■ GitHub Enterprise Server

Azure → AWS Migration Guide

Replication Method + Azure Blob → S3 via Rclone

Interview Ready • Architecture • POC • SOP • Costing

GUI Steps • CLI Commands • Rclone Config • GitHub README

Zero-Downtime Cutover • Validation • Rollback Plan

■ Azure GHES	→ MIGRATION →	■ AWS GHES	■ Rclone	■■ Blob→S3

■■ PRODUCTION MIGRATION — Always test in staging first. Ensure full backup before cutover.

■ Table of Contents

1. Migration Overview & Strategy

■ Interview Explanation

This migration moves a GitHub Enterprise Server (GHES) instance from Azure (Virtual Machine) to AWS (EC2) using GHES's built-in geo-replication feature as the data transport mechanism. Simultaneously, any blob/object data stored in Azure Blob Storage (LFS objects, GitHub Actions artifacts, Container Registry images) is migrated to Amazon S3 using rclone — an open-source cloud storage sync tool. The result is a fully functional GHES instance on AWS with zero data loss.

Why Replication Method (vs Backup/Restore)?

Approach	How It Works	RTO	RPO	Risk
Backup & Restore	Export .ghes-backup → transfer → restore on AWS	4–24 hrs	Hours	HIGH
Replication Method ■	Set AWS GHES as replica → sync → promote → cutover	< 1 hour	Minutes	LOW
Database Dump	pg_dump + git bundle → copy → import	8–48 hrs	Hours	HIGH
Ghost Migration	Third-party tool for repo-by-repo	Days	Hours	MEDIUM

Migration Phases at a Glance:

PHASE 1 ASSESS	Inventory Azure GHES: repos, users, Actions runners, Blob data volumes
PHASE 2 PROVISION	Spin up AWS EC2 GHES instance, configure VPC, EBS, Route53, SSL cert
PHASE 3 REPLICATE	Configure GHES geo-replication: Azure primary → AWS replica. Full sync.
PHASE 4 BLOB SYNC	rclone sync Azure Blob Storage → Amazon S3. Delta sync until cutover.
PHASE 5 CUTOVER	Maintenance mode → final sync → promote AWS replica → update DNS
PHASE 6 VALIDATE	Run health checks, smoke tests, user acceptance, monitoring baseline

Key Insight: GHES replication keeps the AWS instance continuously in sync with Azure during the migration window. This means cutover is just a DNS change + replica promotion — measured in minutes, not hours. Data loss risk is near zero.

2. Architecture: Source vs Target

SOURCE — Azure GHES Architecture

```
AZURE CLOUD ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
Azure Virtual Machine (D8s_v3: 8 vCPU, 32 GB RAM) ■ ■ OS: Ubuntu 20.04 LTS ■ ■ GitHub
Enterprise Server 3.x ■ ■ ■ ■ Managed Disks: ■ ■ /dev/sda → Root OS disk (150 GB Premium SSD)
■ ■ /dev/sdb → Data disk (500 GB Premium SSD) ← GHES data■ ■ /dev/sdc → Pages disk (200 GB) ■
■ /dev/sdd → User data disk (300 GB) ■ ■ ■ ■ Azure Virtual Network → Private Subnet ■ ■
Network Security Group (NSG): 443, 8443, 122, 9418 ■ ■ Azure Load Balancer (optional) ■ ■ Azure
AD → SAML SSO Integration ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Azure Blob
Storage Container ■ ■ Account: ghesprodstorage ■ ■ Containers: ■ ■ /git-lfs-objects (Large File
Storage) ■ ■ /actions-artifacts (GitHub Actions build artifacts) ■ ■ /packages (GitHub Packages
/ Container Reg.) ■ ■ /uploads (User file uploads) ■ ■ Access: Storage Account Key / Service
Principal ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
External: corp-ghes.company.com → Azure Public IP
```

TARGET — AWS GHES Architecture

```
AWS CLOUD ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ VPC:
10.0.0.0/16 | Private Subnet: 10.0.10.0/24 ■ ■ ■ ■ EC2 Instance (r6i.2xlarge: 8 vCPU, 64 GB
RAM) ■ ■ AMI: GitHub Enterprise Server 3.x (AWS Marketplace) ■ ■ ■ ■ EBS Volumes: ■ ■
/dev/xvda → Root 200 GB gp3 (OS + GHES) ■ ■ /dev/xvdb → Data 500 GB io2 (30K IOPS) ← GHES data
■ ■ /dev/xvdc → Pages 200 GB gp3 ■ ■ /dev/xvdd → User 300 GB gp3 (encrypted KMS) ■ ■ ■ ■
Security Group: 443(HTTPS), 8443(Admin), 122(SSH), ■ ■ 9418(Git), 1194(VPN replication) ■ ■ IAM
Role: ghes-s3-role (S3 read/write for Blob data) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Amazon S3
Bucket ■ ■ Bucket: ghes-prod-storage-aws (us-east-1) ■ ■ Prefixes (mirrors Azure containers): ■
■ /git-lfs-objects/ /actions-artifacts/ ■ ■ /packages/ /uploads/ ■ ■ Encryption: SSE-KMS (CMK)
■ ■ Versioning: Enabled | Replication: Cross-region to DR ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ Route 53:
corp-ghes.company.com → NLB → EC2 Private IP ACM Certificate: *.company.com (SSL termination at
GHES)
```

Replication Network Flow:

```
DURING MIGRATION (Replication Active): Azure GHES (PRIMARY)
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■→ AWS GHES (REPLICA) 10.x.x.x (Azure VNet) OpenVPN/SSH
tunnel port 1194 10.0.10.x (AWS VPC) Data replicated: ■ All Git repositories (bare repos) ■
Repository metadata, wikis, issues, PRs ■ GitHub Pages ■ GitHub Actions workflow definitions ■
User accounts, teams, organizations ■ SSH keys, personal access tokens (hashed) ■ Webhooks,
integrations, OAuth apps NOT replicated (handled by rclone separately): ■■ Git LFS objects
(stored in Blob/S3) ■■ GitHub Actions artifacts (stored in Blob/S3) ■■ GitHub Packages /
Container Registry images ■■ User file upload attachments
```

3. Prerequisites & Planning

■ Azure Source Requirements

- ■ GHES version: 3.0+ required for geo-replication. Run: ghe-version on Azure VM.
- ■ GHES Admin Console access (port 8443) with site admin credentials.
- ■ Azure Storage Account name, access key (or Service Principal client ID/secret).
- ■ Azure VM: outbound internet access to AWS (or ExpressRoute/VPN connectivity).
- ■ Current GHES license file (.ghl) — needed for AWS instance activation.
- ■ DNS TTL: reduce corp-ghes.company.com TTL to 60 seconds, 48 hours before cutover.
- ■ Identify all GHES integrations: SAML IdP, LDAP, OAuth apps, webhooks, Actions runners.
- ■ Measure current blob storage size: az storage blob list → estimate rclone sync time.

■ AWS Target Requirements

- ■ AWS Account with permissions: EC2, VPC, S3, Route53, ACM, IAM, KMS.
- ■ AWS Marketplace: subscribe to 'GitHub Enterprise Server' AMI (same version as source).
- ■ EC2 instance type: match or exceed Azure VM specs. Minimum r6i.2xlarge (8 vCPU, 64 GB).
- ■ EBS volumes: same capacity as Azure managed disks. Use io2 for data disk.
- ■ S3 bucket pre-created: ghes-prod-storage-aws with versioning and KMS encryption.
- ■ IAM role for EC2: s3:GetObject, s3:PutObject, s3:DeleteObject, s3:ListBucket.
- ■ ACM certificate for corp-ghes.company.com (or import existing cert).
- ■ Route 53 hosted zone for company.com (or external DNS with API access).
- ■ Security Group: ports 443, 8443, 122, 9418, 1194 (replication VPN port).
- ■ VPC with NAT Gateway (GHES needs outbound for license validation, Actions).

■ Rclone Requirements

- ■ Rclone version 1.65+ installed on a migration host (can be the Azure VM or separate VM).
- ■ Migration host must have network access to BOTH Azure Blob and AWS S3.
- ■ Azure credentials: Storage Account Key OR Service Principal (least privilege).
- ■ AWS credentials: IAM user with S3 write access (or assume role from migration host).
- ■ Estimate bandwidth: test with small prefix first. Azure egress: ~$0.08/GB.
- ■ Schedule initial bulk sync during off-hours to minimize impact on Azure egress bills.
- ■ Plan for delta syncs: rclone sync --checksum runs repeatedly until cutover.

■ Network Connectivity

- ■ Azure GHES → AWS GHES: TCP port 1194 (OpenVPN) or 122 (SSH) for replication.
- ■ Option A: Azure VPN Gateway ↔ AWS Site-to-Site VPN (private, most secure).
- ■ Option B: Azure GHES public IP whitelisted in AWS Security Group (acceptable for migration window).
- ■ Option C: ExpressRoute ↔ Direct Connect (enterprise, lowest latency, highest cost).
- ■ Verify connectivity: nc -zv 1194 from Azure GHES VM.
- ■ Recommended throughput: 1 Gbps+ for large repos. Test with iperf3.

4. Phase 1 — Azure GHES Assessment

Run these commands on the Azure GHES VM to gather inventory data before migration.

4.1 GHES Version & License Check

```
# SSH into Azure GHES VM ssh -p 122 admin@ # Check GHES version (must be 3.0+ for
geo-replication) ghe-version # Check license validity and seat count ghe-check-disk-usage cat
/etc/github/enterprise.lic | python3 -m json.tool | grep -E 'expiration|seats|features' # List
GHES configured features ghe-config app.github.hostname ghe-config app.pages.enabled ghe-config
app.actions.enabled ghe-config app.packages.enabled
```

4.2 Repository & User Inventory

```
# Count repositories, users, organizations ghe-report-stats | jq '{ total_repos: .repos.total,
total_users: .users.total, total_orgs: .organizations.total, total_gists: .gists.total,
actions_enabled: .actions.enabled }' # Check disk usage breakdown df -h du -sh
/data/repositories/ du -sh /data/user/ du -sh /data/pages/ du -sh /data/gitlfs/ # Largest
repositories (for bandwidth planning) ghe-repo-stats --top 20 | column -t # Active webhooks
count github-env rails r "puts Hook.where(active: true).count"
```

4.3 Azure Blob Storage Inventory

```
# Using Azure CLI — get storage account details STORAGE_ACCOUNT="ghesprodstorage"
RESOURCE_GROUP="ghes-prod-rg" # Get storage account key az storage account keys list \
--account-name $STORAGE_ACCOUNT \ --resource-group $RESOURCE_GROUP \ --query '[0].value' -o tsv
# List all containers az storage container list \ --account-name $STORAGE_ACCOUNT \ --output
table # Get size of each container for CONTAINER in git-lfs-objects actions-artifacts packages
uploads; do SIZE=$(az storage blob list \ --container-name $CONTAINER \ --account-name
$STORAGE_ACCOUNT \ --query 'sum([].properties.contentLength)' -o tsv) echo "$CONTAINER: $(echo
$SIZE | awk '{printf %.2f GB, $1/1073741824}')" done # Total blob storage size (critical for
estimating rclone time) az storage account show-usage --name $STORAGE_ACCOUNT
```

4.4 Assessment Output Template

Assessment Item	Command	Captured Value
GHES Version	ghe-version	3.x.x
Total Repositories	ghe-report-stats	_ repos
Total Users	ghe-report-stats	_ users
Total Orgs	ghe-report-stats	_ orgs
Data Disk Used	df -h /data	_ GB / _ GB
LFS Objects Size	du -sh /data/gitlfs	_ GB
Blob: git-lfs-objects	az storage blob list	_ GB
Blob: actions-artifacts	az storage blob list	_ GB
Blob: packages	az storage blob list	_ GB
Blob: uploads	az storage blob list	_ GB
SAML Configured	ghe-config	Yes / No

Actions Enabled	ghe-config	Yes / No
License Expiry	enterprise.lic	YYYY-MM-DD

5. Phase 2 — AWS Target Infrastructure Setup

5.1 EC2 Instance from GitHub Enterprise AMI

```
# Step 1: Find GHES AMI in AWS Marketplace (same version as Azure!) aws ec2 describe-images \
--owners aws-marketplace \ --filters 'Name=name,Values=GitHub Enterprise Server 3.*' \ --query
'sort_by(Images,&CreationDate;)[-1].[ImageId,Name]' \ --output table # Step 2: Create key pair
aws ec2 create-key-pair \ --key-name ghes-migration-key \ --query 'KeyMaterial' \ --output text
> ghes-migration-key.pem chmod 400 ghes-migration-key.pem # Step 3: Create Security Group
SG_ID=$(aws ec2 create-security-group \ --group-name ghes-prod-sg \ --description 'GitHub
Enterprise Server Security Group' \ --vpc-id vpc-xxxxxxxx \ --query 'GroupId' --output text) #
Allow GHES required ports aws ec2 authorize-security-group-ingress \ --group-id $SG_ID \
--ip-permissions \ IpProtocol=tcp,FromPort=443,ToPort=443,IpRanges='[{CidrIp=0.0.0.0/0}]' \
IpProtocol=tcp,FromPort=8443,ToPort=8443,IpRanges='[{CidrIp=10.0.0.0/8}]' \
IpProtocol=tcp,FromPort=122,ToPort=122,IpRanges='[{CidrIp=10.0.0.0/8}]' \
IpProtocol=tcp,FromPort=9418,ToPort=9418,IpRanges='[{CidrIp=0.0.0.0/0}]' \
IpProtocol=tcp,FromPort=1194,ToPort=1194,IpRanges='[{CidrIp=/32}]' # Step 4: Launch EC2 with
GHES AMI aws ec2 run-instances \ --image-id ami- \ --instance-type r6i.2xlarge \ --key-name
ghes-migration-key \ --security-group-ids $SG_ID \ --subnet-id subnet-private-xxxxxxxx \
--no-associate-public-ip-address \ --iam-instance-profile Name=ghes-s3-role \
--block-device-mappings \
'[{"DeviceName":"/dev/xvda","Ebs":{"VolumeSize":200,"VolumeType":"gp3","Encrypted":true}}, {"Dev
iceName":"/dev/xvdb","Ebs":{"VolumeSize":500,"VolumeType":"io2","Iops":30000,"Encrypted":true}},
{"DeviceName":"/dev/xvdc","Ebs":{"VolumeSize":200,"VolumeType":"gp3","Encrypted":true}},
{"DeviceName":"/dev/xvdd","Ebs":{"VolumeSize":300,"VolumeType":"gp3","Encrypted":true}}]' \
--tag-specifications 'ResourceType=instance,Tags=[ {Key=Name,Value=ghes-prod-aws},
{Key=Environment,Value=Production}, {Key=Role,Value=GHES-Replica}]'
```

5.2 S3 Bucket Setup for Blob Data Migration

```
# Create S3 bucket with versioning and encryption aws s3api create-bucket \ --bucket
ghes-prod-storage-aws \ --region us-east-1 # Enable versioning aws s3api put-bucket-versioning \
--bucket ghes-prod-storage-aws \ --versioning-configuration Status=Enabled # Enable KMS
encryption aws s3api put-bucket-encryption \ --bucket ghes-prod-storage-aws \
--server-side-encryption-configuration '{ "Rules": [{"ApplyServerSideEncryptionByDefault":
{"SSEAlgorithm": "aws:kms", "KMSMasterKeyID": "arn:aws:kms:us-east-1:ACCT:key/KEY-ID"} }]} #
Block all public access aws s3api put-public-access-block \ --bucket ghes-prod-storage-aws \
--public-access-block-configuration \ BlockPublicAcls=true,IgnorePublicAcls=true,\
BlockPublicPolicy=true,RestrictPublicBuckets=true # Create prefix structure mirroring Azure
containers for prefix in git-lfs-objects actions-artifacts packages uploads; do aws s3api
put-object --bucket ghes-prod-storage-aws --key $prefix/.keep done # Create lifecycle policy
(clean up old Actions artifacts after 90 days) aws s3api put-bucket-lifecycle-configuration \
--bucket ghes-prod-storage-aws \ --lifecycle-configuration file://ghes-s3-lifecycle.json
```

5.3 IAM Role for GHES EC2 → S3 Access

```
# ghes-s3-policy.json { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": [
"s3:GetObject", "s3:PutObject", "s3:DeleteObject", "s3:ListBucket", "s3:GetBucketLocation" ],
"Resource": [ "arn:aws:s3:::ghes-prod-storage-aws", "arn:aws:s3:::ghes-prod-storage-aws/*" ] }]
} aws iam create-policy \ --policy-name GHES-S3-Policy \ --policy-document
file://ghes-s3-policy.json aws iam attach-role-policy \ --role-name ghes-s3-role \ --policy-arn
arn:aws:iam::ACCT:policy/GHES-S3-Policy
```

6. Phase 3 — GHES Geo-Replication Setup

GHES Geo-Replication uses a primary-replica model. The Azure instance is the PRIMARY. The AWS instance will be configured as a REPLICA. All data flows from Azure to AWS continuously until cutover. This is the core migration engine.

6.1 Configure AWS GHES Instance (First Boot)

```
# SSH to AWS GHES instance (via bastion or SSM Session Manager) ssh -i ghes-migration-key.pem -p 122 admin@ # On FIRST BOOT — GHES setup wizard runs. Choose 'New Install' # BUT — we will NOT finish setup. We configure as replica instead. # Access the admin console: https://:8443 # IMPORTANT: Do NOT complete the setup wizard on the AWS instance. # It will be overwritten by replication from Azure primary.
```

■■ WARNING: Do NOT configure the AWS instance as a standalone GHES. If you complete setup, you will need to re-image. The replica must be initialized from the primary — not independently configured.

6.2 Enable Geo-Replication on Azure Primary

```
# On AZURE GHES primary — generate replica key # Via Admin Console (GUI): # https://:8443/setup/settings # → High availability # → Enable high availability mode # → Add replica # OR via CLI on Azure GHES: ghe-repl-setup # This will: # 1. Generate a replication SSH key pair # 2. Output a command to run on the replica # 3. Open SSH tunnel for replication channel # Expected output: # Generating replication key pair... # Run this command on : # ghe-repl-setup-start [KEY-FINGERPRINT] # Check replication configuration: ghe-repl-status
```

6.3 Initialize Replica on AWS GHES

```
# On AWS GHES instance: ssh -p 122 admin@ # Run the command output by ghe-repl-setup on the primary: ghe-repl-setup-start # This copies: # - SSH authorized keys from primary # - Base configuration # - License file # Start initial full replication sync (this takes HOURS for large instances) ghe-repl-start # Monitor replication progress (run on PRIMARY - Azure GHES) watch -n 30 ghe-repl-status # Expected output when fully synced: # OK: mysql replication in sync with the primary # OK: redis replication in sync with the primary # OK: git repositories in sync with the primary # OK: pages repositories in sync with the primary # OK: elasticsearch indices in sync with the primary # OK: git lfs objects in sync with the primary # Check replication lag (should be seconds or minutes, not hours) ghe-repl-status --verbose
```

6.4 Verify Replication Health (Ongoing Monitoring)

```
# On Azure PRIMARY — check all replication streams: ghe-repl-status -vv # Individual component checks: ghe-repl-status git # Git repository sync ghe-repl-status mysql # Database sync ghe-repl-status redis # Cache sync ghe-repl-status elasticsearch # Search index sync ghe-repl-status pages # GitHub Pages sync # Monitor replication lag continuously: watch -n 60 'ghe-repl-status | grep -E "lag|error|OK"' # If replication falls behind — check disk I/O on AWS: iostat -x 5 3 dmesg | grep -i error # Replication logs (on replica - AWS): tail -f /var/log/github/ghe-replication.log
```

7. Phase 4 — Azure Blob → Amazon S3 via Rclone

Rclone is an open-source command-line tool that syncs cloud storage. We use it to transfer all Azure Blob Storage objects to Amazon S3. Rclone runs incremental syncs (only changed/new files) making pre-cutover and cutover syncs fast. Install rclone on a migration host with access to both clouds.

7.1 Install Rclone

```
# Linux (recommended: run on Azure VM to avoid egress costs during sync) curl
https://rclone.org/install.sh | sudo bash # Verify installation rclone version # rclone v1.66.0
# Alternative: install specific version curl -O
https://downloads.rclone.org/v1.66.0/rclone-v1.66.0-linux-amd64.zip unzip
rclone-v1.66.0-linux-amd64.zip sudo mv rclone-v1.66.0-linux-amd64/rclone /usr/local/bin/ sudo
chmod +x /usr/local/bin/rclone
```

7.2 Configure Rclone Remotes

```
# Create rclone config file: ~/.config/rclone/rclone.conf cat > ~/.config/rclone/rclone.conf <<
'EOF' # ■■ AZURE BLOB STORAGE (SOURCE) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ [azure-ghes] type
= azureblob account = ghesprodstorage key = # OR use Service Principal (more secure): # type =
azureblob # account = ghesprodstorage # service_principal_file = /etc/rclone/azure-sp.json # ■■
AMAZON S3 (DESTINATION) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ [aws-ghes-s3] type = s3
provider = AWS region = us-east-1 # IAM role (preferred if running from EC2 with instance
profile) env_auth = true # OR use explicit keys: # access_key_id = AKIAXXXXXXXXXXXXXXXX #
secret_access_key = server_side_encryption = aws:kms sse_kms_key_id =
arn:aws:kms:us-east-1:ACCT:key/KEY-ID storage_class = STANDARD chunk_size = 64M
upload_concurrency = 8 EOF # Test connections rclone lsd azure-ghes: rclone lsd
aws-ghes-s3:ghes-prod-storage-aws
```

7.3 Initial Bulk Sync (Pre-Migration)

```
# Run initial sync — all containers in parallel # This is the BULK transfer that happens BEFORE
cutover CONTAINERS='git-lfs-objects actions-artifacts packages uploads'
S3_BUCKET='ghes-prod-storage-aws' for CONTAINER in $CONTAINERS; do echo "Starting sync:
$CONTAINER" rclone sync \ azure-ghes:$CONTAINER \ aws-ghes-s3:$S3_BUCKET/$CONTAINER \ --progress
\ --transfers 32 \ --checkers 16 \ --buffer-size 256M \ --s3-upload-concurrency 8 \
--s3-chunk-size 64M \ --log-file /var/log/rclone-$CONTAINER-initial.log \ --log-level INFO \
--stats 60s \ --checksum \ & # Run in background done # Wait for all background syncs to complete
wait echo 'Initial bulk sync complete!' # Check sync logs for errors grep -i 'error\|failed'
/var/log/rclone-*.log | head -50
```

7.4 Delta Sync Loop (Runs Until Cutover)

```
# delta-sync.sh — run this repeatedly until cutover day #!/bin/bash set -euo pipefail
CONTAINERS='git-lfs-objects actions-artifacts packages uploads'
S3_BUCKET='ghes-prod-storage-aws' LOG_DIR='/var/log/rclone' mkdir -p $LOG_DIR TIMESTAMP=$(date
+%Y%m%d_%H%M%S) echo "[$TIMESTAMP] Starting delta sync..." for CONTAINER in $CONTAINERS; do
START=$(date +%s) rclone sync \ azure-ghes:$CONTAINER \ aws-ghes-s3:$S3_BUCKET/$CONTAINER \
--transfers 16 \ --checkers 8 \ --checksum \ --max-age 24h \ --log-file
$LOG_DIR/delta-$CONTAINER-$TIMESTAMP.log \ --log-level INFO END=$(date +%s)
DURATION=$((END-START)) echo " $CONTAINER: synced in ${DURATION}s" done echo "[$TIMESTAMP] Delta
sync complete." # Schedule with cron (every 6 hours during migration window) # 0 */6 * * *
/opt/migration/delta-sync.sh >> /var/log/rclone/cron.log 2>&1
```

7.5 Verify Sync Completeness

```
# Count objects: Azure vs S3 (should match) echo '=== Azure Blob Object Counts ===' for
CONTAINER in git-lfs-objects actions-artifacts packages uploads; do COUNT=$(rclone ls
azure-ghes:$CONTAINER | wc -l) SIZE=$(rclone size azure-ghes:$CONTAINER | grep 'Total size')
echo "Azure $CONTAINER: $COUNT objects | $SIZE" done echo '=== S3 Object Counts ===' for
CONTAINER in git-lfs-objects actions-artifacts packages uploads; do COUNT=$(rclone ls
aws-ghes-s3:ghes-prod-storage-aws/$CONTAINER | wc -l) SIZE=$(rclone size
aws-ghes-s3:ghes-prod-storage-aws/$CONTAINER | grep 'Total size') echo "S3 $CONTAINER: $COUNT
objects | $SIZE" done # Use rclone check for byte-for-byte verification (runs checksums) rclone
check \ azure-ghes:git-lfs-objects \ aws-ghes-s3:ghes-prod-storage-aws/git-lfs-objects \
--checksum \ 2>&1 | tee /var/log/rclone/verification.log # Check for any mismatches grep -E
'^\d+ differences' /var/log/rclone/verification.log
```

7.6 Configure GHES AWS to Use S3 (Post-Cutover)

```
# After promoting AWS replica to primary, configure S3 as blob storage: # GitHub Enterprise
Admin Console → https://:8443 # → Settings → GitHub LFS → Amazon S3 # Bucket:
ghes-prod-storage-aws # Region: us-east-1 # Use IAM instance profile: Yes (EC2 has IAM role
attached) # OR via CLI on AWS GHES: ghe-config app.lfs.storage-provider s3 ghe-config
app.lfs.s3-bucket ghes-prod-storage-aws ghe-config app.lfs.s3-region us-east-1 ghe-config
app.lfs.s3-prefix git-lfs-objects ghe-config app.actions.storage-provider s3 ghe-config
app.actions.s3-bucket ghes-prod-storage-aws ghe-config app.packages.s3-bucket
ghes-prod-storage-aws # Apply configuration ghe-config-apply
```

8. Phase 5 — Cutover Procedure

The cutover is the critical final step. It should be performed during a maintenance window (typically 30–60 minutes). All steps below are sequential — do NOT skip any.

■■ CUTOVER CHECKLIST — Assign a named owner to each step. Have rollback plan ready (Section 10). All stakeholders must be on a call.

Time	Phase	Action	Owner
T-48h	Pre-Cutover	Reduce DNS TTL to 60 seconds for corp-ghes.company.com	DNS Admin
T-24h	Pre-Cutover	Run final full rclone sync. Verify 0 differences with rclone check	Migration Eng
T-1h	Pre-Cutover	Confirm ghe-repl-status shows all streams in sync. Lag < 5 minutes	Migration Eng
T-0	CUTOVER	Send maintenance notification to all GHES users	Comms
T+1m	CUTOVER	Enable maintenance mode on Azure PRIMARY: ghe-maintenance -s	Migration Eng
T+2m	CUTOVER	Wait for all active Git operations to drain (watch netstat -an | grep 9418)	Migration Eng
T+3m	CUTOVER	Run FINAL rclone sync: rclone sync azure-ghes: aws-ghes-s3: --checksum	Migration Eng
T+5m	CUTOVER	Confirm replication is fully caught up: ghe-repl-status (all OK)	Migration Eng
T+6m	CUTOVER	PROMOTE AWS replica to PRIMARY: ghe-repl-promote (on AWS GHES)	Migration Eng
T+7m	CUTOVER	Configure S3 as storage backend on AWS GHES (ghe-config + apply)	Migration Eng
T+8m	CUTOVER	Update DNS: corp-ghes.company.com → AWS GHES IP/NLB	DNS Admin
T+10m	CUTOVER	Disable maintenance mode on AWS: ghe-maintenance -u	Migration Eng
T+12m	VALIDATE	Run smoke tests: git clone, push, PR create, Actions trigger	QA Team
T+20m	VALIDATE	Verify LFS downloads work (uses S3 now), check Actions artifacts	QA Team
T+30m	COMPLETE	Announce migration complete. Monitor CloudWatch/GHES dashboards	Migration Eng

9. Phase 6 — Validation & Testing

9.1 GHES System Health Checks

```
# On AWS GHES (now primary) — comprehensive health check ghe-check-disk-usage ghe-diagnostics # Check all services are running ghe-run-scheduled-jobs nomad status consul members # Check GHES system services for svc in github-env github-resqued github-timerd treelights-web; do echo -n "$svc: " systemctl is-active $svc done # GitHub API health check curl -k https://localhost/api/v3 | python3 -m json.tool # Check system resource usage free -h && df -h && uptime # View system monitor (built-in GHES dashboard) # https://:8443/stafftools/system-monitor
```

9.2 Git Operations Smoke Tests

```
#!/bin/bash # smoke-test-git.sh — run AFTER DNS propagation GHES_HOST='corp-ghes.company.com' TEST_TOKEN='ghes_pat_xxxxxxxxxxxx' TEST_ORG='migration-test' echo '=== 1. API Health Check ===' curl -s -H 'Authorization: token $TEST_TOKEN' \ https://$GHES_HOST/api/v3/user | jq '.login' echo '=== 2. Clone Test Repository ===' cd /tmp && rm -rf smoke-test-repo git clone https://$TEST_TOKEN@$GHES_HOST/$TEST_ORG/smoke-test-repo.git echo 'CLONE: SUCCESS' echo '=== 3. Push Commit Test ===' cd /tmp/smoke-test-repo echo "Migration test $(date)" >> test.txt git add . && git commit -m 'Post-migration smoke test' git push origin main echo 'PUSH: SUCCESS' echo '=== 4. Create Pull Request ===' git checkout -b smoke-test-pr echo 'PR test' >> pr-test.txt && git add . && git commit -m 'PR smoke test' git push origin smoke-test-pr curl -s -X POST \ -H 'Authorization: token $TEST_TOKEN' \ -H 'Content-Type: application/json' \ https://$GHES_HOST/api/v3/repos/$TEST_ORG/smoke-test-repo/pulls \ -d '{"title":"Smoke test PR","head":"smoke-test-pr","base":"main"}' \ | jq '.number' echo 'PR CREATE: SUCCESS' echo '=== 5. Git LFS Test ===' git lfs install echo 'LFS test file' > lfs-test.bin git lfs track '*.bin' git add .gitattributes lfs-test.bin git commit -m 'LFS smoke test' git push origin main echo 'LFS PUSH (→ S3): SUCCESS' echo '=== All smoke tests PASSED ==='
```

9.3 Validation Checklist (Sign-Off Required)

■ Infrastructure

- ■ ghe-diagnostics reports no errors
- ■ All GHES services are active (systemctl status)
- ■ Disk usage < 80% on all volumes
- ■ CPU and memory within normal range
- ■ SSL certificate valid for corp-ghes.company.com

■ Git Operations

- ■ Clone via HTTPS works
- ■ Clone via SSH (port 9418) works
- ■ Git push succeeds
- ■ Pull request creation works
- ■ Code review and merge works
- ■ Git LFS push/pull works (data from S3)

■ Blob Storage (S3)

- ■ LFS objects accessible (rclone check shows 0 differences)
- ■ GitHub Actions artifacts downloadable

- ■ GitHub Packages (Container Registry) images pullable
- ■ S3 bucket policy allows GHES EC2 IAM role

■ Integrations

- ■ SAML SSO login works (Azure AD / Okta)
- ■ LDAP user sync completes
- ■ Webhooks firing to external systems
- ■ GitHub Actions runners connected
- ■ GitHub Apps still functional
- ■ OAuth Applications accessible

■ Observability

- ■ CloudWatch metrics flowing (CPU, disk, network)
- ■ GHES audit log streaming to S3/CloudTrail
- ■ Alerting configured (PagerDuty/SNS)
- ■ Performance Insights baseline established

10. Rollback Plan

If the cutover fails or critical issues are found within 2 hours of go-live, execute this rollback plan. Decision point: if any P0 issue is found that cannot be resolved in 30 minutes → initiate rollback immediately.

```
ROLLBACK DECISION: If after cutover you find: - GHES services not starting on AWS - Data corruption detected - S3 storage config broken - Critical integrations broken → ROLLBACK IMMEDIATELY ROLLBACK STEPS: Step 1: Revert DNS immediately (60s TTL already set!) aws route53 change-resource-record-sets \ --hosted-zone-id ZXXXXXX \ --change-batch file://rollback-dns.json # rollback-dns.json points corp-ghes.company.com back to Azure IP Step 2: Re-enable Azure GHES (if maintenance mode was enabled) # SSH to Azure GHES: ghe-maintenance -u Step 3: Verify Azure GHES is accepting traffic curl -k https://corp-ghes.company.com/api/v3 | jq '.message' Step 4: If AWS GHES was promoted, re-configure Azure as primary # On Azure GHES: ghe-repl-stop # Stop any residual replication ghe-repl-teardown # Remove replica config # Azure is now standalone primary again Step 5: Notify all stakeholders — migration postponed Step 6: Post-mortem — identify root cause before re-attempting Step 7: Azure Blob data was NOT deleted during migration → No rollback needed for blob data → S3 data can be left or deleted (it's a copy, not a move) RTO for rollback: ~5 minutes (DNS TTL is 60 seconds) RPO for rollback: Zero — Azure primary was in maintenance mode, no writes accepted during cutover window
```

Rollback Go/No-Go Criteria:

Condition	Action
ghe-repl-promote fails	ROLLBACK — re-run from Step 5 of cutover
S3 storage config fails	ROLLBACK — investigate IAM permissions
DNS propagation fails	WAIT 5min, retry. If >15min → ROLLBACK
SAML SSO broken after cutover	ROLLBACK → fix IdP metadata, re-attempt
GitHub Actions runners not connecting	NO ROLLBACK — fix runner config on AWS
Performance degraded (>20% slower)	MONITOR 30min → if no improvement → ROLLBACK
LFS downloads returning 404	CHECK S3 config → if unfixable in 15min → ROLLBACK

11. Benefits & Features

■ Near-Zero Downtime
Replication method means data is continuously synced. Cutover is just a DNS change + promote — 5–10 minutes of maintenance window vs 4–24 hours for backup/restore.

■ Near-Zero Data Loss
Synchronous replication keeps AWS replica current. With maintenance mode + final sync before promote, RPO is effectively zero.

■ Incremental Rclone
Rclone's --checksum mode only transfers new/changed blobs. After initial bulk sync, delta syncs complete in minutes regardless of total storage size.

■■ Safe Rollback
Azure remains untouched as primary until you decommission it. With 60s DNS TTL, rollback takes ~5 minutes. No data loss on rollback — Azure Blob untouched.

■ Encrypted Pipeline
EBS volumes (KMS), S3 objects (SSE-KMS), replication channel (SSH/TLS), rclone over TLS. End-to-end encryption throughout migration.

■■ GHES Native Tool
ghe-repl-* commands are GitHub's official replication toolset — tested, documented, supported. No third-party migration agents required.

■ Full Observability
CloudWatch metrics on AWS from day 1. ghe-diagnostics, Performance Insights, rclone stats — full visibility into every migration stage.

■■ Cloud Native Target
AWS EC2 + S3 gives you Auto Scaling, Multi-AZ, EBS snapshots, Route 53 failover, AWS WAF, Shield DDoS protection post-migration.

■ Cost Optimization
Move from Azure Premium SSD to AWS gp3/io2 (20–30% cheaper). Azure Blob egress: one-time cost. S3 storage typically 15–20% cheaper than Azure Blob.

■■ Enterprise Security
AWS IAM roles for S3 access (no keys in config), KMS CMKs, VPC isolation, Security Groups, CloudTrail audit logging, AWS Config compliance.

12. Costing Analysis

One-Time Migration Costs:

Cost Item	Provider	Estimate	Notes
Azure VM egress (Blob data)	Azure	$0.08/GB	Only pay for data leaving Azure to internet
EC2 instance (parallel run ~1wk)	AWS	~$200–400	r6i.2xlarge On-Demand during migration
S3 PUT requests (initial sync)	AWS	~$5–50	$0.005/1000 PUTs, depends on object count
Data transfer to S3	AWS	$0 (free in)	S3 ingress is free — no cost for receiving
Rclone compute (migration host)	Azure/AWS	~$50–100	Small VM running rclone during migration
Route 53 changes	AWS	~$1	Negligible — $0.50/hosted zone/month
Engineers (40 hours @ $150/hr)	Internal	~$6,000	Planning, execution, validation
TOTAL ONE-TIME		**~$6,500–7,500**	**Primarily engineer time**

Monthly Ongoing Costs: Azure vs AWS Comparison:

Component	Azure (Current)	Monthly	AWS (Target)	Monthly	Saving
Compute	D8s_v3 (8vCPU 32GB)	$387	r6i.2xlarge (8vCPU 64GB)	$420	-$33
Compute (RI 1yr)	Azure Reserved 1yr	$232	r6i.2xlarge RI 1yr	$265	-$33
Root OS disk	256GB Premium SSD P15	$35	200GB gp3	$23	+$12
Data disk	512GB Premium SSD P30	$73	500GB io2 (30K IOPS)	$82	-$9
Pages disk	256GB Premium SSD P15	$35	200GB gp3	$23	+$12
Blob / S3 storage	Azure Blob (1 TB LRS)	$20	S3 Standard (1 TB)	$23	-$3
Load balancer	Azure Standard LB	$18	AWS NLB	$18	$0
Egress (100 GB/mo)	Azure Egress	$8	AWS Egress	$9	-$1
TOTAL (On-Demand)		**$576/mo**		**$578/mo**	**≈ Same**
TOTAL (RI 1yr)		**$421/mo**		**$443/mo**	**Similar**

Note: Primary financial benefit is strategic (AWS ecosystem, native S3 for LFS, better Actions runners, direct connect to other AWS workloads) rather than pure cost savings. If GHES workloads are already on AWS, you eliminate cross-cloud egress charges (~$0.08/GB).

Azure Egress Cost for Migration (One-Time):

```
Azure charges for data egress to internet (rclone transfers): - First 100 GB/month: FREE - Next 9.9 TB: $0.087/GB Example: 500 GB Blob data migration = (500 - 100) × $0.087 = $34.80 Azure egress cost Example: 5 TB Blob data migration = (5000 - 100) × $0.087 = $426.30 Azure egress cost Tip: Run rclone FROM within Azure (on Azure VM) → route to S3 via internet This minimizes Azure
```

```
egress (intra-Azure bandwidth is free for sync source compute) Alternatively use Azure
ExpressRoute + AWS Direct Connect to avoid internet egress entirely
```

13. Enterprise Architecture Diagram

During Migration (Replication Active):

```
AZURE CLOUD AWS CLOUD ■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■ GitHub
Enterprise ■ ■ GitHub Enterprise ■ ■ Server (PRIMARY) ■ ■ Server (REPLICA) ■ ■ ■■■■■ Geo
■■■■■■ ■ ■ corp-ghes.company.com■ Replication ■ 10.0.10.50 (Private) ■ ■ D8s_v3 VM ■ ghe-repl
■ r6i.2xlarge EC2 ■ ■ Premium SSD: 1.25TB ■ port 1194 ■ EBS: 1.2TB (enc KMS) ■
■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■
■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■ Azure Blob Storage ■■■■■
rclone ■■■ Amazon S3 ■ ■ ghesprodstorage ■ (continuous ■ ghes-prod-storage-aws ■ ■
/git-lfs-objects ■ delta sync)■ /git-lfs-objects ■ ■ /actions-artifacts ■ ■ /actions-artifacts
■ ■ /packages ■ ■ /packages ■ ■ /uploads ■ ■ /uploads ■ ■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■ [Migration Host
(VM)]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■^ Runs rclone sync every 6 hours Access:
Azure Storage Key + AWS IAM Role [Users] ■■DNS■■■ corp-ghes.company.com ■■■ Azure GHES
(active) (Azure primary serves all traffic)
```

After Cutover (AWS is Primary):

```
AZURE CLOUD (DECOMMISSION) AWS CLOUD (PRODUCTION) ■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ GitHub Enterprise ■ ■ VPC: 10.0.0.0/16 ■ ■
Server (DECOMMISSION)■ ■ ■ ■ Kept 30 days then ■ ■ Route 53 (DNS) ■ ■ shut down ■ ■
corp-ghes.company.com ■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■ ↓ ■ ■ Network Load Balancer ■
■■■■■■■■■■■■■■■■■■■■■■■■ ■ ↓ ■ ■ Azure Blob Storage ■ ■ EC2: GitHub Enterprise Server ■
■ Read-only archive ■ ■ r6i.2xlarge (PRIMARY) ■ ■ Delete after 90 days ■ ■ IAM Role → S3
access ■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■ ↓ ■ ■ Amazon S3 (Blob Storage) ■ ■
ghes-prod-storage-aws ■ ■ SSE-KMS | Versioning | CRR ■ ■ ■ ■ CloudWatch → Alarms → PagerDuty
■ ■ CloudTrail → S3 → Athena (audit) ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ [Users]
■■DNS■■■ corp-ghes.company.com ■■■ AWS NLB ■■■ AWS GHES (active) [Git LFS]
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ S3 (via GHES) [Actions Artifacts]
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ S3 (via GHES)
```

14. Proof of Concept (POC)

Run this POC in a staging environment BEFORE production migration. Use a GHES staging instance on Azure and a test EC2 on AWS. This validates all assumptions before touching production.

POC Objectives:

- ■ Confirm GHES replication works between Azure and AWS
- ■ Measure replication lag under load (use GitHub API to create test repos)
- ■ Confirm rclone can sync Azure Blob to S3 at expected speed
- ■ Validate cutover procedure (promote replica → update DNS)
- ■ Verify Git LFS works after switching to S3 as backend
- ■ Measure total cutover time
- ■ Test rollback procedure

POC Environment Setup:

```
# POC uses small instances to keep costs low # Azure staging GHES: B4ms (4 vCPU, 16 GB RAM) # AWS test GHES: m5.xlarge (4 vCPU, 16 GB RAM) # Step 1: Restore staging GHES from production backup # (Azure GHES staging should be a recent snapshot of prod) # Step 2: Populate test blob data in Azure Blob staging container STAGING_STORAGE='ghesstaging' # Create test LFS objects (100 MB each x 50 = 5 GB test data) for i in $(seq 1 50); do dd if=/dev/urandom bs=1M count=100 | \ az storage blob upload --container-name git-lfs-objects \ --account-name $STAGING_STORAGE \ --name 'test-lfs-obj-'$i --data @- done # Step 3: Run replication setup as documented in Phase 3 # (same steps as production) # Step 4: Load test during replication # Simulate git push activity on staging GHES for i in $(seq 1 100); do curl -s -X POST \ -H 'Authorization: token $STAGING_TOKEN' \ https://staging-ghes.company.com/api/v3/orgs/test-org/repos \ -d '{"name":"load-test-'$i'","private":true}' done # Step 5: Measure replication lag under load watch -n 10 'ghe-repl-status | grep lag' # Step 6: Rclone speed test time rclone sync \ azure-ghes-staging:git-lfs-objects \ aws-ghes-s3-staging:ghes-staging-bucket/git-lfs-objects \ --transfers 32 --stats 10s # Record: GB/s throughput and time for 5 GB # Step 7: Execute full cutover as per Phase 5 steps # Record: total time from maintenance mode to DNS propagation # Step 8: Validate all smoke tests pass # Step 9: Execute rollback and verify it works # Step 10: Document results → go/no-go for production
```

POC Success Criteria:

Metric	Target	Measured Value
Replication lag (idle)	< 60 seconds	_____ seconds
Replication lag (under load)	< 5 minutes	_____ minutes
Rclone throughput	> 100 MB/s	_____ MB/s
Total cutover time	< 30 minutes	_____ minutes
Rollback time	< 10 minutes	_____ minutes
LFS upload to S3	Works correctly	Pass / Fail
LFS download from S3	Works correctly	Pass / Fail

Git clone post-cutover	Works correctly	Pass / Fail
SAML login post-cutover	Works correctly	Pass / Fail

15. Standard Operating Procedures (SOP)

■ SOP-GHES-MIG-001: Pre-Migration Preparation

```
1. Complete Azure GHES assessment (Section 4 commands)
2. Document all current GHES settings: Admin Console → Download diagnostics
3. Export current configuration: ghe-config-export > ghes-config-backup.json
4. Notify all GHES users of upcoming maintenance window (1 week notice)
5. Reduce DNS TTL to 60 seconds (48h before cutover)
6. Verify Azure GHES license has enough seats for AWS instance
7. Provision AWS EC2 and S3 (Phase 2 steps)
8. Run POC in staging environment — document results
9. Create rollback runbook with named owners
10. Schedule maintenance window — get change approval
```

■ SOP-GHES-MIG-002: Rclone Daily Monitoring During Migration

```
Daily procedure while replication is active:
1. Check rclone cron completed: grep 'delta sync complete' /var/log/rclone/cron.log
2. Check for rclone errors: grep -i 'error\|failed' /var/log/rclone/*.log | tail -20
3. Verify object counts are increasing (not decreasing): rclone lsf --count
4. Check S3 bucket size growing: aws s3 ls --summarize --human-readable --recursive
s3://ghes-prod-storage-aws
5. Check Azure Blob size for comparison
6. Review AWS S3 storage metrics in CloudWatch
7. Document daily: Date, Azure object count, S3 object count, delta sync duration
8. Escalate if sync duration > 2 hours (indicates performance issue)
```

■ SOP-GHES-MIG-003: GHES Replication Daily Health Check

```
Daily procedure while GHES replication is active:
1. SSH to Azure GHES: ghe-repl-status → all streams must show OK
2. Check replication lag: ghe-repl-status --verbose | grep 'lag'
3. If lag > 30 minutes: investigate immediately
4. Check replication logs on replica (AWS): tail /var/log/github/ghe-replication.log
5. Verify network connectivity: ping -c 5 from Azure GHES
6. Check bandwidth utilization: iftop or nethogs on Azure GHES
7. If replication is stuck: ghe-repl-stop && ghe-repl-start
8. If replication fails to restart: open GitHub Support ticket immediately
```

■ SOP-GHES-MIG-004: Post-Migration AWS GHES Hardening

```
Run within 24 hours of successful cutover:
1. Change all admin passwords on AWS GHES
2. Rotate GitHub Application credentials (OAuth secrets)
3. Update all webhook URLs to use new GHES hostname (if changed)
4. Reconnect GitHub Actions self-hosted runners to AWS GHES
5. Configure SAML/LDAP IdP metadata for AWS GHES hostname
6. Enable AWS WAF in front of NLB (GHES web traffic)
7. Configure CloudTrail logging → S3 for audit trail
8. Set up CloudWatch alarms: CPU > 80%, Disk > 80%, Memory > 90%
9. Configure automated EBS snapshots (daily, 30-day retention)
10. Update CMDB: decommission Azure GHES entry, add AWS GHES entry
```

■ SOP-GHES-MIG-005: Azure Resource Decommission (30 Days Post-Migration)

```
DO NOT decommission until 30 days post-migration with zero issues:
1. Confirm all teams are successfully using AWS GHES
2. Confirm no outstanding support tickets related to migration
3. Take final snapshot of Azure GHES VM (archive for 90 days)
4. Export final Azure Blob Storage inventory (for audit trail)
5. Stop Azure GHES VM (do not delete — keep 30 more days)
6. Set S3 lifecycle: move old Azure Blob copies to S3 Glacier (save cost)
7. After 60 days total: delete Azure Blob containers
8. After 60 days total: delete Azure VM and managed disks
9. Cancel Azure subscription if GHES was the only workload
10. Calculate and report total migration cost vs budget
```

16. GitHub README (Professional Format)

```
# ■ GHES Azure → AWS Migration

![GHES](https://img.shields.io/badge/GitHub%20Enterprise-3.x-black?logo=github)
![Rclone](https://img.shields.io/badge/rclone-Blob→S3-teal)
![AWS](https://img.shields.io/badge/AWS-EC2+S3-orange?logo=amazonaws)
![License](https://img.shields.io/badge/license-MIT-green)

Production-grade migration of GitHub Enterprise Server from Azure VM
to AWS EC2 using GHES geo-replication + Azure Blob → S3 via rclone.

## ■ Migration Strategy
- **Method**: GHES Geo-Replication (primary → replica)
- **Blob Migration**: rclone sync (Azure Blob → Amazon S3)
- **Expected Downtime**: 5–10 minutes (maintenance window only)
- **RPO**: Near-zero (continuous replication)
- **RTO**: < 30 minutes

## ■ Repository Structure
ghes-azure-aws-migration/
■■■ docs/               # Architecture diagrams, runbooks
■■■ scripts/
■   ■■■ assess/         # Azure GHES assessment scripts
■   ■■■ rclone/         # rclone sync and delta sync scripts
■   ■■■ cutover/        # Cutover automation scripts
■   ■■■ validate/       # Smoke test and validation scripts
■■■ terraform/          # AWS infrastructure as code
■   ■■■ ec2.tf          # GHES EC2 instance
■   ■■■ s3.tf           # S3 bucket + lifecycle policies
■   ■■■ iam.tf          # IAM role for EC2 → S3
■   ■■■ security_group.tf
■   ■■■ route53.tf
■■■ rclone/
■   ■■■ rclone.conf.template  # Rclone config template
■   ■■■ lifecycle.json        # S3 lifecycle configuration
■■■ runbooks/           # Step-by-step SOPs

## ■ Quick Start
```bash
git clone https://github.com/your-org/ghes-azure-aws-migration.git
cd ghes-azure-aws-migration

# 1. Run assessment on Azure GHES
bash scripts/assess/azure-ghes-inventory.sh

# 2. Provision AWS infrastructure
cd terraform && terraform init && terraform apply

# 3. Configure rclone
cp rclone/rclone.conf.template ~/.config/rclone/rclone.conf
# Edit with your Azure/AWS credentials

# 4. Initial bulk blob sync
bash scripts/rclone/initial-sync.sh
```
```

```
# 5. Set up GHES replication (see docs/replication-setup.md)

# 6. Monitor delta syncs
bash scripts/rclone/delta-sync.sh

# 7. Execute cutover (follow cutover/runbook.md exactly)
```

■■ Variables

Variable	Description
AZURE_STORAGE_ACC	Azure storage account name
S3_BUCKET	Target S3 bucket name
AWS_REGION	Target AWS region
GHES_VERSION	Must match Azure GHES version

■ Security Notes

- Never commit rclone.conf (contains credentials)
- Use AWS IAM role on EC2 (no access keys in config)
- Azure: use Service Principal (not Storage Account Key)
- All S3 data encrypted with KMS CMK

■ Estimated Migration Cost

- Azure egress: ~$35–450 (depends on blob data size)
- AWS compute during migration: ~$200–400
- Engineer time: ~40 hours

■ License

MIT License — Internal use for GHES migration

17. Top 20 Interview Q&A;

Q1: What is GHES?	GitHub Enterprise Server — self-hosted GitHub deployed on your own infrastructure (VM/EC2). You manage the server, AWS/Azure manages the infrastructure. Gives full data sovereignty.
Q2: Why migrate GHES from Azure to AWS?	Cost optimization (other workloads on AWS → eliminate cross-cloud egress), AWS-native integrations (CodePipeline, ECS, Lambda), better EC2 pricing with Reserved Instances, S3 for LFS/artifacts (cheaper than Azure Blob for large volumes).
Q3: What is GHES geo-replication?	GHES built-in feature that maintains a synchronous or near-synchronous replica of all GHES data (Git repos, MySQL, Redis, Elasticsearch, Pages) on a secondary server. Used for HA and DR.
Q4: Why use replication over backup/restore?	Replication: continuous sync → RTO < 1 hour, RPO near-zero. Backup/restore: point-in-time snapshot → RTO 4–24 hours, RPO hours. Replication runs while users continue working — cutover is just DNS change + promote.
Q5: What data does GHES replication NOT cover?	Git LFS objects, GitHub Actions artifacts, GitHub Packages (Container Registry), and user file uploads — these are stored in external object storage (Azure Blob/S3), NOT in GHES's internal storage. That's why rclone is needed separately.
Q6: What is rclone?	Open-source CLI tool (Go-based) that syncs cloud storage across 70+ providers. Supports Azure Blob, Amazon S3, GCS, Backblaze, etc. Incremental sync with checksums. Used here to transfer Azure Blob → S3.
Q7: How does rclone --checksum work?	Instead of comparing modification time (which may differ across clouds), --checksum compares MD5/SHA1 hashes. Only transfers files with different checksums. More accurate but slower than default time-based comparison.
Q8: How do you minimize Azure egress costs during migration?	1. Run rclone from an Azure VM (intra-Azure bandwidth is free/cheap). 2. Schedule initial sync during off-peak hours. 3. Use --max-age for delta syncs to only transfer recent changes. 4. Consider Azure ExpressRoute + AWS Direct Connect to avoid internet egress.
Q9: What GHES version is required for geo-replication?	GHES 3.0 and later. Check with ghe-version. If running older GHES, must upgrade to 3.0+ before configuring replication.
Q10: How long does initial GHES replication take?	Depends on data volume. Rough estimates: < 100 GB: 1–2 hours | 1 TB: 4–8 hours | 5 TB: 24+ hours. Run during off-hours. Monitor with ghe-repl-status.

Q11: What is ghe-repl-promote?	Command run on the REPLICA that promotes it to PRIMARY. Disconnects from primary, switches to standalone mode, allows writes. Irreversible without reconfiguring replication. Run during cutover AFTER final sync confirmed.
Q12: How does DNS cutover work?	Pre-reduce DNS TTL to 60s (48h before cutover). During cutover: update A/CNAME record from Azure GHES IP to AWS GHES IP/NLB. DNS propagates in ~60 seconds. Old TTL would have kept users on Azure for hours — low TTL enables fast switchover.
Q13: How do you configure GHES to use S3 for LFS?	Admin Console → Settings → GitHub LFS → Amazon S3. Set bucket name, region, use IAM instance profile (EC2 role). Or CLI: ghe-config app.lfs.storage-provider s3 + ghe-config-apply.
Q14: What ports does GHES replication use?	Port 1194 (OpenVPN for replication tunnel) and port 122 (SSH). Must be open between Azure primary and AWS replica in both security group and NSG.
Q15: What is the maintenance mode in GHES?	Puts GHES into read-only mode. Users can browse but cannot push/create. Enable: ghe-maintenance -s. Disable: ghe-maintenance -u. Used during cutover to prevent writes after final replication sync.
Q16: What is the difference between rclone sync and rclone copy?	rclone copy: one-way copy, never deletes destination files. rclone sync: makes destination IDENTICAL to source — deletes files in destination that don't exist in source. Use sync for migration (ensures exact copy).
Q17: How do you verify rclone migration completeness?	rclone check: compares source and destination with checksums, reports differences. rclone ls: count objects on both sides. rclone size: compare total bytes. All three should match after final sync.
Q18: What are GHES Actions runner considerations for migration?	Self-hosted runners must be re-registered to AWS GHES. Runner registration tokens expire. Hosted runners (GHES.com runners) need updated GHES endpoint URL. Action secrets and variables are replicated — no need to recreate.
Q19: What security steps post-migration?	Rotate admin passwords, rotate OAuth app secrets, update webhooks to new hostname, reconfigure SAML IdP with new entityID, enable WAF, set up CloudTrail logging, configure automated EBS snapshots, update IP allowlists for API access.
Q20: What is the rollback strategy?	Azure GHES kept in maintenance mode during cutover window. If cutover fails: revert DNS (60s TTL), disable maintenance mode on Azure, Azure GHES serves traffic again. Data: Azure Blob untouched. GHES data: Azure primary had no writes during maintenance mode — still current.

■ GHES AZURE → AWS MIGRATION COMPLETE — YOU'RE READY ■