GitHub Actions Self-Hosted Runner Setup Guide

Windows Server 2025 on Azure with Hyper-V and Failover Clustering

This guide provides complete instructions for deploying and registering a GitHub Actions self-hosted runner on a Windows Server 2025 Azure VM with Hyper-V and Failover Clustering features enabled.

Table of Contents

Prerequisites

Before starting, ensure you have:

- An Azure subscription with permissions to create VMs
- A GitHub repository or organization where you have admin access
- Azure CLI or Azure Portal access
- Remote Desktop (RDP) access to the Azure VM
- (Optional) A domain account for the runner service if using Active Directory

---

Azure VM Requirements

Supported VM Sizes for Nested Virtualization

Hyper-V on Azure requires nested virtualization support. Use one of these VM series:

Series	Example Sizes	Notes
Dv3/Dsv3	Standard_D4s_v3, Standard_D8s_v3	General purpose
Dv4/Dsv4	Standard_D4s_v4, Standard_D8s_v4	Newer generation
Dv5/Dsv5	Standard_D4s_v5, Standard_D8s_v5	Latest generation
Ev3/Esv3	Standard_E4s_v3, Standard_E8s_v3	Memory optimized
Ev4/Esv4	Standard_E4s_v4, Standard_E8s_v4	Memory optimized
Ev5/Esv5	Standard_E4s_v5, Standard_E8s_v5	Latest memory optimized

Create the Azure VM via Azure CLI

powershell

```
# Login to Azure
az login

# Set variables
$resourceGroup = "rg-github-runners"
$location = "eastus"
$vmName = "vm-github-runner-01"
$vmSize = "Standard_D8s_v5"
$adminUsername = "azureadmin"

# Create resource group
az group create --name $resourceGroup --location $location

# Create the VM with Windows Server 2025
az vm create `
  --resource-group $resourceGroup `
  --name $vmName `
  --image "MicrosoftWindowsServer:WindowsServer:2025-datacenter-azure-edition:latest" `
  --size $vmSize `
  --admin-username $adminUsername `
  --admin-password "<YourSecurePassword>" `
  --public-ip-sku Standard `
  --nsg-rule RDP

# Open RDP port (if not already open)
az vm open-port --port 3389 --resource-group $resourceGroup --name $vmName
```

Create via Azure Portal

1. Navigate to **Virtual Machines → Create**
2. Select **Windows Server 2025 Datacenter: Azure Edition**
3. Choose a VM size that supports nested virtualization (Dv3/Dv4/Dv5 or Ev3/Ev4/Ev5 series)
4. Configure networking, disks, and management options as needed
5. Review and create

Prepare Windows Server

Connect to the VM

```
powershell
```

```
# Get the public IP
az vm show -d -g $resourceGroup -n $vmName --query publicIps -o tsv

# Connect via RDP using the IP address
mstsc /v:<public-ip>
```

Install Required Windows Features

Open PowerShell as Administrator on the VM:

```
powershell

# Install Hyper-V and Failover Clustering with management tools
Install-WindowsFeature -Name Hyper-V -IncludeManagementTools -Restart
```

After reboot, continue with Failover Clustering:

```
powershell

# Install Failover Clustering
Install-WindowsFeature -Name Failover-Clustering -IncludeManagementTools

# Verify installation
Get-WindowsFeature -Name Hyper-V, Failover-Clustering

# Expected output should show [X] for both features
```

Verify Hyper-V is Working

```
powershell

# Check Hyper-V service status
Get-Service vmms, vmcompute

# List virtual switches (will be empty initially)
Get-VMSwitch

# Create a default virtual switch for VMs (optional)
New-VMSwitch -Name "InternalSwitch" -SwitchType Internal
```

Configure Failover Clustering (If Using Cluster)

For a single-node cluster (useful for testing):

```
powershell
```

```
# Create a single-node cluster (for dev/test scenarios)
New-Cluster -Name "GitHubRunnerCluster" -Node $env:COMPUTERNAME -NoStorage -AdministrativeAccessPoint Dns

# Verify cluster
Get-Cluster
Get-ClusterNode
```

For a multi-node cluster, follow standard Azure Failover Cluster setup procedures with shared storage.

Download GitHub Actions Runner

Create Runner Directory

```
powershell

# Create directory for the runner
New-Item -ItemType Directory -Path "C:\actions-runner" -Force
Set-Location -Path "C:\actions-runner"
```

Download the Latest Runner Package

Check the latest version at: https://github.com/actions/runner/releases

```
powershell
```

```powershell
# Set the runner version (update to latest)
$runnerVersion = "2.321.0"

# Download the runner package
$downloadUrl = "https://github.com/actions/runner/releases/download/v$runnerVersion/actions-runner-win-x64-$runnerVersi
Invoke-WebRequest -Uri $downloadUrl -OutFile "actions-runner.zip"

# Verify the hash (optional but recommended)
# Get the expected hash from the GitHub releases page
$expectedHash = "<hash-from-github-releases-page>"
$actualHash = (Get-FileHash -Path "actions-runner.zip" -Algorithm SHA256).Hash
if ($actualHash -eq $expectedHash) {
    Write-Host "Hash verified successfully" -ForegroundColor Green
} else {
    Write-Host "Hash mismatch! Downloaded file may be corrupted." -ForegroundColor Red
}

# Extract the runner
Add-Type -AssemblyName System.IO.Compression.FileSystem
[System.IO.Compression.ZipFile]::ExtractToDirectory("$PWD\actions-runner.zip", "$PWD")

# Clean up the zip file
Remove-Item "actions-runner.zip"
```

Register the Runner

Get Registration Token

Option 1: Via GitHub Web UI

1. Navigate to your repository: `https://github.com/{owner}/{repo}`
2. Go to **Settings** → **Actions** → **Runners**
3. Click **New self-hosted runner**
4. Select **Windows** and copy the token from the configuration command

For organization-level runners:

1. Navigate to your organization: `https://github.com/{org}`
2. Go to **Settings** → **Actions** → **Runners**
3. Click **New self-hosted runner**

Option 2: Via GitHub CLI

```powershell
# Install GitHub CLI if not present
winget install GitHub.cli

# Authenticate
gh auth login

# Get token for repository runner
$repoToken = gh api -X POST /repos/{owner}/{repo}/actions/runners/registration-token --jq '.token'

# Get token for organization runner
$orgToken = gh api -X POST /orgs/{org}/actions/runners/registration-token --jq '.token'
```

Option 3: Via REST API

```powershell
# Set your Personal Access Token (requires admin:org or repo scope)
$pat = "<your-personal-access-token>"
$headers = @{
    "Authorization" = "Bearer $pat"
    "Accept" = "application/vnd.github+json"
    "X-GitHub-Api-Version" = "2022-11-28"
}

# For repository runner
$repoOwner = "your-username"
$repoName = "your-repo"
$response = Invoke-RestMethod -Uri "https://api.github.com/repos/$repoOwner/$repoName/actions/runners/registration-toke
$token = $response.token

# For organization runner
$orgName = "your-org"
$response = Invoke-RestMethod -Uri "https://api.github.com/orgs/$orgName/actions/runners/registration-token" -Method PO
$token = $response.token
```

Configure the Runner

Interactive Configuration

```powershell
```

```
# Navigate to runner directory
Set-Location -Path "C:\actions-runner"

# Run configuration (will prompt for inputs)
.\config.cmd --url https://github.com/{owner}/{repo} --token <YOUR_TOKEN>
```

Non-Interactive Configuration (Recommended for Automation)

```
powershell

# Set variables
$githubUrl = "https://github.com/{owner}/{repo}"  # Or https://github.com/{org} for org runner
$token = "<YOUR_REGISTRATION_TOKEN>"
$runnerName = "azure-win2025-hyperv-01"
$runnerLabels = "self-hosted,windows,azure,hyperv,failover-cluster,windows-2025"
$workFolder = "_work"

# Configure the runner
.\config.cmd `
    --url $githubUrl `
    --token $token `
    --name $runnerName `
    --labels $runnerLabels `
    --work $workFolder `
    --replace `
    --unattended
```

Configuration Options Reference

Option	Description
--url	GitHub repository or organization URL
--token	Registration token
--name	Runner name (must be unique)
--labels	Comma-separated custom labels
--work	Work directory for job execution
--replace	Replace existing runner with same name
--unattended	Run without prompts
--runasservice	Configure to run as Windows service
--windowslogonaccount	Service account for Windows service
--windowslogonpassword	Password for service account
--ephemeral	Runner exits after one job (for clean environments)
--disableupdate	Disable automatic runner updates

Install as Windows Service

Install the Service

powershell

```powershell
# Navigate to runner directory
Set-Location -Path "C:\actions-runner"

# Install as service (runs as Local System by default)
.\svc.cmd install

# Or install with a specific account
.\svc.cmd install "NT AUTHORITY\NETWORK SERVICE"

# Or install with a domain account
.\svc.cmd install "DOMAIN\svc-github-runner" "P@ssw0rd"
```

Manage the Service

powershell

```powershell
# Start the service
.\svc.cmd start

# Check service status
.\svc.cmd status

# Stop the service
.\svc.cmd stop

# Uninstall the service
.\svc.cmd uninstall
```

Alternative: Manage via Windows Services

```powershell
# Get the service name
$serviceName = Get-Service | Where-Object { $_.DisplayName -like "*actions*" } | Select-Object -ExpandProperty Name

# Start service
Start-Service -Name $serviceName

# Set to automatic startup
Set-Service -Name $serviceName -StartupType Automatic

# Check status
Get-Service -Name $serviceName
```

Configure Permissions for Hyper-V and Clustering

Hyper-V Permissions

The runner service account needs Hyper-V administrator permissions:

```powershell
```

```powershell
# Add service account to Hyper-V Administrators group
# For Local System (default)
Add-LocalGroupMember -Group "Hyper-V Administrators" -Member "NT AUTHORITY\SYSTEM"

# For Network Service
Add-LocalGroupMember -Group "Hyper-V Administrators" -Member "NT AUTHORITY\NETWORK SERVICE"

# For a domain account
Add-LocalGroupMember -Group "Hyper-V Administrators" -Member "DOMAIN\svc-github-runner"

# Verify membership
Get-LocalGroupMember -Group "Hyper-V Administrators"
```

Failover Cluster Permissions

```powershell
# Grant full cluster access to the service account
# For a domain account
Grant-ClusterAccess -User "DOMAIN\svc-github-runner" -Full

# For local accounts, you may need to configure cluster security
# Check current cluster permissions
Get-ClusterAccess

# Add local administrator account (if running as SYSTEM)
# The SYSTEM account typically has access if the computer is a cluster node
```

Windows Remote Management (WinRM) - If Needed

```powershell
# Enable WinRM for remote management
Enable-PSRemoting -Force

# Configure trusted hosts (if needed for cross-machine access)
Set-Item WSMan:\localhost\Client\TrustedHosts -Value "*" -Force
```

Network and Firewall Configuration

Required Outbound Connections

The GitHub Actions runner requires outbound HTTPS (port 443) access to:

Domain	Purpose
`github.com`	GitHub API and repository access
`api.github.com`	GitHub REST API
`*.actions.githubusercontent.com`	Action downloads and logs
`codeload.github.com`	Repository archive downloads
`*.pkg.github.com`	GitHub Packages
`ghcr.io`	GitHub Container Registry
`*.blob.core.windows.net`	Azure Blob storage (for some actions)

Configure Windows Firewall

```powershell
# Allow outbound HTTPS (usually allowed by default)
New-NetFirewallRule -DisplayName "GitHub Actions Runner - HTTPS Outbound" `
    -Direction Outbound `
    -Protocol TCP `
    -RemotePort 443 `
    -Action Allow `
    -Profile Any

# Allow the runner executable specifically
$runnerPath = "C:\actions-runner\bin\Runner.Listener.exe"
New-NetFirewallRule -DisplayName "GitHub Actions Runner Listener" `
    -Direction Outbound `
    -Protocol TCP `
    -Action Allow `
    -Program $runnerPath `
    -Profile Any

$workerPath = "C:\actions-runner\bin\Runner.Worker.exe"
New-NetFirewallRule -DisplayName "GitHub Actions Runner Worker" `
    -Direction Outbound `
    -Protocol TCP `
    -Action Allow `
    -Program $workerPath `
    -Profile Any
```

Azure Network Security Group (NSG) Rules

If you have restrictive NSG rules, ensure outbound HTTPS is allowed:

powershell

```powershell
# Add NSG rule via Azure CLI
az network nsg rule create `
  --resource-group $resourceGroup `
  --nsg-name "<your-nsg-name>" `
  --name "AllowGitHubOutbound" `
  --priority 100 `
  --direction Outbound `
  --access Allow `
  --protocol Tcp `
  --destination-port-ranges 443 `
  --destination-address-prefixes Internet
```

Using the Runner in Workflows

Basic Workflow Example

Create `.github/workflows/example.yml` in your repository:

yaml

```yaml
name: Build and Deploy on Self-Hosted Runner

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
  workflow_dispatch:

jobs:
  build:
    # Use your custom labels to target the runner
    runs-on: [self-hosted, windows, azure, hyperv]

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Display runner info
        run: |
          Write-Host "Runner Name: $env:RUNNER_NAME"
          Write-Host "Runner OS: $env:RUNNER_OS"
          Write-Host "Workspace: $env:GITHUB_WORKSPACE"
          Get-ComputerInfo | Select-Object WindowsProductName, OsArchitecture

      - name: Verify Hyper-V
        run: |
          Get-WindowsFeature -Name Hyper-V
          Get-Service vmms, vmcompute
          Get-VMSwitch

      - name: Verify Failover Clustering
        run: |
          Get-WindowsFeature -Name Failover-Clustering
          Get-Cluster
          Get-ClusterNode
```

Hyper-V VM Deployment Workflow

```yaml
```

```yaml
name: Deploy VM to Hyper-V

on:
  workflow_dispatch:
    inputs:
      vm_name:
        description: 'Name of the VM to create'
        required: true
        default: 'TestVM'
      memory_gb:
        description: 'Memory in GB'
        required: true
        default: '4'

jobs:
  deploy-vm:
    runs-on: [self-hosted, windows, hyperv]

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Create VM
        run: |
          $vmName = "${{ github.event.inputs.vm_name }}"
          $memoryGB = [int]"${{ github.event.inputs.memory_gb }}"
          $memoryBytes = $memoryGB * 1GB

          # Check if VM already exists
          if (Get-VM -Name $vmName -ErrorAction SilentlyContinue) {
            Write-Host "VM $vmName already exists, removing..."
            Stop-VM -Name $vmName -Force -ErrorAction SilentlyContinue
            Remove-VM -Name $vmName -Force
          }

          # Create new VM
          New-VM -Name $vmName `
            -MemoryStartupBytes $memoryBytes `
            -Generation 2 `
            -NewVHDPath "C:\Hyper-V\Virtual Hard Disks\$vmName.vhdx" `
            -NewVHDSizeBytes 50GB

          Write-Host "VM $vmName created successfully"
```

```
    Get-VM -Name $vmName

- name: Configure VM
  run: |
    $vmName = "${{ github.event.inputs.vm_name }}"

    # Set processor count
    Set-VMProcessor -VMName $vmName -Count 2

    # Enable nested virtualization (if needed)
    Set-VMProcessor -VMName $vmName -ExposeVirtualizationExtensions $true

    # Configure network
    $switch = Get-VMSwitch | Select-Object -First 1
    if ($switch) {
      Add-VMNetworkAdapter -VMName $vmName -SwitchName $switch.Name
    }

    Get-VM -Name $vmName | Format-List *
```

Failover Cluster Workflow

yaml

```yaml
name: Cluster Operations

on:
  workflow_dispatch:
    inputs:
      operation:
        description: 'Cluster operation to perform'
        required: true
        type: choice
        options:
          - status
          - failover
          - validate

jobs:
  cluster-ops:
    runs-on: [self-hosted, windows, failover-cluster]

    steps:
      - name: Check Cluster Status
        if: ${{ github.event.inputs.operation == 'status' }}
        run: |
          Write-Host "=== Cluster Information ==="
          Get-Cluster | Format-List *

          Write-Host "`n=== Cluster Nodes ==="
          Get-ClusterNode | Format-Table Name, State, NodeWeight

          Write-Host "`n=== Cluster Resources ==="
          Get-ClusterResource | Format-Table Name, State, ResourceType, OwnerGroup

          Write-Host "`n=== Cluster Groups ==="
          Get-ClusterGroup | Format-Table Name, State, OwnerNode

      - name: Validate Cluster
        if: ${{ github.event.inputs.operation == 'validate' }}
        run: |
          # Run cluster validation
          Test-Cluster -Node $env:COMPUTERNAME -Include "Inventory","Network","System Configuration"
```

Using Runner Groups (Organization Level)

For organization-level runners, you can create runner groups:

```yaml
jobs:
  production-deploy:
    runs-on:
      group: production-runners
      labels: [windows, hyperv]
    steps:
      - run: echo "Running on production runner group"
```

Maintenance and Troubleshooting

View Runner Logs

```powershell
# Runner logs location
$logPath = "C:\actions-runner\_diag"

# View recent runner logs
Get-ChildItem $logPath -Filter "Runner_*.log" |
    Sort-Object LastWriteTime -Descending |
    Select-Object -First 1 |
    Get-Content -Tail 100

# View worker logs
Get-ChildItem $logPath -Filter "Worker_*.log" |
    Sort-Object LastWriteTime -Descending |
    Select-Object -First 1 |
    Get-Content -Tail 100
```

Common Issues and Solutions

Runner Not Connecting

```powershell
```

```powershell
# Test connectivity to GitHub
Test-NetConnection -ComputerName github.com -Port 443
Test-NetConnection -ComputerName api.github.com -Port 443

# Check DNS resolution
Resolve-DnsName github.com
Resolve-DnsName api.github.com

# Check if proxy is required
[System.Net.WebRequest]::GetSystemWebProxy()
```

Service Won't Start

```powershell
# Check Windows Event Log
Get-EventLog -LogName Application -Source "*actions*" -Newest 20

# Verify service account permissions
$serviceName = (Get-Service | Where-Object { $_.DisplayName -like "*actions*" }).Name
sc.exe qc $serviceName

# Reset and reconfigure
Set-Location C:\actions-runner
.\svc.cmd stop
.\svc.cmd uninstall
.\svc.cmd install
.\svc.cmd start
```

Hyper-V Access Denied

```powershell
# Check Hyper-V Administrators group membership
Get-LocalGroupMember -Group "Hyper-V Administrators"

# Check effective permissions
whoami /groups

# Restart the service after permission changes
Restart-Service -Name $serviceName
```

Update the Runner

```powershell
# The runner auto-updates by default
# To manually update:

Set-Location C:\actions-runner

# Stop the service
.\svc.cmd stop

# Download new version
$newVersion = "2.322.0"  # Check GitHub releases for latest
Invoke-WebRequest -Uri "https://github.com/actions/runner/releases/download/v$newVersion/actions-runner-win-x64-$newV

# Backup current runner
Copy-Item -Path "C:\actions-runner" -Destination "C:\actions-runner-backup" -Recurse

# Extract update (excluding config)
# Note: Be careful not to overwrite .credentials and .runner files

# Start service
.\svc.cmd start
```

Remove/Unregister the Runner

```powershell
Set-Location C:\actions-runner

# Stop and uninstall the service
.\svc.cmd stop
.\svc.cmd uninstall

# Get removal token from GitHub
# Via GitHub CLI:
$removeToken = gh api -X POST /repos/{owner}/{repo}/actions/runners/remove-token --jq '.token'

# Unregister the runner
.\config.cmd remove --token $removeToken
```

Security Best Practices

Runner Security Recommendations

1. **Use dedicated service accounts**

```powershell
# Create a dedicated local account
$password = ConvertTo-SecureString "ComplexP@ssw0rd!" -AsPlainText -Force
New-LocalUser -Name "svc-github-runner" -Password $password -PasswordNeverExpires
Add-LocalGroupMember -Group "Administrators" -Member "svc-github-runner"
```

2. **Limit repository access** - Use runner groups to restrict which repositories can use the runner

3. **Use ephemeral runners for sensitive workloads**

```powershell
.\config.cmd --ephemeral --url ... --token ...
```

4. **Enable audit logging**

```powershell
# Enable PowerShell script block logging
$basePath = 'HKLM:\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging'
if (-not (Test-Path $basePath)) {
    New-Item -Path $basePath -Force
}
Set-ItemProperty -Path $basePath -Name EnableScriptBlockLogging -Value 1
```

5. **Keep the runner updated** - Enable automatic updates (default behavior)

6. **Use secrets management** - Never hardcode secrets in workflows

```yaml
- name: Use secret
  env:
    API_KEY: ${{ secrets.API_KEY }}
  run: |
    # Secret is available as environment variable
```

7. **Network isolation** - Place runners in a dedicated subnet with appropriate NSG rules

Credential Management

```powershell
# Use Windows Credential Manager for stored credentials
# Install the CredentialManager module
Install-Module -Name CredentialManager -Force

# Store credentials securely
New-StoredCredential -Target "GitHubRunner" -UserName "svc-account" -Password "password" -Persist LocalMachine

# Retrieve in scripts
$cred = Get-StoredCredential -Target "GitHubRunner"
```

Appendix: Quick Reference Commands

Registration Commands

```powershell
# Configure repository runner
.\config.cmd --url https://github.com/{owner}/{repo} --token <TOKEN>

# Configure organization runner
.\config.cmd --url https://github.com/{org} --token <TOKEN>

# Full configuration with all options
.\config.cmd `
  --url https://github.com/{owner}/{repo} `
  --token <TOKEN> `
  --name "runner-name" `
  --labels "label1,label2" `
  --work "_work" `
  --replace `
  --unattended
```

Service Commands

```powershell
```

```
.\svc.cmd install              # Install service
.\svc.cmd install "DOMAIN\user" "password"  # Install with specific account
.\svc.cmd start                # Start service
.\svc.cmd stop                 # Stop service
.\svc.cmd status               # Check status
.\svc.cmd uninstall            # Remove service
```

Useful PowerShell Commands

```
powershell

# Check runner status
Get-Service | Where-Object { $_.DisplayName -like "*actions*" }

# View recent logs
Get-Content "C:\actions-runner\_diag\Runner_*.log" -Tail 50

# Test GitHub connectivity
Test-NetConnection github.com -Port 443

# List Hyper-V VMs
Get-VM

# Check cluster status
Get-Cluster
Get-ClusterNode
```

Document Information

- **Version:** 1.0
- **Last Updated:** 2025
- **Compatibility:** Windows Server 2025, GitHub Actions Runner 2.321.0+
- **Azure VM Requirements:** Dv3/Dv4/Dv5 or Ev3/Ev4/Ev5 series (nested virtualization support)

Additional Resources

- GitHub Actions Self-Hosted Runners Documentation
- GitHub Actions Runner Releases
- Azure Nested Virtualization Documentation

- Windows Server Hyper-V Documentation
- Failover Clustering Documentation