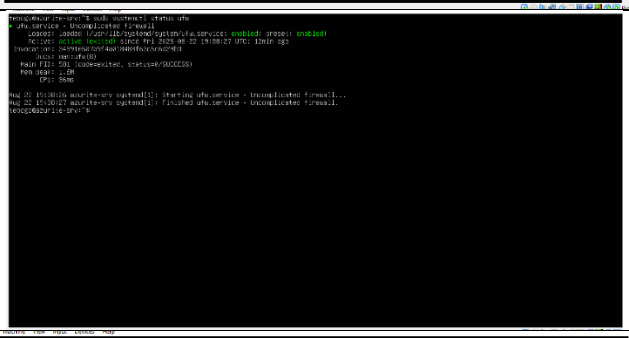
Make sure the system is up-to-date with the latest software and security patches



UFW status

(using sudo systemctl status ufw)

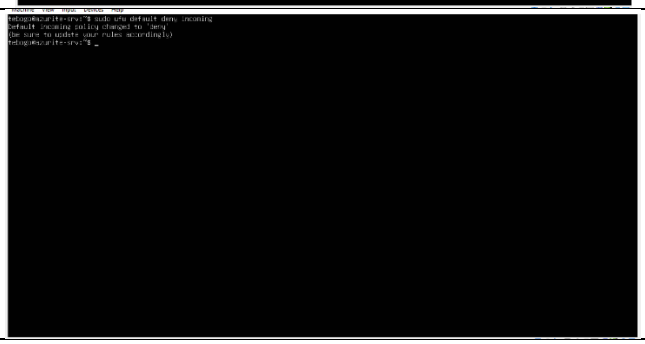
Check the status of the firewall (UFW – Uncomplicated Firewall) If inactive, we'll enable it later after setting rules.

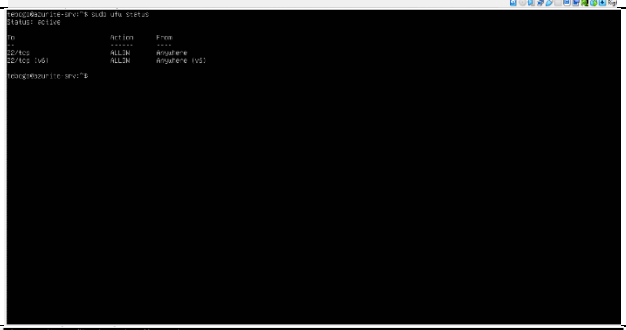
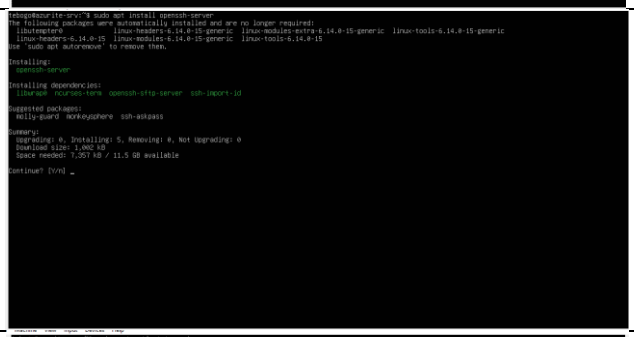
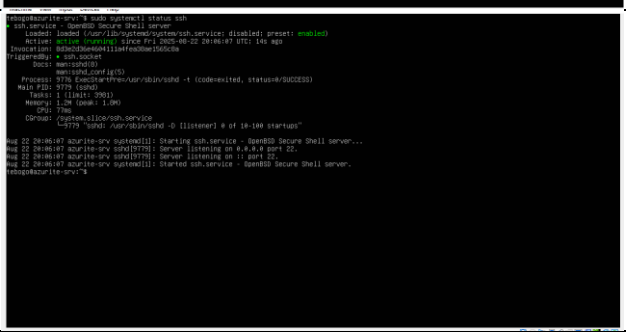


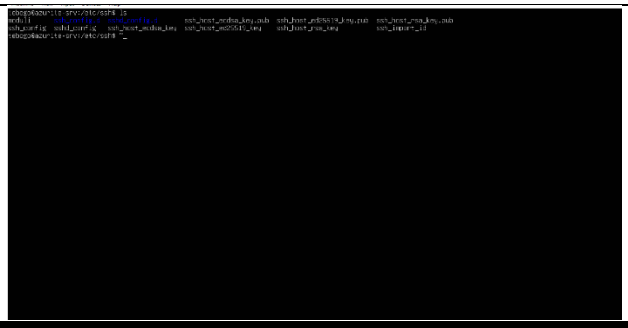
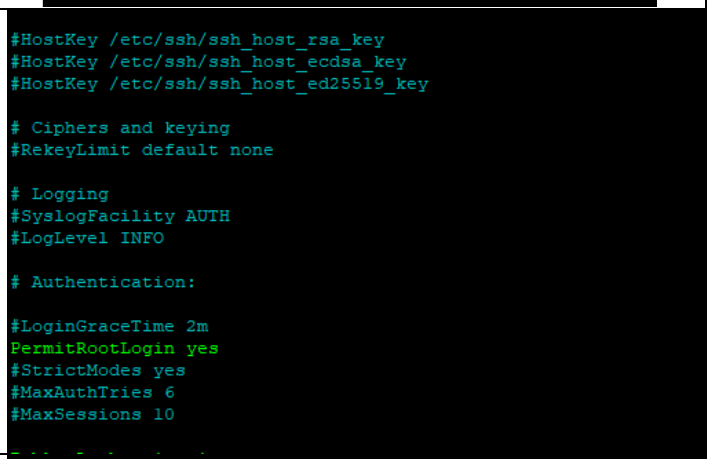
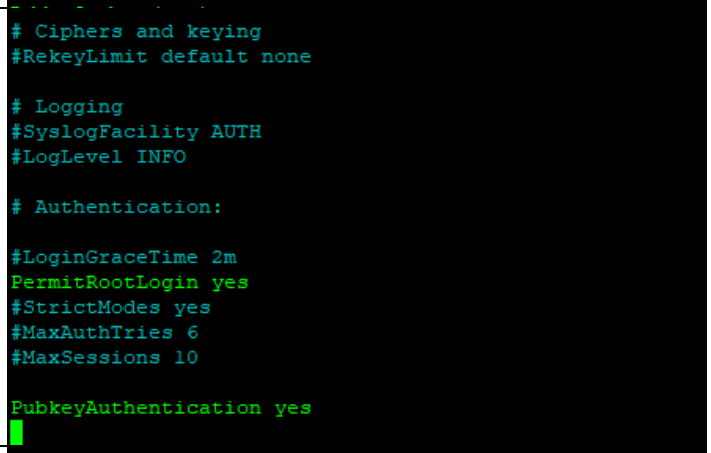
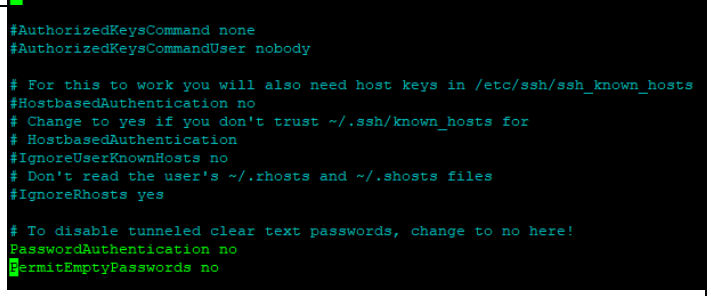
Close all ports

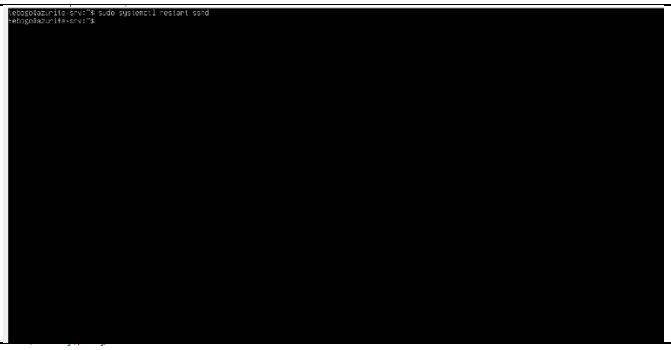
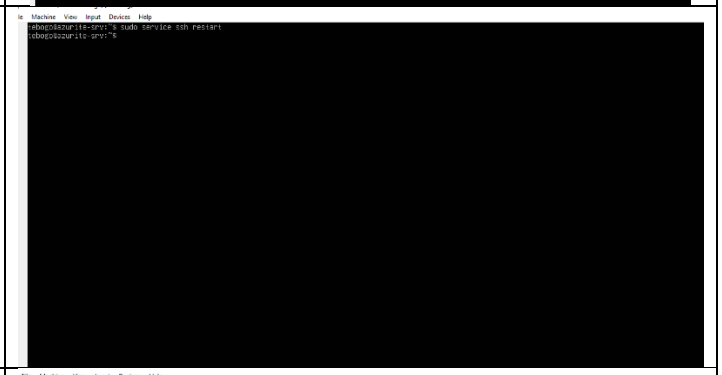
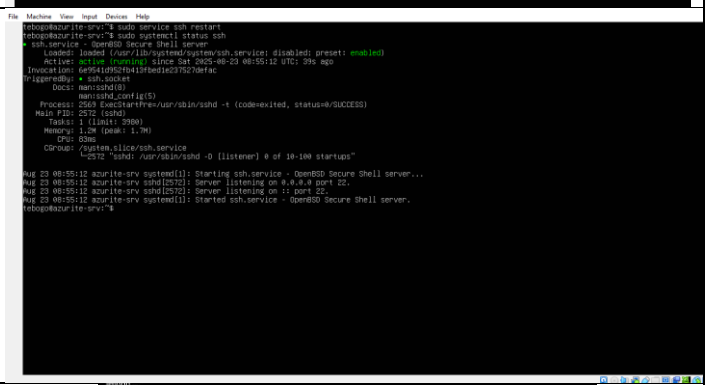
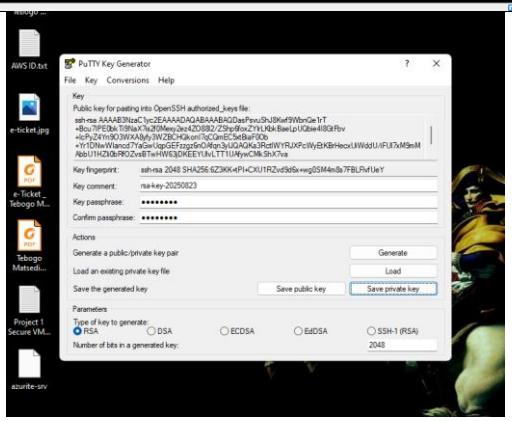
(using sudo ufw default deny incoming)

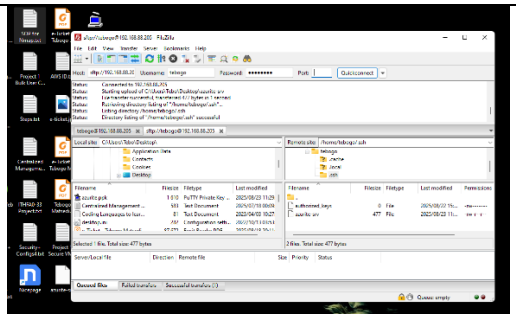
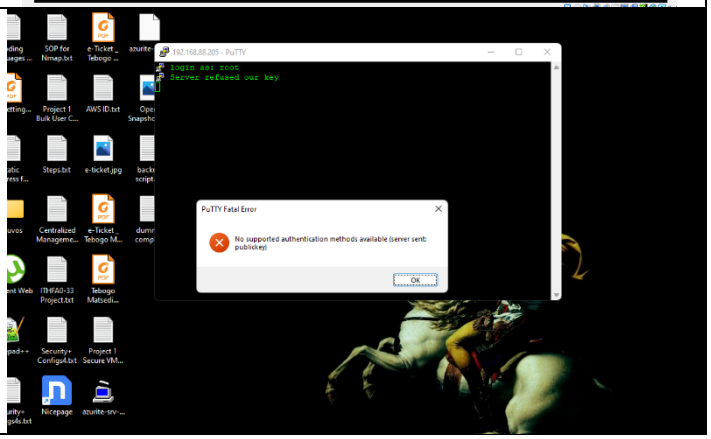
Start by blocking all incoming connections to make the server secure

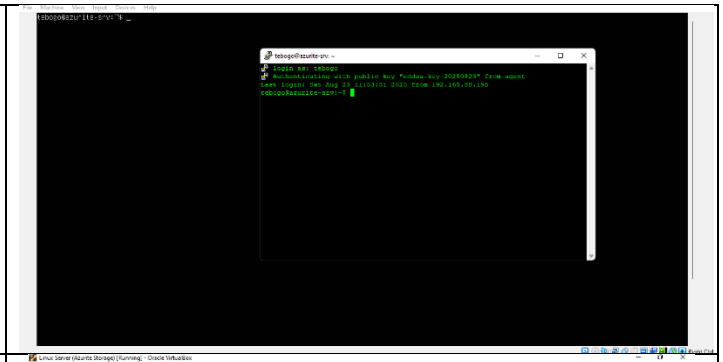
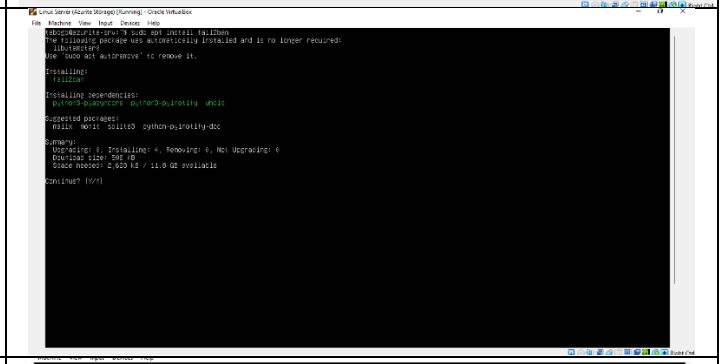
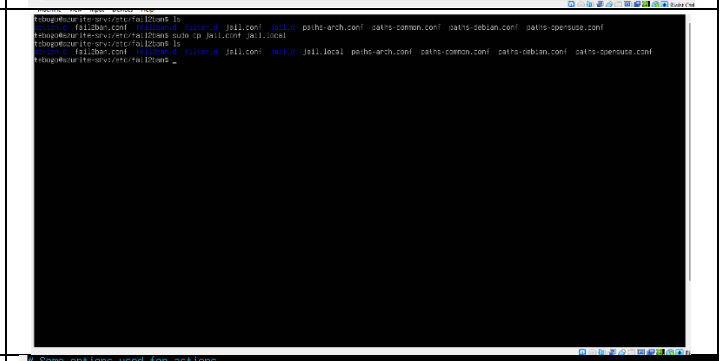


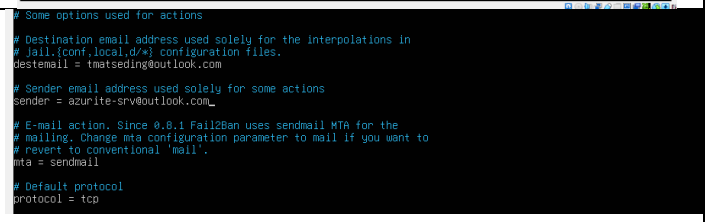
Allow SSH traffic (using sudo ufw allow ssh) To maintain remote access via SSH, open port 22	
Enable UFW (using sudo ufw enable) Turn on the firewall with the configured rules	
Verify rules were applied (using sudo ufw status) Check that SSH is allowed and other ports are blocked	
Install OpenSSH-Server (using sudo apt install openssh-server) Install the SSH service to allow secure remote logins	
OpenSSH-Server Status (using sudo systemctl status ssh) Ensure the SSH server is running	

Config rules (using sudo nano /etc/ssh/sshd_config) Edit the SSH configuration file to harden access	
Permit Root Login (yes) For security, root login should be disabled	
Public Key Authentication (yes) Enable key-based login for secure access	
Password Authenitication & Permit Empty Password (no & no) Disable password logins and empty passwords	

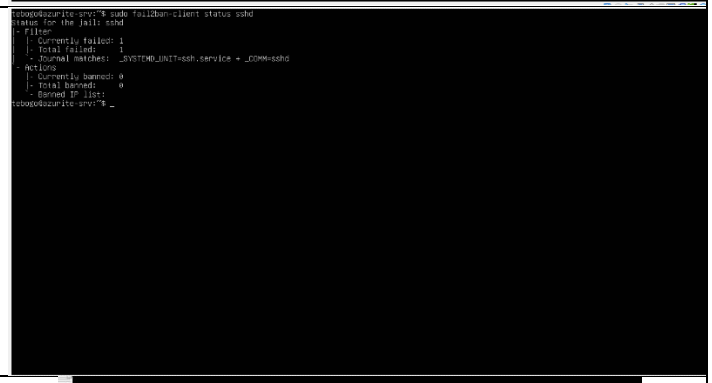
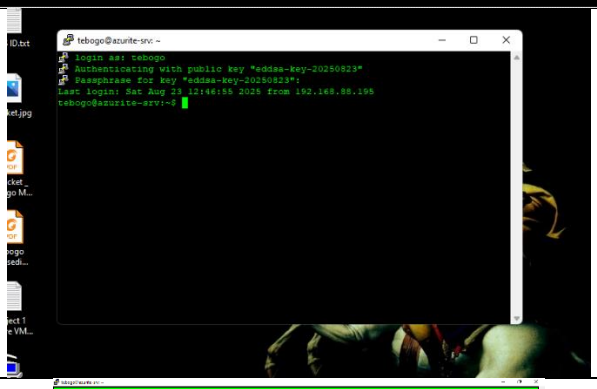
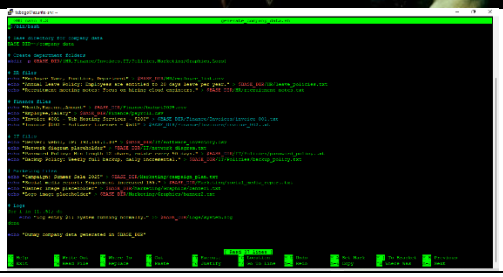
Restart sshd (using sudo systemctl restart sshd) Apply configuration changes by restarting SSH	
Restart ssh (using sudo service ssh restart) Optionally, restart the SSH service fully	
Verify changes didnt break anything	
Generate Keys (using PuTTY) Use PuTTYgen on your Windows machine to generate an SSH key pair. Save the private key (.ppk) on your local computer. Save the public key (.pub) file, which will be uploaded to the server.	

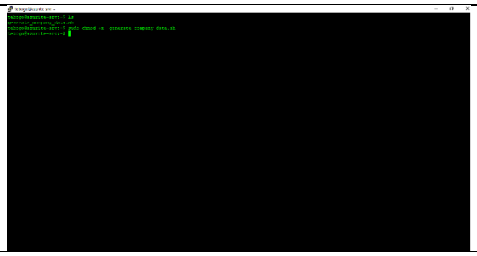
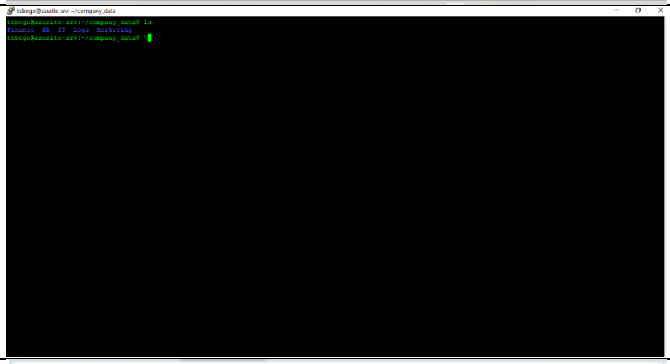
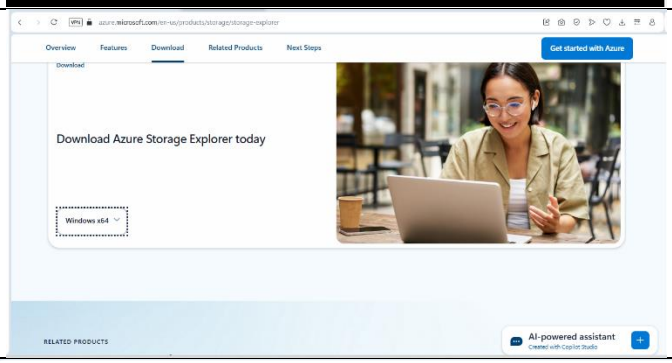
# Transfer Public Key (using sftp on filezilla) Open FileZilla and connect to your VM using the temporary username/password. Navigate to the .ssh folder in your user's home directory on the VM. Upload the public key file	
# Permissions (chmod 600 & 700) Ensure .ssh folder and files have correct permissions	
# Append Contents (of my public to authorized_keys) Append the public key to authorized_keys	
# Try logging with root Confirm root login is blocked; only your user can log in.	

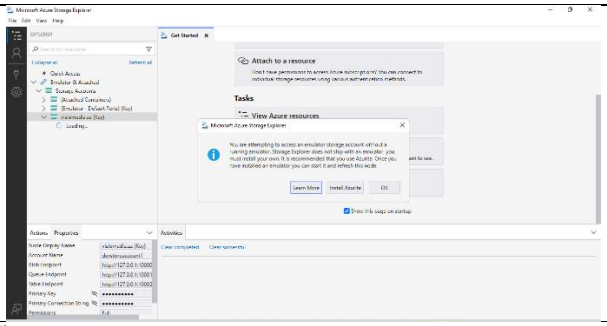
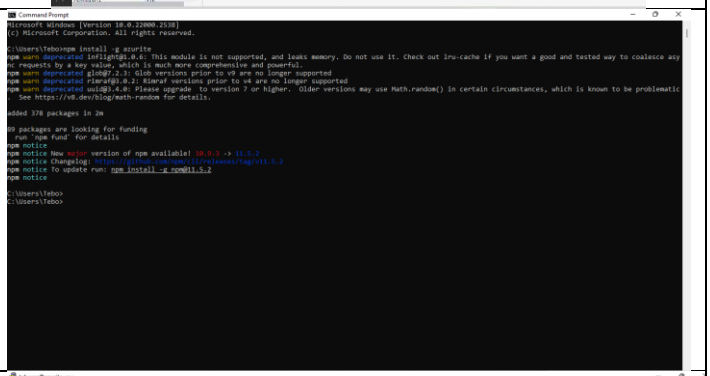
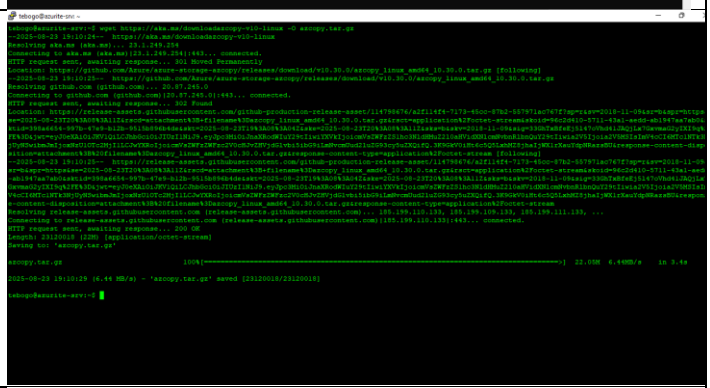
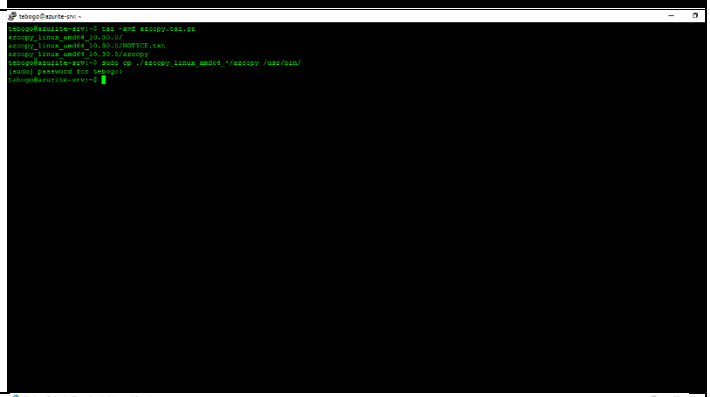
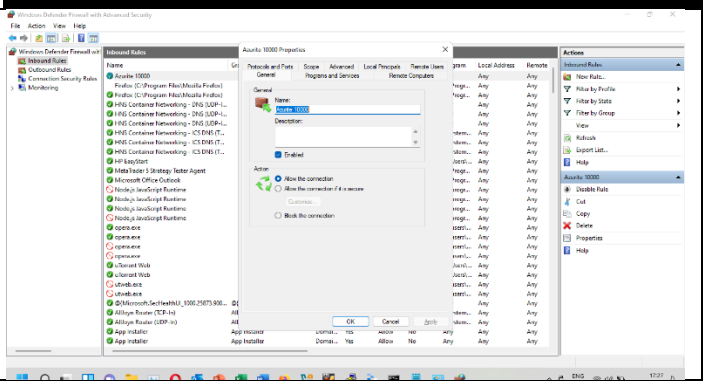
# Log In as user (using my private key) Test SSH login with the normal user account.	
# Install Fail2ban (using sudo apt install fail2ban) Install Fail2Ban to block repeated login attempts	
# Making copy (of the jail.conf to jail.local) Backup the default Fail2Ban configuration	

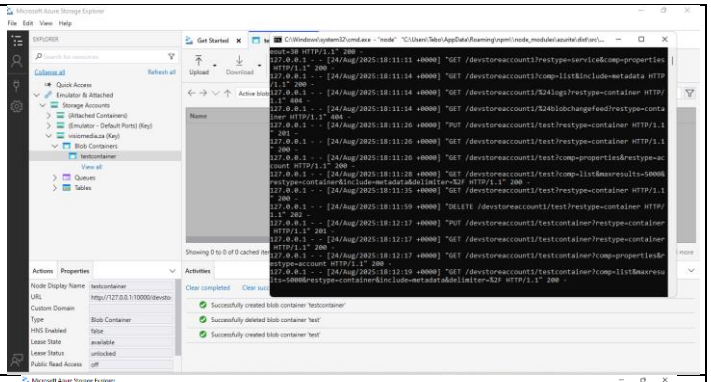
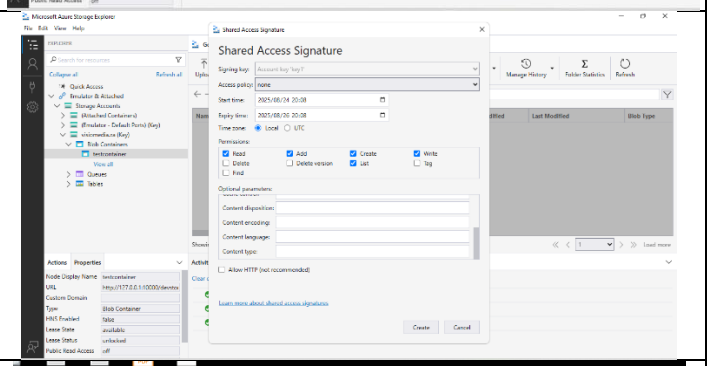
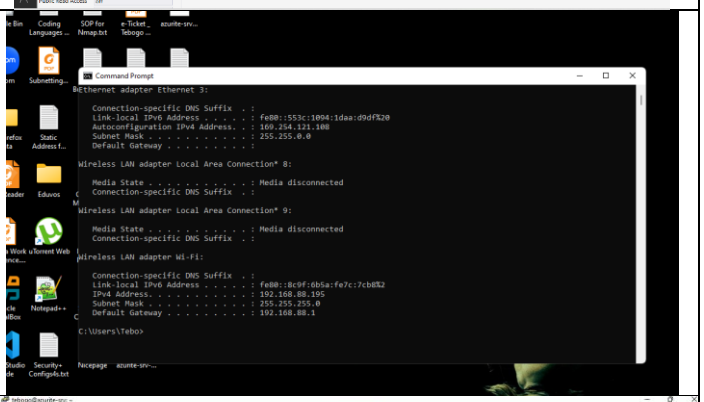
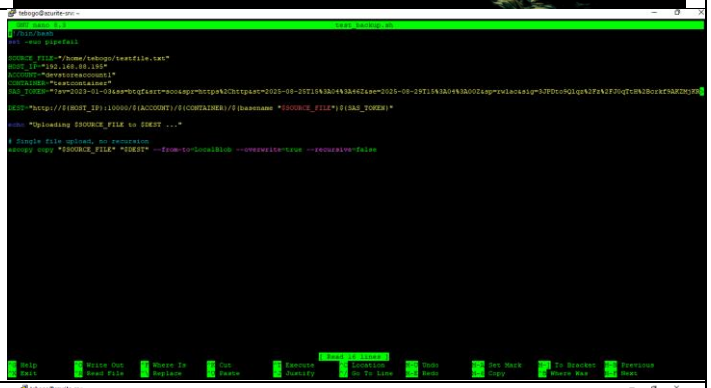
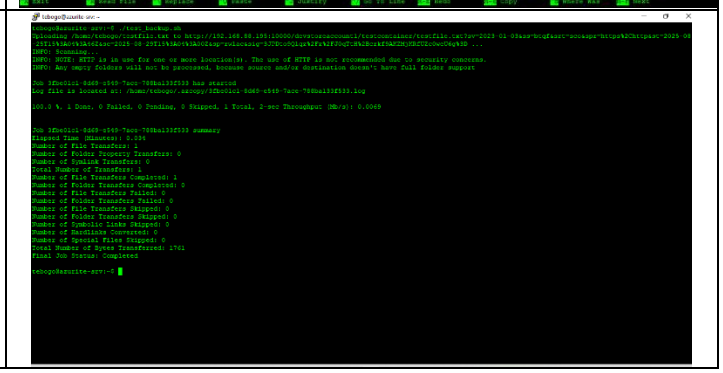
# Email Setting 	
(receive alerts if someone is trying to access the server)
Set email notifications in jail.local to get alerts for blocked attempts.
	
	``
	# Some options used for actions
	
	# Destination email address used solely for the interpolations in
	# jail.{conf,local,d/*} configuration files.
	destemail = tmatseding@outlook.com
	
	# Sender email address used solely for some actions
	sender = azurite-srv@outlook.com
	
	# E-mail action. Since 0.8.1 Fail2Ban uses sendmail MTA for the
	# mailing. Change mta configuration parameter to mail if you want to
	# revert to conventional 'mail'.
	mta = sendmail
	
	# Default protocol
	protocol = tcp
	``

# SSHD rules Enable SSH protection in Fail2Ban config.	``` [sshd] enabled = true maxretry = 3 bantime = 1w findtime = 5m logpath = /var/log/auth.log # To use more aggressive sshd modes set filter parameter "mode" in jail.local: # normal (default), ddos, extra or aggressive (combines all). # See "tests/files/logs/sshd" or "filter.d/sshd.conf" for usage example and details. #mode = normal port = ssh logpath = %(sshd_log)s backend = %(sshd_backend)s [dropbear] port = ssh logpath = %(dropbear_log)s ```

# Recidive Enable recidive jail to block repeated offenders for longer periods.	
# Fail2ban Jails Check active jails to confirm SSH protection	
# Test Fail2ban (by attempting to log in with another computer)	
# Logging using PuTTY Monitor SSH login attempts using PuTTY or local logs	
# Create Script (that generates a dummy company information) [dummy company script]()	

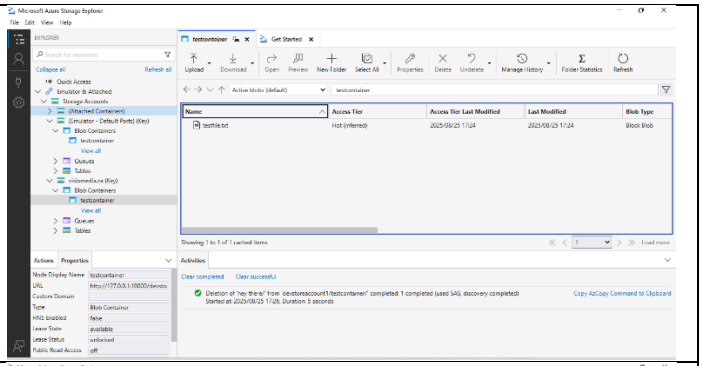
Script permissions	
Script Complete	
Verify Creation	
Download Microsoft Azure Storage Explorer	
Download Nodejs	

Install Azurite	
	
Download azcopy	
Unzip and move azcopy Extract and move it to a folder in your PATH for easy use	
Create A Firewall Rule Ensure Azurite can be accessed safely through firewall if needed.	

Create Test Container Create a test storage container in Azurite for initial uploads.	
Get SAS Token Generate a SAS token to authenticate uploads.	
Verify IP Address	
Test script for backup Run the backup script and check it completes without errors	
Script worked	

Verify Upload

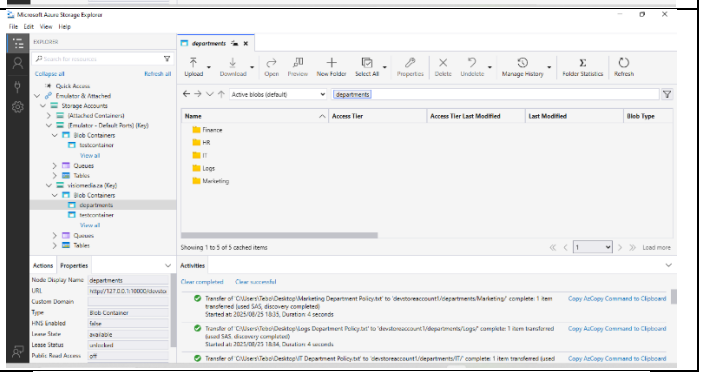
Use Storage Explorer to check that files appear in the container



Create Container

(with corresponding folders)

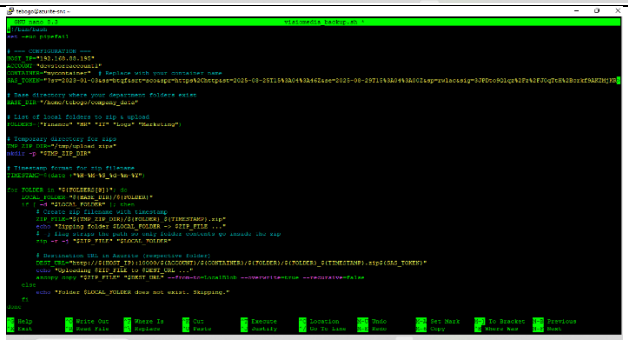
Create a dedicated container for official backups



Official Backup Script

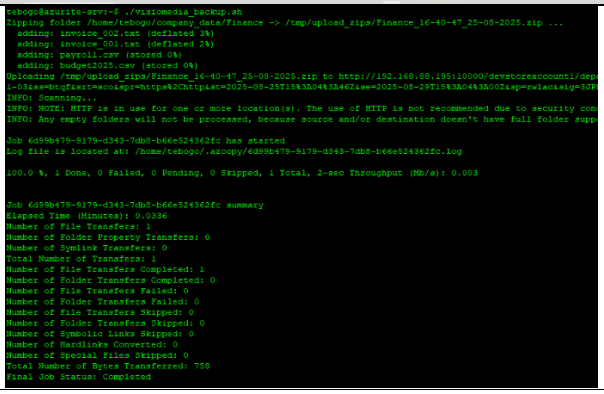
Update the script to use the official container and SAS token

[backup script]

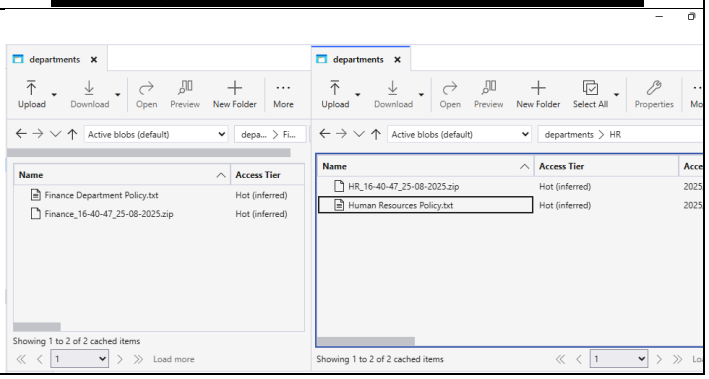


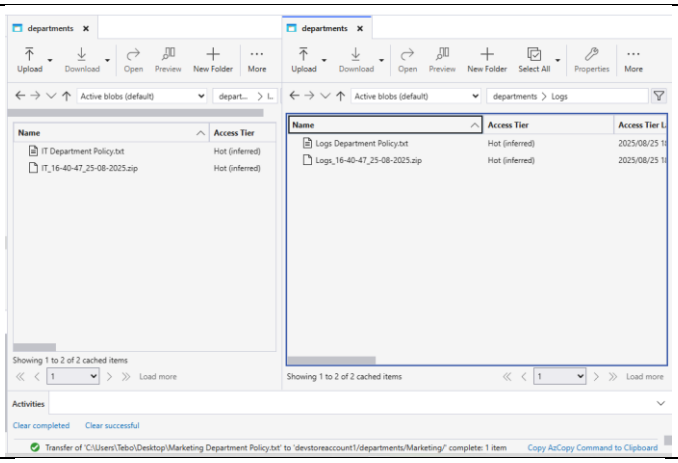
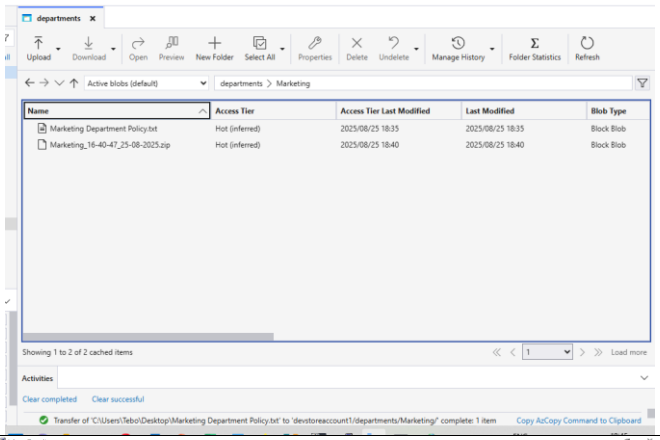
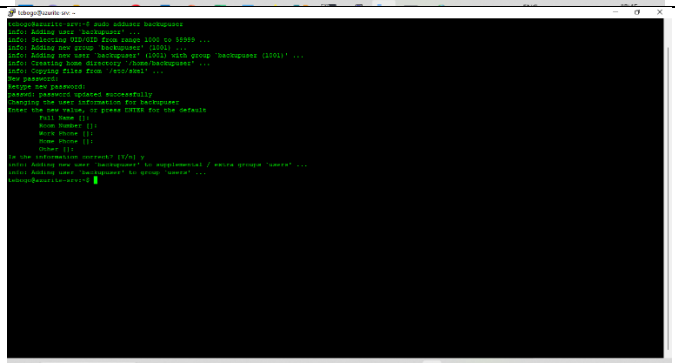
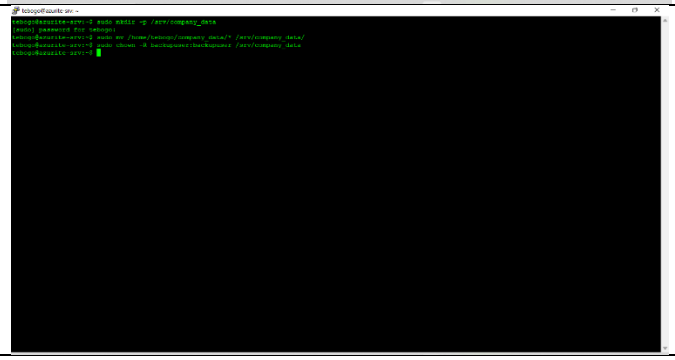
Backup is successful

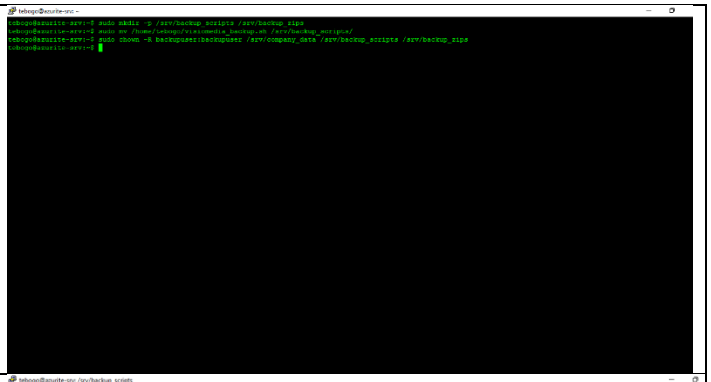
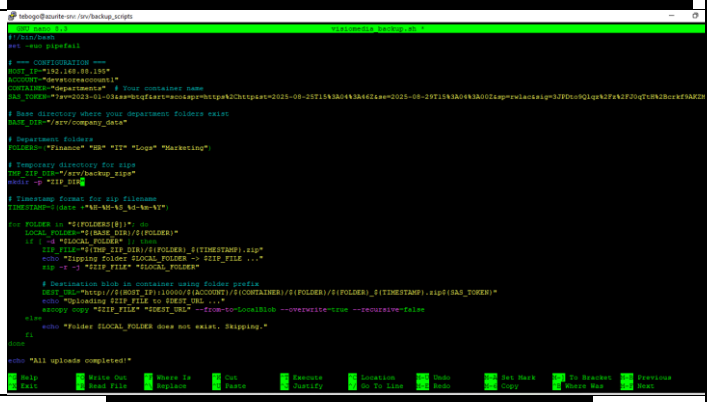
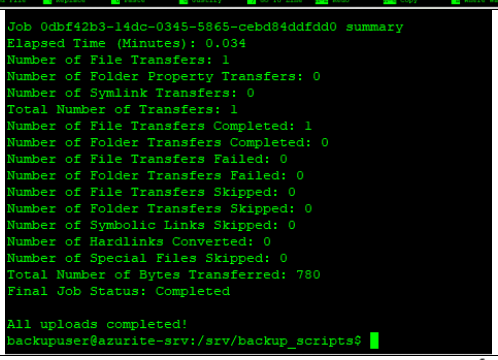
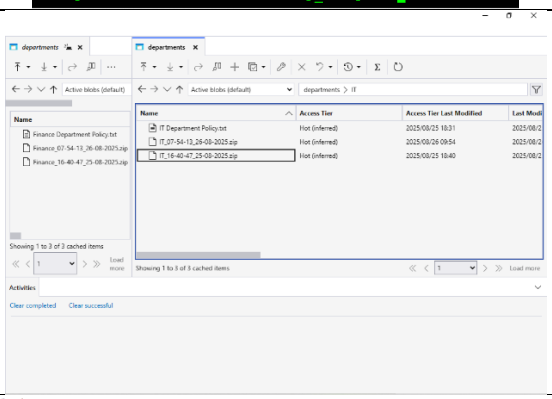
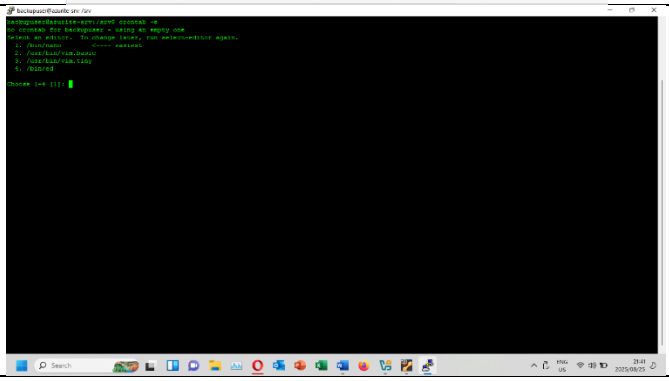
Run script and verify files are correctly uploaded

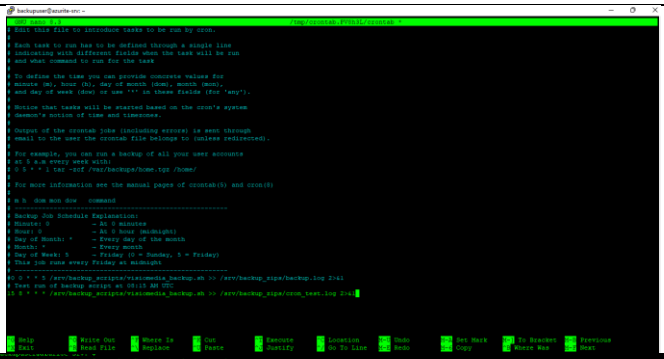
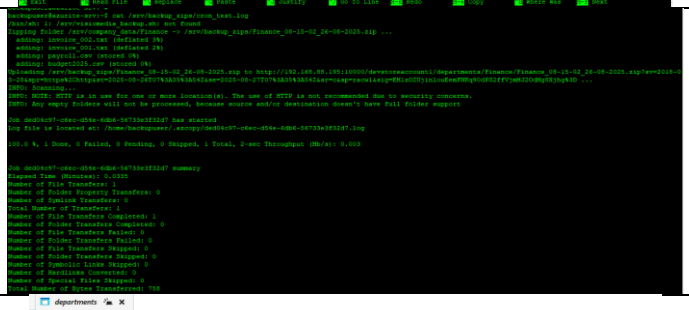
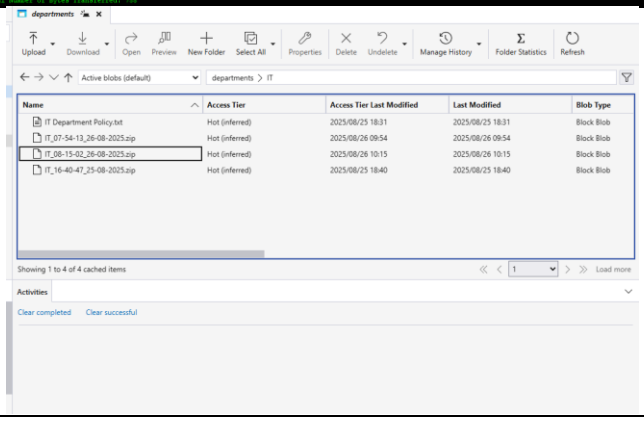


Verify Backup



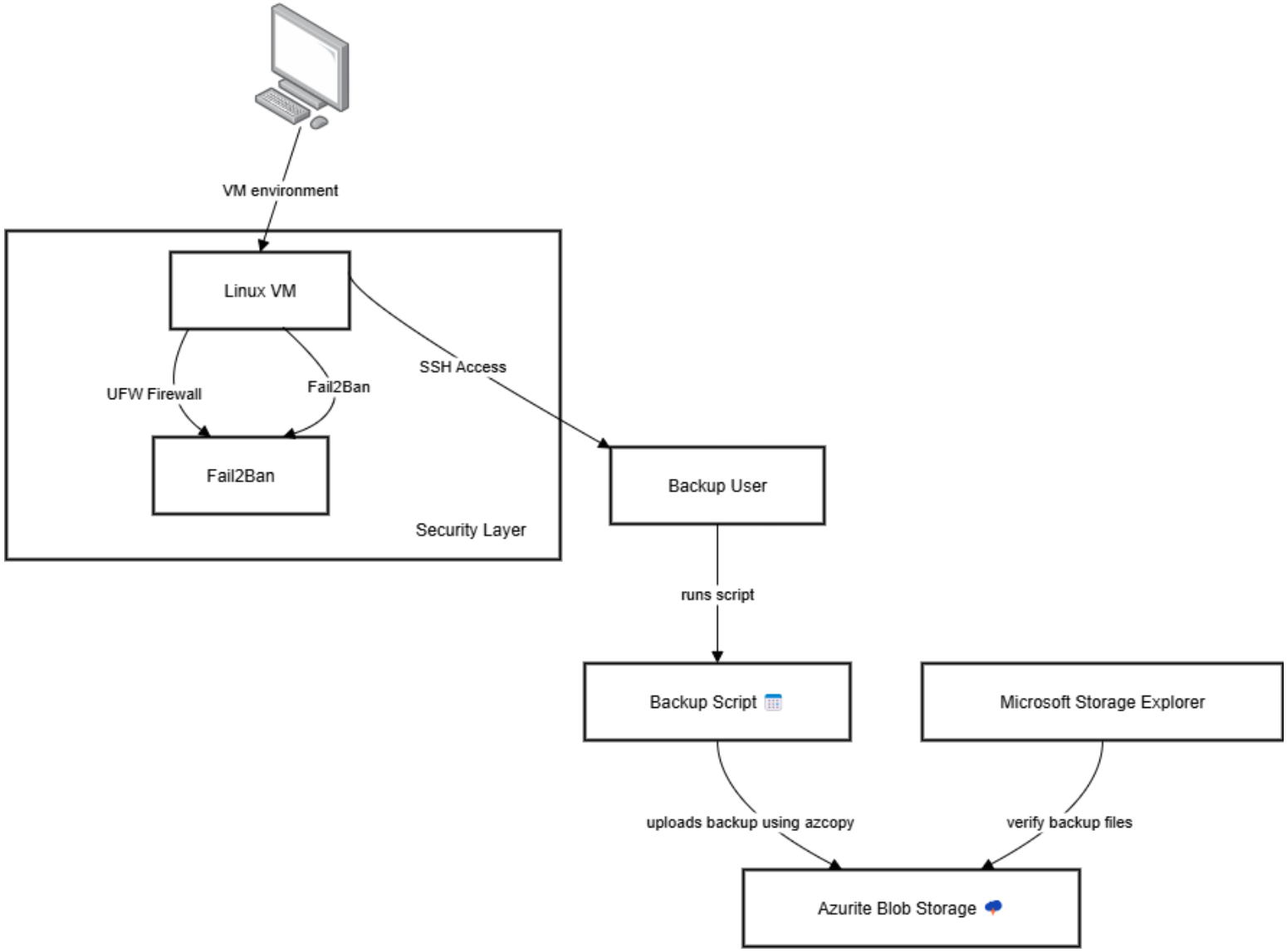
	
	
Create Backupuser Create a dedicated user for running backup scripts	
Move script and hand over ownership Give the new user ownership of the backup script	

sudo chown backupuser:backupuser	
Update Script	
Verify Backup User can use script	
Verify Backup	
Crontab and schedule backup Schedule automatic backups	

Test Crontab Force a manual run to confirm cron executes the script	
Crontab worked	
Done	

Demonstration

This project is a working demo of how to secure a Linux VM and automate backups to Azurite Blob Storage. It shows the full process from hardening the server, configuring SSH, and setting up Fail2Ban, to running backup scripts with AzCopy and automating them with cron.



Improvements

Use Real Azure Storage

While Azurite is great for testing locally, using an actual Azure Blob Storage account would make this setup cloud-ready. This would also allow testing with production features like redundancy, access tiers, and integration with other Azure services.

Encrypt Backups

Adding an encryption layer ensures that even if data is intercepted or the storage account is compromised, files remain secure. This could be done with tools like GPG, OpenSSL, or even Azure Key Vault for enterprise-grade key management.

Monitoring & Alerts

Right now, the only way to confirm backups is through manual checks in Microsoft Storage Explorer. A better approach would be to add automated logging and email/SMS alerts. For example, if a backup fails or the script doesn’t run, the system could notify the administrator immediately.

Infrastructure as Code

Instead of manually configuring the VM, firewall, SSH, and backup scripts, tools like Ansible or Terraform could be used to automate the entire setup. This would make the project repeatable, scalable, and easier to manage across multiple machines or environments.