Phase 5: Image Automation with Packer - Windows 10 Golden Image

🎯 Overview

This documentation provides a comprehensive guide for building automated Windows 10 golden images using HashiCorp Packer in Microsoft Azure. The solution creates standardized, pre-configured VM images with essential applications and security updates, enabling rapid deployment of consistent Windows environments.

📋 Table of Contents

🔧 Prerequisites

System Requirements

- **Operating System**: Windows 10/11 with WSL2 or Linux distribution
- **Memory**: Minimum 8GB RAM (16GB recommended)
- **Storage**: 50GB free disk space

- **Network**: Stable internet connection for Azure operations

Required Tools & Services

Tool	Version	Purpose
HashiCorp Packer	Latest	Image building automation
Azure CLI	2.0+	Azure resource management
PowerShell	5.1+	Windows configuration scripts
WSL2 Debian	Latest	Linux subsystem for Windows
Azure Subscription	Active	Cloud infrastructure

Required Permissions

- **Azure Subscription**: Contributor role
- **Resource Group**: Full access to create/manage resources
- **Compute**: VM creation and management permissions
- **Storage**: Disk and image management access

🚀 Environment Setup

Step 1: Install Packer on WSL Debian

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install required dependencies
sudo apt install -y wget unzip curl gnupg lsb-release

# Add HashiCorp GPG key and repository
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrin
gs/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(gr
ep -oP '(?<=UBUNTU_CODENAME=).*' /etc/os-release || lsb_release -cs)
main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

# Install Packer
sudo apt-get update && sudo apt-get install packer
```

```
# Verify installation
packer version
```

```
Preparing to unpack .../packer_1.13.0-1_amd64.deb ...
Unpacking packer (1.13.0-1) ...
Setting up packer (1.13.0-1) ...
Scanning processes...

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
suhaib@IND-147:~$ packer version
Packer v1.13.0
suhaib@IND-147:~$
```

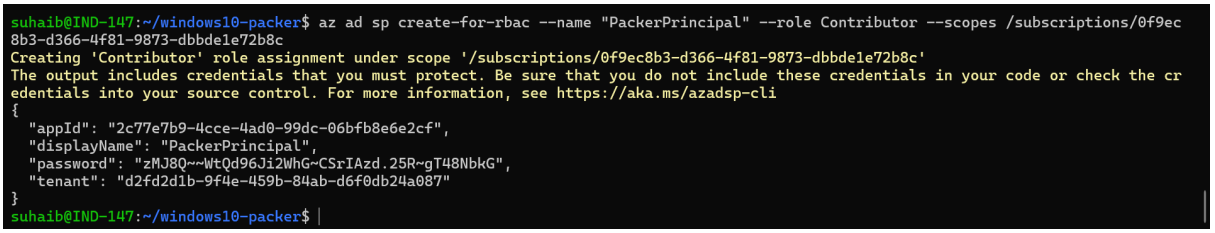
Step 2: Install Azure CLI

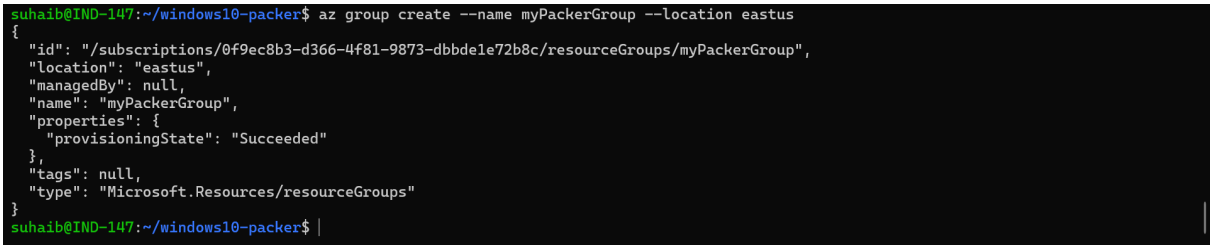
```
# Install Azure CLI
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

# Login to Azure
az login

# Verify login and list subscriptions
az account list --output table
```

```
suhaib@IND-147:~$ az login
A web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in
the web browser. If no web browser is available or if the web browser fails to open, use device code flow with `az login --use-device
-code`.

Retrieving tenants and subscriptions for the selection...

[Tenant and subscription selection]

No     Subscription name    Subscription ID                       Tenant
-----  -------------------  ------------------------------------  -----------------
[1] *  Azure for Students   0f9ec8b3-d366-4f81-9873-dbbde1e72b8c  Default Directory

The default is marked with an *; the default tenant is 'Default Directory' and subscription is 'Azure for Students' (0f9ec8b3-d366-4f
81-9873-dbbde1e72b8c).

Select a subscription and tenant (Type a number or Enter for no changes): 1

Tenant: Default Directory
Subscription: Azure for Students (0f9ec8b3-d366-4f81-9873-dbbde1e72b8c)

[Announcements]
With the new Azure CLI login experience, you can select the subscription you want to use more easily. Learn more about it and its con
figuration at https://go.microsoft.com/fwlink/?linkid=2271236

If you encounter any problem, please open an issue at https://aka.ms/azclibug

[Warning] The login output has been updated. Please be aware that it no longer displays the full list of available subscriptions by d
efault.

suhaib@IND-147:~$
```

Step 3: Create Azure Service Principal

```
# Create service principal for Packer
az ad sp create-for-rbac --name "PackerPrincipal" --role Contributor --sco
pe /subscriptions/YOUR_SUBSCRIPTION_ID

# Create resource group
az group create --name myPackerGroup --location eastus
```





Expected Output:

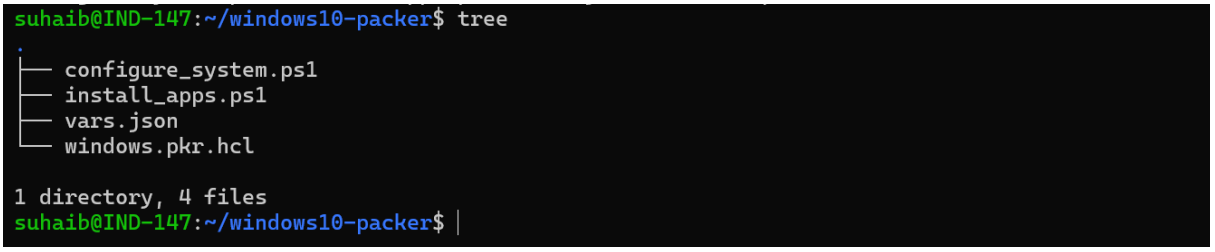
```
{
  "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "displayName": "PackerPrincipal",
  "password": "your-generated-password",
  "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
```

> ⚠️ Important: Save these credentials securely - they'll be needed for the Packer configuration.

📁 Project Structure

Create the following directory structure for your Packer project:

```
windows10-packer/
├── README.md
├── windows.pkr.hcl       # Main Packer template
├── vars.json             # Sensitive variables (add to .gitignore)
├── install-apps.ps1      # Application installation script
└── configure-system.ps1  # System configuration script
```



Create Project Directory

```
mkdir -p windows10-packer/
cd windows10-packer
```

⚙️ Configuration Files

Main Packer Template (windows.pkr.hcl)

```
# Packer configuration for Windows 10 golden image in Azure
packer {
  required_plugins {
    azure = {
      version = ">= 1.0.0"
      source  = "github.com/hashicorp/azure"
    }
  }
}

variable "client_id" {
  type      = string
  sensitive = true
}
```

```
variable "client_secret" {
  type      = string
  sensitive = true
}

variable "tenant_id" {
  type      = string
  sensitive = true
}

variable "subscription_id" {
  type      = string
  sensitive = true
}

variable "resource_group" {
  type    = string
  default = "myPackerGroup"
}

variable "location" {
  type    = string
  default = "eastus"
}

locals {
  timestamp = regex_replace(timestamp(), "[: TZ-]", "")
}

source "azure-arm" "windows-10" {
  client_id                       = var.client_id
  client_secret                   = var.client_secret
  tenant_id                       = var.tenant_id
  subscription_id                 = var.subscription_id
  managed_image_resource_group_name = var.resource_group
  managed_image_name              = "windows-10-golden-${local.timestam
p}"
```

```
  location                  = var.location
  vm_size                    = "Standard_D2s_v3"
  os_type                   = "Windows"
  image_publisher                = "MicrosoftWindowsDesktop"
  image_offer                = "Windows-10"
  image_sku                  = "20h2-ent"
  communicator                = "winrm"
  winrm_use_ssl               = true
  winrm_insecure               = true
  winrm_timeout               = "30m"
  winrm_username                = "packer"
  winrm_password                = "SuperS3cr3t!!!!"
}

build {
  name    = "windows-10-golden"
  sources = ["source.azure-arm.windows-10"]

  # Configure WinRM and install PSWindowsUpdate
  provisioner "powershell" {
    inline = [
      "Write-Host 'Configuring WinRM...'",
      "winrm quickconfig -q",
      "winrm set winrm/config/winrs '@{MaxMemoryPerShellMB=\"1024\"}'",
      "winrm set winrm/config '@{MaxTimeoutms=\"1800000\"}'",
      "winrm set winrm/config/service '@{AllowUnencrypted=\"true\"}'",
      "winrm set winrm/config/service/auth '@{Basic=\"true\"}'",
      "netsh advfirewall firewall add rule name=\"WinRM 5985\" protocol=TC
P dir=in localport=5985 action=allow",
      "net user packer SuperS3cr3t!!!! /add /y",
      "net localgroup administrators packer /add",
      "Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -For
ce",
      "Install-Module -Name PSWindowsUpdate -Force"
    ]
  }

  # Install Windows Updates
```

```
  provisioner "powershell" {
    inline = [
      "Write-Host 'Installing Windows Updates...'",
      "$ErrorActionPreference = 'Stop'",
      "Install-WindowsUpdate -AcceptAll -AutoReboot"
    ]
  }

  # Restart Windows after updates
  provisioner "windows-restart" {
    restart_timeout = "15m"
  }

  # Install sample applications (Notepad++, 7Zip, and Chrome)
  provisioner "powershell" {
    script = "./install_apps.ps1"
  }

  # Configure system settings
  provisioner "powershell" {
    script = "./configure_system.ps1"
  }

  # Final cleanup and sysprep preparation
  provisioner "powershell" {
    inline = [
      "Write-Host 'Final cleanup and preparation for sysprep...'",
      "# Stop Windows Update service",
      "Stop-Service -Name 'wuauserv' -Force -ErrorAction SilentlyContinue",
      "Set-Service -Name 'wuauserv' -StartupType Disabled",
      "# Clear temp files",
      "Get-ChildItem -Path 'C:\\Windows\\Temp' -Recurse -ErrorAction Silentl
yContinue | Remove-Item -Force -Recurse -ErrorAction SilentlyContinue",
      "Get-ChildItem -Path 'C:\\Users\\*\\AppData\\Local\\Temp' -Recurse -Er
rorAction SilentlyContinue | Remove-Item -Force -Recurse -ErrorAction Sile
ntlyContinue",
      "# Clear event logs",
      "Get-EventLog -LogName * | ForEach-Object { Clear-EventLog -LogNa
```

```
me $_.Log -ErrorAction SilentlyContinue }",
      "# Remove packer user (optional)",
      "# net user packer /delete",
      "Write-Host 'Cleanup completed. Ready for sysprep.'"
    ]
  }

  # Sysprep (correct and blocking)
  provisioner "powershell" {
    inline = [
      "C:\\Windows\\System32\\Sysprep\\Sysprep.exe /oobe /generalize /shu
tdown /quiet"
    ]
  }
}
```

Variables File (vars.json)

```
{
  "client_id": "your-service-principal-app-id",
  "client_secret": "your-service-principal-password",
  "tenant_id": "your-azure-tenant-id",
  "subscription_id": "your-azure-subscription-id"
}
```

> ⚠️ Security Note: Add vars.json to your .gitignore file to prevent credential exposure.

🔨 PowerShell Scripts

Application Installation Script (install-apps.ps1)

```
# Install sample applications
$ErrorActionPreference = 'Stop'
Write-Host 'Installing sample applications...'
```

```
try {
    # Create temp directory
    New-Item -ItemType Directory -Path 'C:\Temp' -Force

    # Install Chocolatey
    Set-ExecutionPolicy Bypass -Scope Process -Force
    [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.Ser
vicePointManager]::SecurityProtocol -bor 3072
    Invoke-Expression ((New-Object System.Net.WebClient).DownloadString
('https://community.chocolatey.org/install.ps1'))

    # Install Notepad++, 7Zip, and Chrome
    choco install notepadplusplus 7zip googlechrome -y

    Write-Host 'Applications installed successfully'
}
catch {
    Write-Host "Error during application installation: $($_.Exception.Messag
e)"
    throw
}
finally {
    # Clean up
    Remove-Item -Path 'C:\Temp' -Recurse -Force -ErrorAction SilentlyConti
nue
}
```

System Configuration Script (configure-system.ps1)

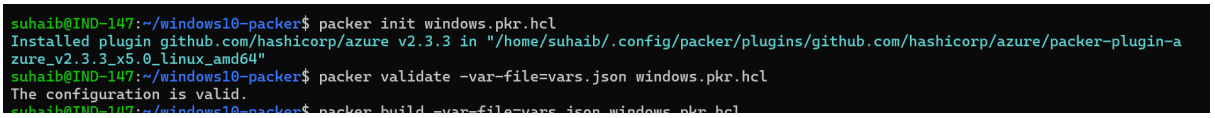
```
# Configure system settings
$ErrorActionPreference = 'Stop'
Write-Host 'Configuring system settings...'

try {
    # Set timezone
    Set-TimeZone -Id 'Eastern Standard Time'
    Write-Host 'Timezone set to Eastern Standard Time'
```

```
    # Disable Windows Defender real-time monitoring (optional, for performa
nce)
    try {
        Set-MpPreference -DisableRealtimeMonitoring $true
        Write-Host 'Windows Defender real-time monitoring disabled'
    }
    catch {
        Write-Host 'Could not disable Windows Defender - continuing...'
    }

    # Set power plan to High Performance
    try {
        powercfg /setactive SCHEME_MIN
        Write-Host 'Power plan set to High Performance'
    }
    catch {
        Write-Host 'Could not set power plan - continuing...'
    }

    # Disable UAC (optional, for automation)
    try {
        Set-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\Curre
ntVersion\Policies\System' -Name 'EnableLUA' -Value 0
        Write-Host 'UAC disabled'
    }
    catch {
        Write-Host 'Could not disable UAC - continuing...'
    }

    Write-Host 'System configuration completed successfully'
}
catch {
    Write-Host "Error during system configuration: $($_.Exception.Messag
e)"
    throw
}
```

🚀 Execution Guide

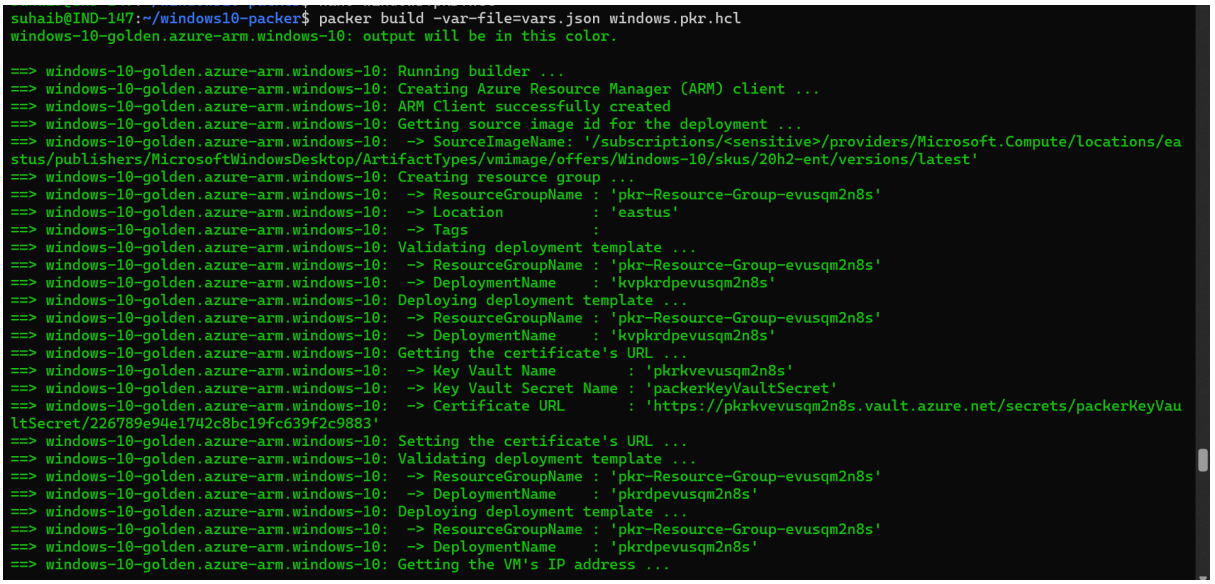
Step 1: Prepare Environment

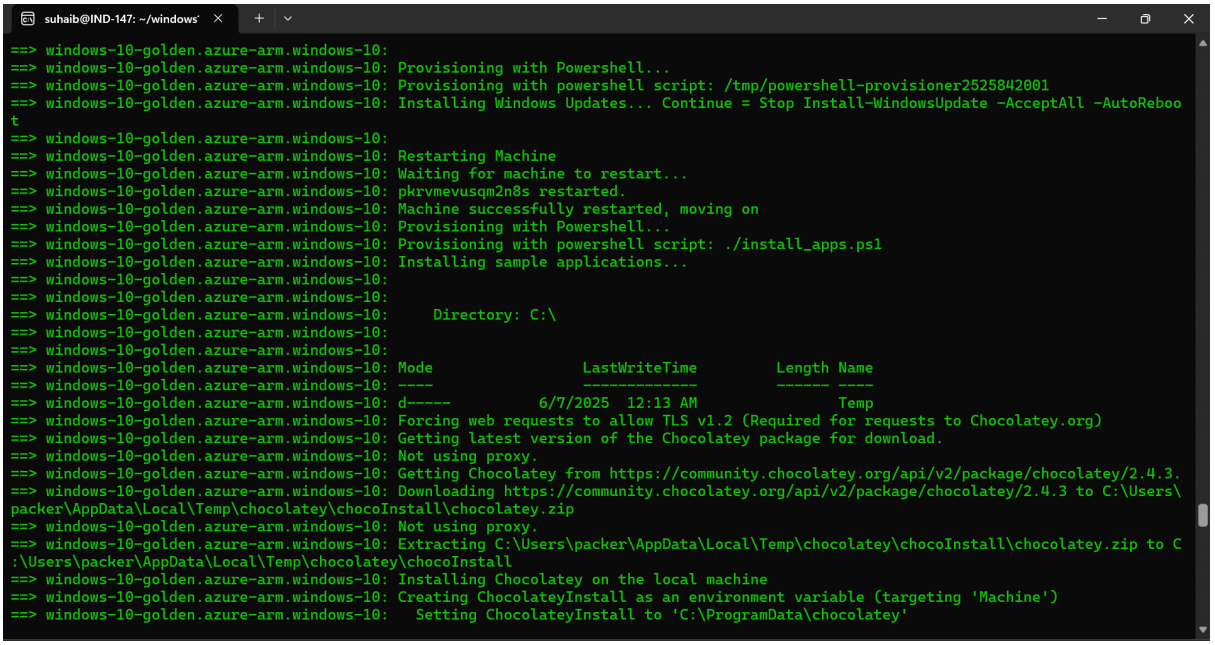
```
# Navigate to project directory
cd windows10-packer

# Initialize Packer (downloads required plugins)
packer init windows.pkr.hcl

# Validate the Packer template
packer validate -var-file="vars.json" windows.pkr.hcl
```

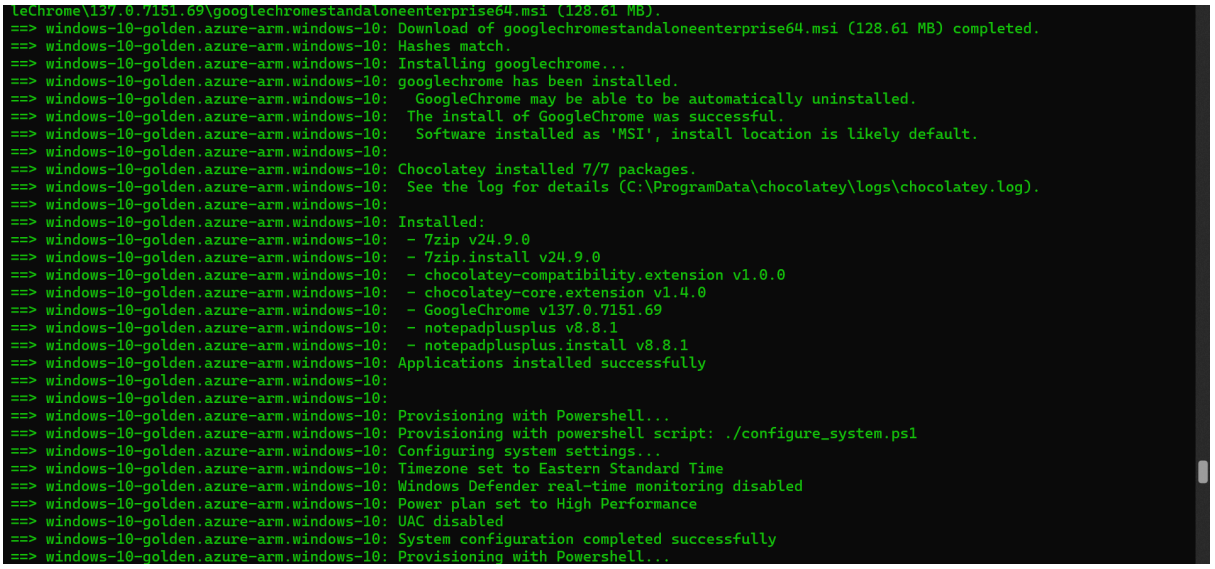


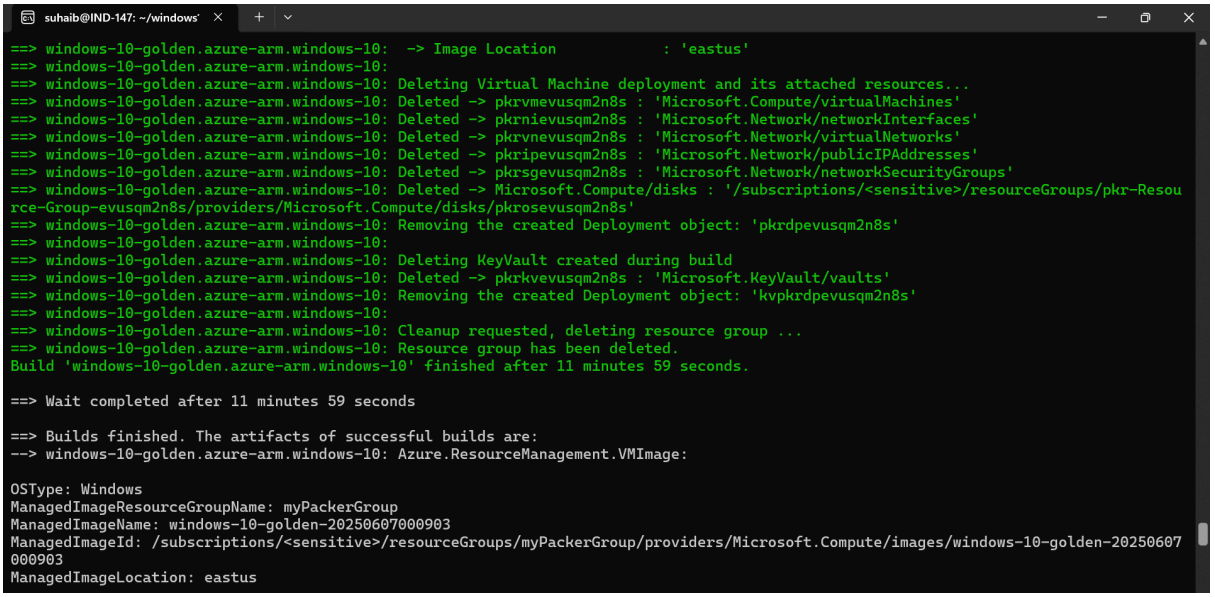
Step 2: Execute Image Build

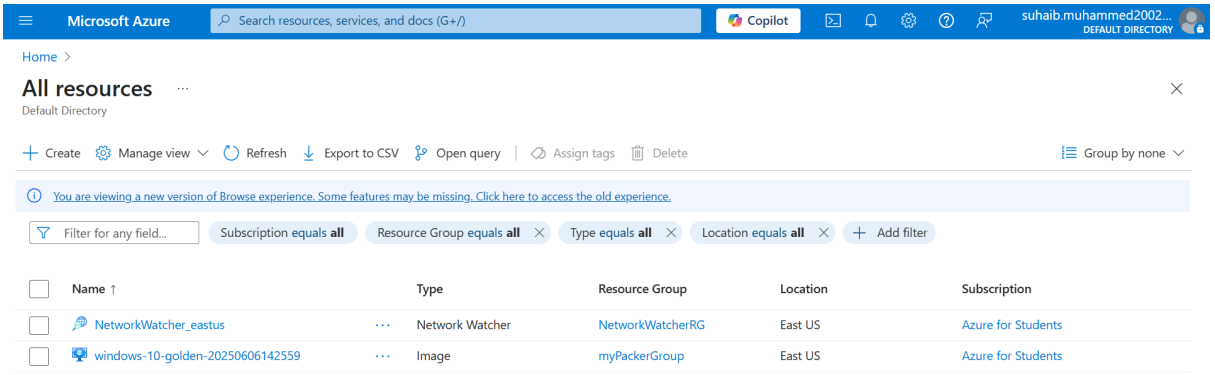
```
# Build the golden image
packer build -var-file="vars.json" windows.pkr.hcl
```

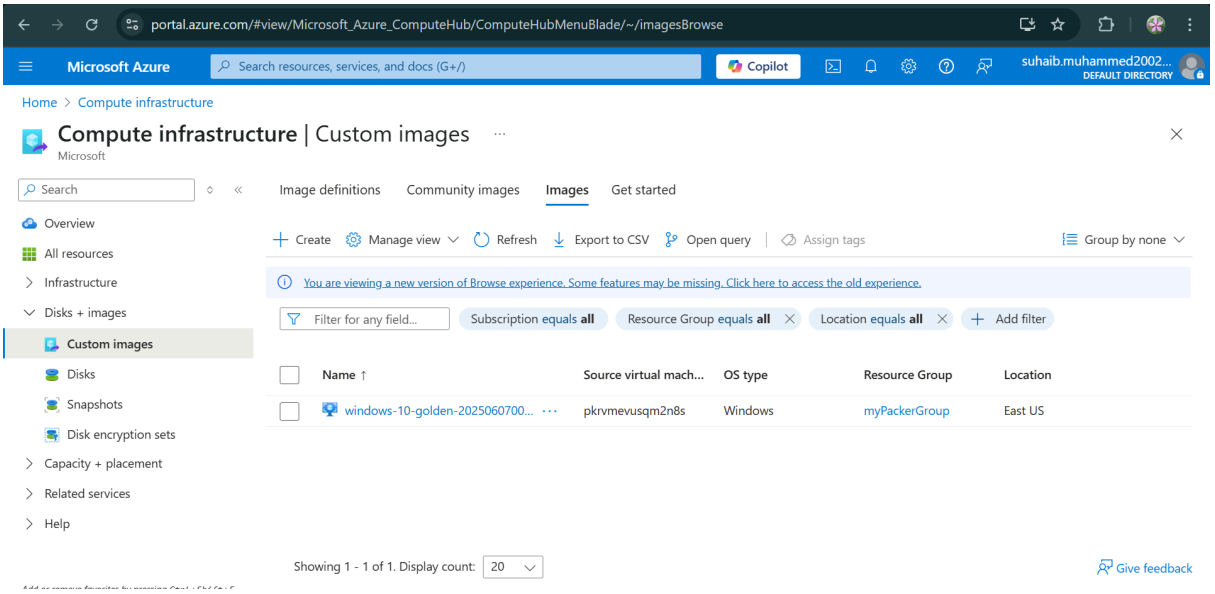



suhaib@IND-147: ~/windows
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10: Provisioning with Powershell...
==> windows-10-golden.azure-arm.windows-10: Provisioning with powershell script: /tmp/powershell-provisioner2525842001
==> windows-10-golden.azure-arm.windows-10: Installing Windows Updates... Continue = Stop Install-WindowsUpdate -AcceptAll -AutoReboo
t
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10: Restarting Machine
==> windows-10-golden.azure-arm.windows-10: Waiting for machine to restart...
==> windows-10-golden.azure-arm.windows-10: pkrvmevusqm2n8s restarted.
==> windows-10-golden.azure-arm.windows-10: Machine successfully restarted, moving on
==> windows-10-golden.azure-arm.windows-10: Provisioning with Powershell...
==> windows-10-golden.azure-arm.windows-10: Provisioning with powershell script: ./install_apps.ps1
==> windows-10-golden.azure-arm.windows-10: Installing sample applications...
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10:   Directory: C:\
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10: Mode   LastWriteTime   Length Name
==> windows-10-golden.azure-arm.windows-10: ----   -------------   ------ ----
==> windows-10-golden.azure-arm.windows-10: d-----   6/7/2025 12:13 AM   Temp
==> windows-10-golden.azure-arm.windows-10: Forcing web requests to allow TLS v1.2 (Required for requests to Chocolatey.org)
==> windows-10-golden.azure-arm.windows-10: Getting latest version of the Chocolatey package for download.
==> windows-10-golden.azure-arm.windows-10: Not using proxy.
==> windows-10-golden.azure-arm.windows-10: Getting Chocolatey from https://community.chocolatey.org/api/v2/package/chocolatey/2.4.3.
==> windows-10-golden.azure-arm.windows-10: Downloading https://community.chocolatey.org/api/v2/package/chocolatey/2.4.3 to C:\Users\
packer\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip
==> windows-10-golden.azure-arm.windows-10: Not using proxy.
==> windows-10-golden.azure-arm.windows-10: Extracting C:\Users\packer\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip to C
:\Users\packer\AppData\Local\Temp\chocolatey\chocoInstall
==> windows-10-golden.azure-arm.windows-10: Installing Chocolatey on the local machine
==> windows-10-golden.azure-arm.windows-10: Creating ChocolateyInstall as an environment variable (targeting 'Machine')
==> windows-10-golden.azure-arm.windows-10:   Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'


leChrome\137.0.7151.69\googlechromestandaloneenterprise64.msi (128.61 MB).
==> windows-10-golden.azure-arm.windows-10: Download of googlechromestandaloneenterprise64.msi (128.61 MB) completed.
==> windows-10-golden.azure-arm.windows-10: Hashes match.
==> windows-10-golden.azure-arm.windows-10: Installing googlechrome...
==> windows-10-golden.azure-arm.windows-10: googlechrome has been installed.
==> windows-10-golden.azure-arm.windows-10:   GoogleChrome may be able to be automatically uninstalled.
==> windows-10-golden.azure-arm.windows-10: The install of GoogleChrome was successful.
==> windows-10-golden.azure-arm.windows-10:   Software installed as 'MSI', install location is likely default.
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10: Chocolatey installed 7/7 packages.
==> windows-10-golden.azure-arm.windows-10: See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10: Installed:
==> windows-10-golden.azure-arm.windows-10: - 7zip v24.9.0
==> windows-10-golden.azure-arm.windows-10: - 7zip.install v24.9.0
==> windows-10-golden.azure-arm.windows-10: - chocolatey-compatibility.extension v1.0.0
==> windows-10-golden.azure-arm.windows-10: - chocolatey-core.extension v1.4.0
==> windows-10-golden.azure-arm.windows-10: - GoogleChrome v137.0.7151.69
==> windows-10-golden.azure-arm.windows-10: - notepadplusplus v8.8.1
==> windows-10-golden.azure-arm.windows-10: - notepadplusplus.install v8.8.1
==> windows-10-golden.azure-arm.windows-10: Applications installed successfully
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10:
==> windows-10-golden.azure-arm.windows-10: Provisioning with Powershell...
==> windows-10-golden.azure-arm.windows-10: Provisioning with powershell script: ./configure_system.ps1
==> windows-10-golden.azure-arm.windows-10: Configuring system settings...
==> windows-10-golden.azure-arm.windows-10: Timezone set to Eastern Standard Time
==> windows-10-golden.azure-arm.windows-10: Windows Defender real-time monitoring disabled
==> windows-10-golden.azure-arm.windows-10: Power plan set to High Performance
==> windows-10-golden.azure-arm.windows-10: UAC disabled
==> windows-10-golden.azure-arm.windows-10: System configuration completed successfully
==> windows-10-golden.azure-arm.windows-10: Provisioning with Powershell...







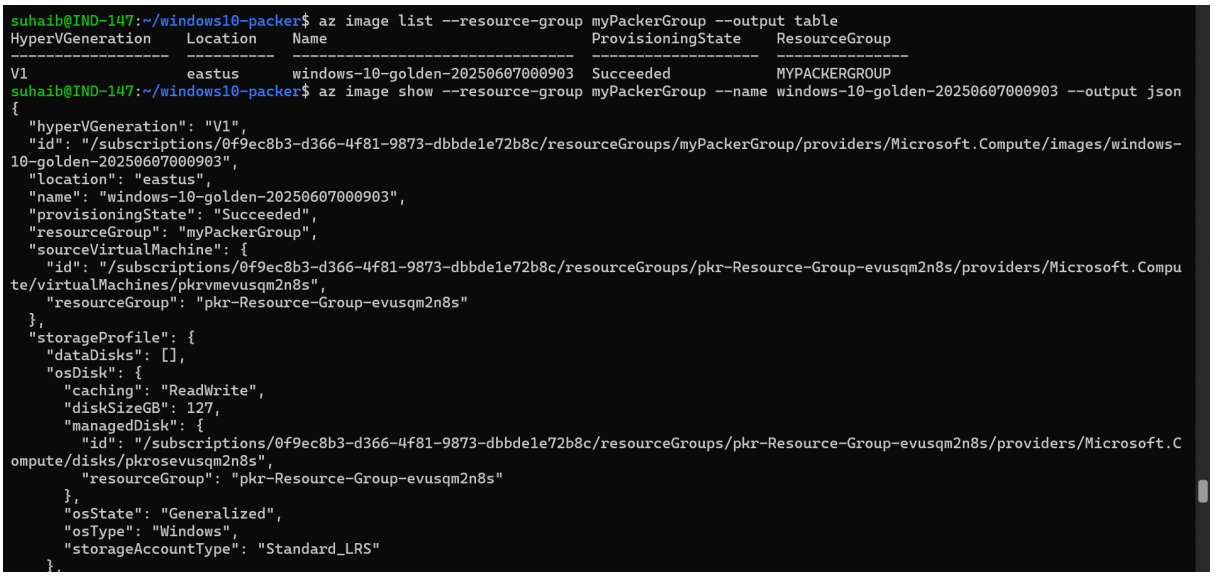
Expected Build Process Timeline

Phase	Duration	Description
VM Creation	5-10 min	Azure VM provisioning
WinRM Setup	2-5 min	Windows Remote Management configuration
Windows Updates	15-30 min	System updates installation
App Installation	10-15 min	Chocolatey and applications
System Config	5-10 min	Settings and optimizations
Cleanup & Sysprep	5-10 min	Final preparation
Total	**45-80 min**	Complete build process

✅ Image Verification

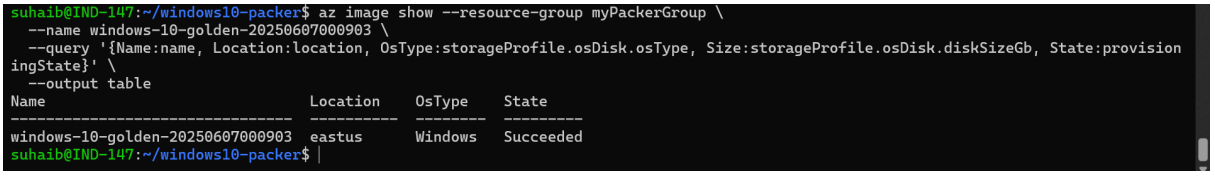
Step 1: Verify Built Image

```
# List all custom images in resource group
az image list --resource-group myPackerGroup --output table

# Get detailed image information
az image show --resource-group myPackerGroup \
  --name windows-10-golden-TIMESTAMP \
  --output json
```



Step 2: Check Image Properties

```
# Verify image specifications
az image show --resource-group myPackerGroup \
  --name windows-10-golden-20250607000903 \
  --query '{Name:name, Location:location, OsType:storageProfile.osDisk.os
Type, Size:storageProfile.osDisk.diskSizeGb, State:provisioningState}' \
  --output table
```



Expected Output

```
Name                          Location    OsType    Size    State
----------------------------- ----------- -------- ------ ---------
windows-10-golden-20250607... eastus      Windows   127    Succeeded
```

🚀 Deployment & Testing

Create Test VM from Golden Image

Step 1: Create Network Infrastructure

```
# Create virtual network
az network vnet create \
  --resource-group myPackerGroup \
  --name myVNet \
  --address-prefix 10.0.0.0/16 \
  --subnet-name mySubnet \
  --subnet-prefix 10.0.1.0/24

# Create network security group
az network nsg create \
  --resource-group myPackerGroup \
  --name myNetworkSecurityGroup
```

```
# Add RDP rule
az network nsg rule create \
  --resource-group myPackerGroup \
  --nsg-name myNetworkSecurityGroup \
  --name AllowRDP \
  --protocol tcp \
  --priority 1000 \
  --destination-port-range 3389 \
  --access allow

# Create public IP
az network public-ip create \
  --resource-group myPackerGroup \
  --name myPublicIP \
  --allocation-method Static

# Create network interface
az network nic create \
  --resource-group myPackerGroup \
  --name myNic \
  --vnet-name myVNet \
  --subnet mySubnet \
  --public-ip-address myPublicIP \
  --network-security-group myNetworkSecurityGroup
```

```
suhaib@IND-147:~/windows10-packer$ az network nsg create \
  --resource-group myPackerGroup \
  --name myNetworkSecurityGroup
{
  "NewNSG": {
    "defaultSecurityRules": [
      {
        "access": "Allow",
        "description": "Allow inbound traffic from all VMs in VNET",
        "destinationAddressPrefix": "VirtualNetwork",
        "destinationAddressPrefixes": [],
        "destinationPortRange": "*",
        "destinationPortRanges": [],
        "direction": "Inbound",
        "etag": "W/\"6f941e25-4e28-4563-9037-563068b839ae\"",
        "id": "/subscriptions/0f9ec8b3-d366-4f81-9873-dbbde1e72b8c/resourceGroups/myPackerGroup/providers/Microsoft.Network/networkSe
curityGroups/myNetworkSecurityGroup/defaultSecurityRules/AllowVnetInBound",
        "name": "AllowVnetInBound",
        "priority": 65000,
        "protocol": "*",
```

```
suhaib@IND-147: ~/windows
}
suhaib@IND-147:~/windows10-packer$ az network nsg rule create \
  --resource-group myPackerGroup \
  --nsg-name myNetworkSecurityGroup \
  --name myNetworkSecurityGroupRuleRDP \
  --protocol tcp \
  --priority 1000 \
  --destination-port-range 3389 \
  --access allow
{
  "access": "Allow",
  "destinationAddressPrefix": "*",
  "destinationAddressPrefixes": [],
  "destinationPortRange": "3389",
  "destinationPortRanges": [],
  "direction": "Inbound",
  "etag": "W/\"f962433b-3844-46c3-93ec-76002418fe4e\"",
  "id": "/subscriptions/0f9ec8b3-d366-4f81-9873-dbbde1e72b8c/resourceGroups/myPackerGroup/providers/Microsoft.Network/networkSecurity
Groups/myNetworkSecurityGroup/securityRules/myNetworkSecurityGroupRuleRDP",
  "name": "myNetworkSecurityGroupRuleRDP",
  "priority": 1000,
  "protocol": "Tcp",
  "provisioningState": "Succeeded",
  "resourceGroup": "myPackerGroup",
  "sourceAddressPrefix": "*",
  "sourceAddressPrefixes": [],
  "sourcePortRange": "*",
  "sourcePortRanges": [],
  "type": "Microsoft.Network/networkSecurityGroups/securityRules"
}
suhaib@IND-147:~/windows10-packer$ az network public-ip create \
  --resource-group myPackerGroup \
```

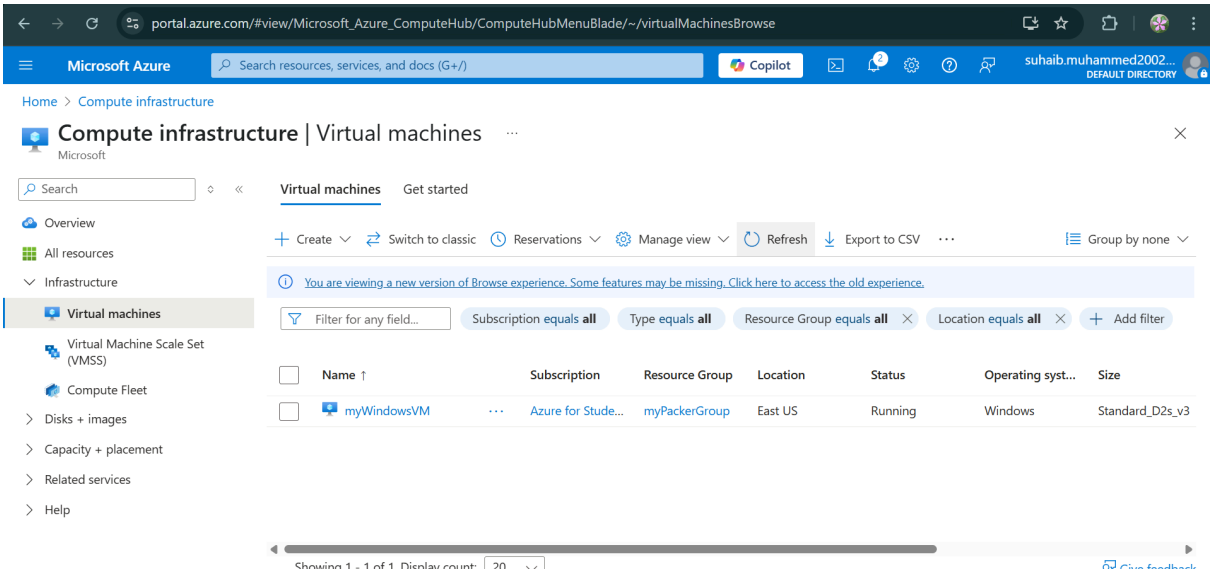
```
suhaib@IND-147:~/windows10-packer$ az network nic create \
  --resource-group myPackerGroup \
  --name myNic \
  --vnet-name myVNet \
  --subnet mySubnet \
  --public-ip-address myPublicIP \
  --network-security-group myNetworkSecurityGroup
{
  "NewNIC": {
    "auxiliaryMode": "None",
    "auxiliarySku": "None",
    "disableTcpStateTracking": false,
    "dnsSettings": {
      "appliedDnsServers": [],
      "dnsServers": [],
      "internalDomainNameSuffix": "ti2uohsp51uubjgtrhr5z2jmec.bx.internal.cloudapp.net"
    },
    "enableAcceleratedNetworking": false,
    "enableIPForwarding": false,
    "etag": "W/\"2cf0b04c-c770-4dde-997f-35e9227fa4c4\"",
    "hostedWorkloads": [],
    "id": "/subscriptions/0f9ec8b3-d366-4f81-9873-dbbde1e72b8c/resourceGroups/myPackerGroup/providers/Microsoft.Network/networkInterf
aces/myNic",
    "ipConfigurations": [
      {
        "etag": "W/\"2cf0b04c-c770-4dde-997f-35e9227fa4c4\"",
        "id": "/subscriptions/0f9ec8b3-d366-4f81-9873-dbbde1e72b8c/resourceGroups/myPackerGroup/providers/Microsoft.Network/networkIn
```

Step 2: Deploy VM from Golden Image

```
# Create VM from golden image
az vm create \
  --resource-group myPackerGroup \
```

```
  --name myWindowsVM \
  --image windows-10-golden-TIMESTAMP \
  --admin-username azureuser \
  --admin-password 'YourSecurePassword123!' \
  --nics myNic \
  --size Standard_D2s_v3 \
  --storage-sku Premium_LRS

# Get public IP for connection
az vm show \
  --resource-group myPackerGroup \
  --name myWindowsVM \
  --show-details \
  --query publicIps \
  --output tsv
```

```
suhaib@IND-147:~/windows10-packer$ az vm create \
  --resource-group myPackerGroup \
  --name myWindowsVM \
  --image windows-10-golden-20250607000903 \
  --admin-username azureuser \
  --admin-password 'YourSecurePassword123!' \
  --nics myNic \
  --size Standard_D2s_v3 \
  --storage-sku Premium_LRS
The default value of '--size' will be changed to 'Standard_D2s_v5' from 'Standard_DS1_v2' in a future release.
{
  "fqdns": "",
  "id": "/subscriptions/0f9ec8b3-d366-4f81-9873-dbbde1e72b8c/resourceGroups/myPackerGroup/providers/Microsoft.Compute/virtualMachines
/myWindowsVM",
  "location": "eastus",
  "macAddress": "00-22-48-1F-5D-69",
  "powerState": "VM running",
  "privateIpAddress": "10.0.1.4",
  "publicIpAddress": "52.190.22.150",
  "resourceGroup": "myPackerGroup",
  "zones": ""
}
```

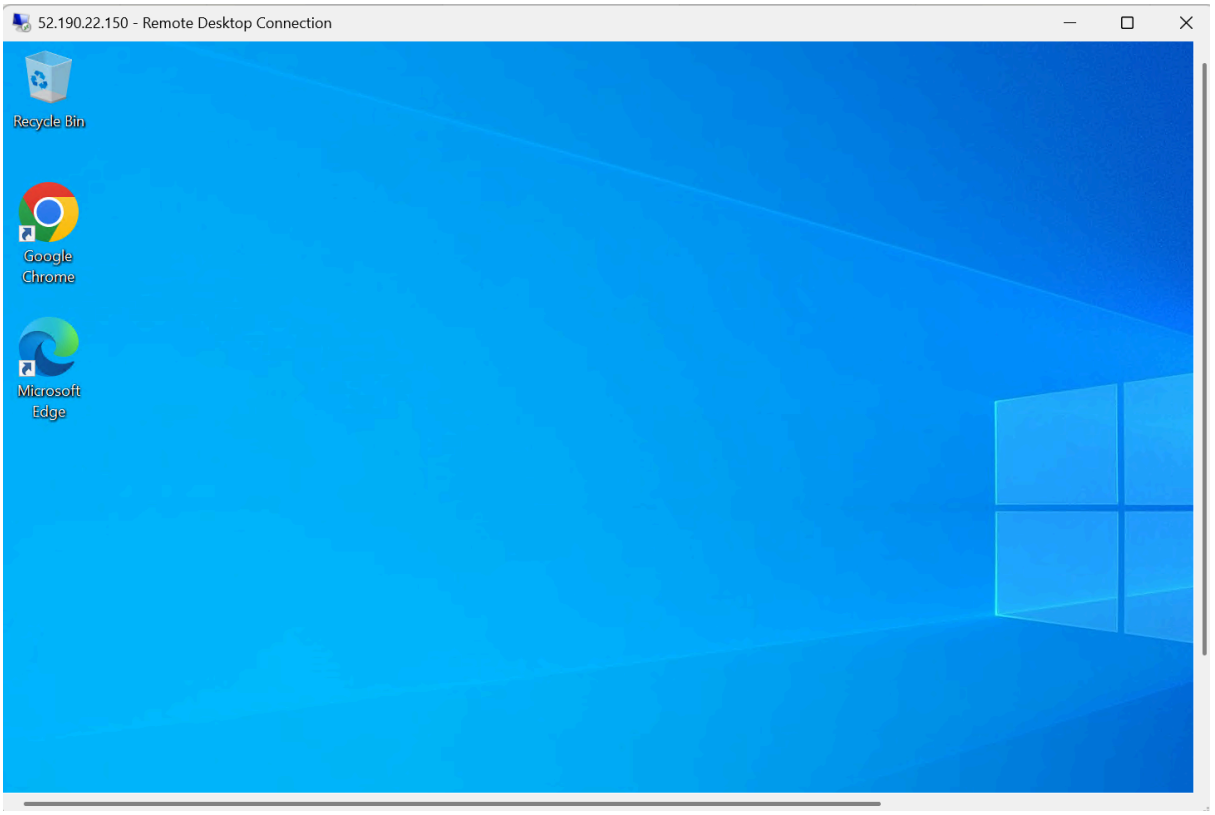


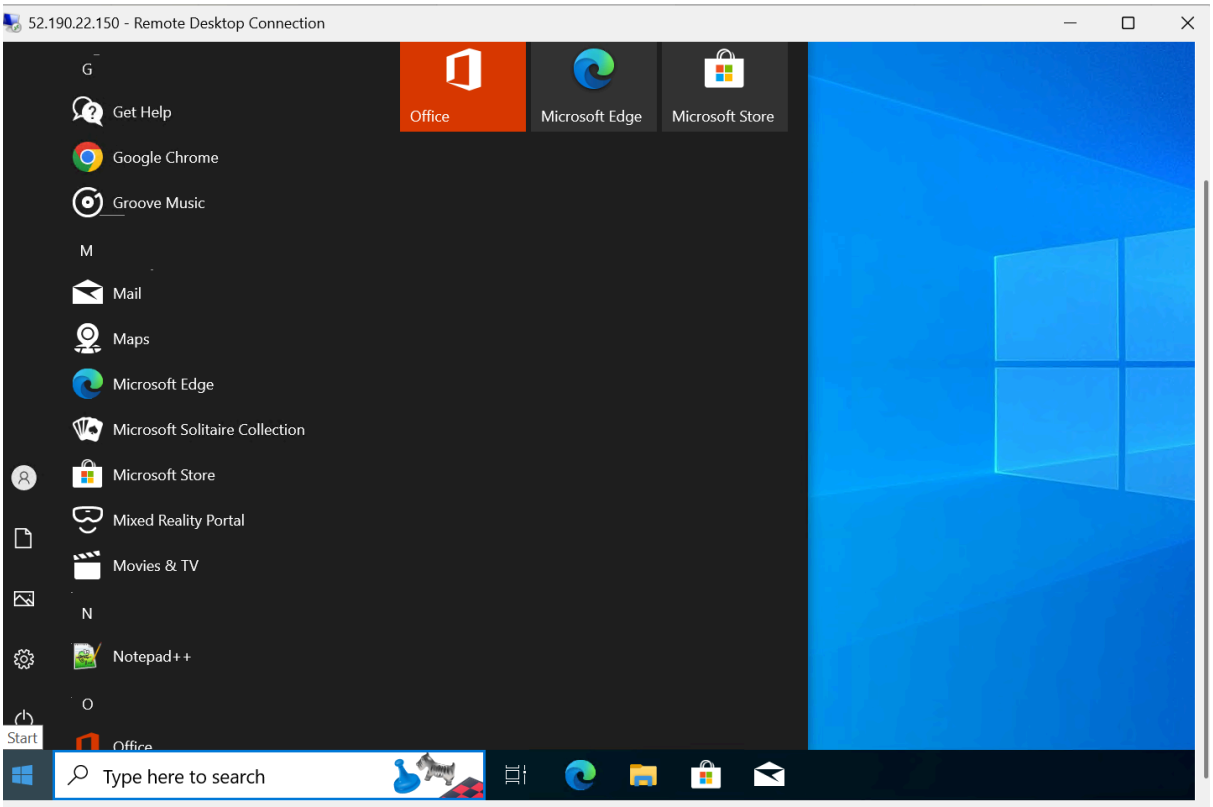
```
suhaib@IND-147:~/windows10-packer$ az vm show \
  --resource-group myPackerGroup \
  --name myWindowsVM \
  --show-details \
  --query publicIps \
  --output tsv
52.190.22.150
suhaib@IND-147:~/windows10-packer$
```

Connection Methods

Option 1: Windows RDP Client

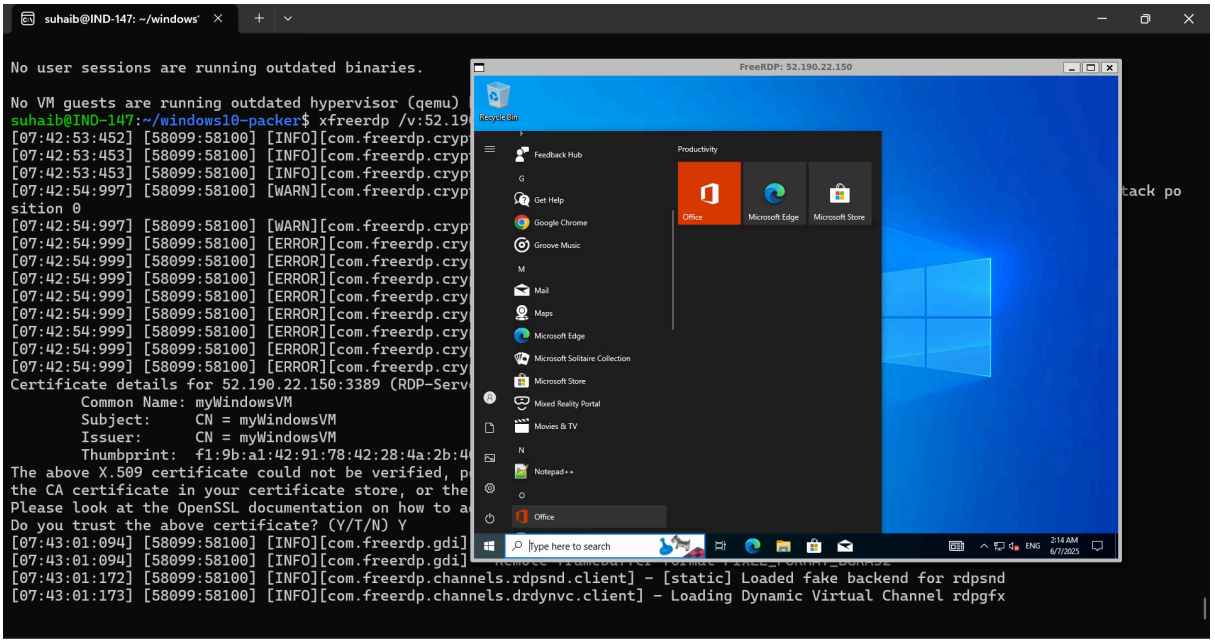
1. Open **Remote Desktop Connection**
2. Enter the public IP address
3. Username: `azureuser`
4. Password: `YourSecurePassword123!`





Option 2: Linux/WSL RDP Client

```
# Install FreeRDP
sudo apt install freerdp2-x11

# Connect to VM
xfreerdp /v:YOUR_PUBLIC_IP /u:azureuser /p:'YourSecurePassword123!' /si
ze:1920×1080
```



Verification Checklist

Once connected to your VM, verify the following components:

Check Installed Applications

- [ ] Notepad++ is installed
- [ ] 7-Zip is installed
- [ ] Google Chrome is installed
- [ ] Check Start Menu → All Apps

Check System Configuration

- [ ] Timezone is set to Eastern Standard Time
- [ ] Power plan is set to High Performance
- [ ] Windows Defender settings (if disabled)
- [ ] UAC settings (if disabled)

Check Windows Updates

- [ ] Go to Settings → Update & Security → Windows Update
- [ ] Verify updates are installed

🔧 Troubleshooting

Common Issues and Solutions

WinRM Connection Failures

Problem: Packer cannot connect via WinRM

```
Error: timeout waiting for WinRM connection
```

Solutions:

1. Verify firewall rules allow port 5985
2. Check WinRM service status:

   ```
   winrm get winrm/config
   ```

3. Ensure proper authentication settings
4. Increase `winrm_timeout` in template

Azure Authentication Errors

Problem: Service principal authentication fails

```
Error: azure-arm builder error: authentication failure
```

Solutions:

1. Verify service principal credentials in `vars.json`
2. Check service principal permissions:

   ```
   az role assignment list --assignee YOUR_CLIENT_ID
   ```

3. Ensure subscription ID is correct
4. Regenerate service principal if needed

PowerShell Script Execution Failures

Problem: Scripts fail with execution policy errors

```
Error: execution of scripts is disabled on this system
```

Solutions:

1. Add execution policy bypass to scripts:

   ```
   Set-ExecutionPolicy Bypass -Scope Process -Force
   ```

2. Use inline PowerShell commands instead of script files
3. Check script syntax and error handling

Chocolatey Installation Issues

Problem: Package installation fails

```
Error: The remote name could not be resolved: 'chocolatey.org'
```

Solutions:

1. Ensure internet connectivity during build
2. Set TLS 1.2 protocol:

   ```
   [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]::Tls12
   ```

3. Use alternative package sources
4. Install packages individually with error handling

Sysprep Failures

Problem: Sysprep process fails or hangs

```
Error: sysprep failed with exit code 1
```

Solutions:

1. Check sysprep logs: `C:\Windows\System32\Sysprep\Panther\`
2. Remove user profiles before sysprep
3. Ensure no pending reboots
4. Disable Windows Store updates during build

Debug Mode Execution

For detailed troubleshooting, enable debug logging:

```
# Enable debug output
export PACKER_LOG=1
export PACKER_LOG_PATH="packer-debug.log"

# Run build with debugging
packer build -var-file="vars.json" windows.pkr.hcl
```

Resource Cleanup

If build fails, clean up Azure resources:

```
# List resource groups
az group list --output table

# Delete resource group (removes all resources)
az group delete --name myPackerGroup --yes --no-wait

# Or delete specific resources
az vm delete --resource-group myPackerGroup --name packer-build-vm --yes
az disk delete --resource-group myPackerGroup --name packer-build-disk --yes
```

📚 Best Practices

Security Best Practices

1. **Credential Management**
   - Use Azure Key Vault for sensitive variables
   - Rotate service principal credentials regularly
   - Never commit credentials to version control
2. **Network Security**
   - Use private subnets for build process

- Implement least-privilege network access
- Configure NSG rules to restrict RDP/WinRM access
- Use Azure Bastion for secure remote access

3. **Image Security**
   - Keep base images updated with latest patches
   - Remove unnecessary services and features
   - Implement proper antivirus exclusions
   - Configure Windows Defender appropriately

Performance Optimization

1. **Build Performance**
   - Use SSD storage for build VMs
   - Select appropriate VM sizes (Standard_D2s_v3 minimum)
   - Parallel provisioning where possible
   - Cache frequently downloaded packages
2. **Image Optimization**
   - Remove temporary files and caches
   - Optimize disk usage before sysprep
   - Configure services for optimal startup
   - Implement proper power management settings

Automation Best Practices

1. **Template Design**
   - Use variables for all configurable parameters
   - Implement proper error handling in scripts
   - Create modular, reusable components
   - Document all customizations
2. **CI/CD Integration**

```
# Azure DevOps Pipeline Example
trigger:
  branches:
    include:
      - main
  paths:
    include:
      - windows10-packer/*

pool:
  vmImage: 'ubuntu-latest'

variables:
  - group: packer-variables

steps:
- task: PackerTool@0
  inputs:
    version: 'latest'

- script: |
    packer init windows.pkr.hcl
    packer validate -var-file="vars.json" windows.pkr.hcl
    packer build -var-file="vars.json" windows.pkr.hcl
  displayName: 'Build Golden Image'
  workingDirectory: 'windows10-packer'
```

3. **Version Control**
   - Tag releases with semantic versioning
   - Maintain changelog for image versions
   - Use branching strategy for different environments
   - Implement automated testing

Monitoring and Maintenance

1. **Image Lifecycle Management**

- Schedule regular image updates (monthly/quarterly)
- Track image usage and deployment metrics
- Implement automated cleanup of old images
- Monitor security updates and patches

2. **Build Monitoring**

```
# Monitor build progress
watch -n 30 'az resource list --resource-group myPackerGroup --outp
ut table'

# Check build logs
tail -f packer-debug.log
```

3. **Cost Optimization**
   - Use spot instances for build process when possible
   - Delete failed builds immediately
   - Implement automatic resource cleanup
   - Monitor Azure costs and set budgets

📊 Advanced Configuration Options

Custom Application Installations

Add custom applications to your `install-apps.ps1` script:

```
# Custom Enterprise Applications
$EnterpriseApps = @(
    @{
        Name = "Microsoft Office"
        Installer = "https://your-repo.com/office-installer.exe"
        Arguments = "/S /v/qn"
    },
    @{
        Name = "Corporate VPN Client"
        Installer = "\\your-server\software\vpn-client.msi"
```

```
        Arguments = "/quiet"
    }
)

foreach ($App in $EnterpriseApps) {
    Write-Host "Installing $($App.Name)..." -ForegroundColor Yellow
    try {
        $InstallerPath = "C:\Temp\$($App.Name).exe"
        Invoke-WebRequest -Uri $App.Installer -OutFile $InstallerPath
        Start-Process -FilePath $InstallerPath -ArgumentList $App.Arguments
-Wait
        Remove-Item $InstallerPath -Force
        Write-Host "$($App.Name) installed successfully" -ForegroundColor
Green
    } catch {
        Write-Warning "Failed to install $($App.Name): $($_.Exception.Messa
ge)"
    }
}
```

Registry Customizations

Add registry modifications to `configure-system.ps1`:

```
# Corporate Registry Settings
$RegistrySettings = @(
    @{
        Path = "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpda
te"
        Name = "WUServer"
        Value = "http://your-wsus-server.domain.com"
        Type = "String"
    },
    @{
        Path = "HKLM:\SOFTWARE\Policies\Microsoft\Edge"
        Name = "HomepageLocation"
        Value = "https://your-intranet.com"
        Type = "String"
```

```
    }
)

foreach ($Setting in $RegistrySettings) {
    try {
        if (-not (Test-Path $Setting.Path)) {
            New-Item -Path $Setting.Path -Force | Out-Null
        }
        Set-ItemProperty -Path $Setting.Path -Name $Setting.Name -Value $S
etting.Value -Type $Setting.Type
        Write-Host "Set registry: $($Setting.Path)\$($Setting.Name)" -Foregro
undColor Green
    } catch {
        Write-Warning "Failed to set registry setting: $($Setting.Path)\$($Setti
ng.Name)"
    }
}
```

Multi-Environment Support

Create environment-specific variable files:

```
# Production environment
vars-prod.json

# Development environment
vars-dev.json

# Testing environment
vars-test.json
```

```
# Build for different environments
packer build -var-file="vars-prod.json" windows.pkr.hcl
packer build -var-file="vars-dev.json" windows.pkr.hcl
```

🔄 Continuous Integration Example

GitHub Actions Workflow

Create `.github/workflows/packer-build.yml`:

```
name: Build Windows 10 Golden Image

on:
  push:
    branches: [ main ]
    paths:
      - 'windows10-packer/**'
  pull_request:
    branches: [ main ]
  schedule:
    - cron: '0 2 * * 1'  # Weekly builds on Monday at 2 AM

env:
  PACKER_VERSION: "1.9.4"

jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Setup Packer
      uses: hashicorp/setup-packer@main
      with:
        version: ${{ env.PACKER_VERSION }}

    - name: Initialize Packer
      run: packer init windows10-packer/windows.pkr.hcl

    - name: Validate Packer Template
      run: |
        cd windows10-packer
        packer validate -var-file="vars-template.json" windows.pkr.hcl

  build:
```

```
    needs: validate
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v3

    - name: Setup Packer
      uses: hashicorp/setup-packer@main
      with:
        version: ${{ env.PACKER_VERSION }}

    - name: Azure Login
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: Create Variables File
      run: |
        cat > windows10-packer/vars.json << EOF
        {
          "client_id": "${{ secrets.AZURE_CLIENT_ID }}",
          "client_secret": "${{ secrets.AZURE_CLIENT_SECRET }}",
          "tenant_id": "${{ secrets.AZURE_TENANT_ID }}",
          "subscription_id": "${{ secrets.AZURE_SUBSCRIPTION_ID }}"
        }
        EOF

    - name: Build Golden Image
      run: |
        cd windows10-packer
        packer init windows.pkr.hcl
        packer build -var-file="vars.json" windows.pkr.hcl

    - name: Cleanup
      if: always()
      run: rm -f windows10-packer/vars.json
```

📋 Quality Assurance Testing

Automated Testing Script

Create `test-golden-image.ps1`:

```
# Golden Image Quality Assurance Test Script
param(
    [Parameter(Mandatory=$true)]
    [string]$VMName,

    [Parameter(Mandatory=$true)]
    [string]$ResourceGroup
)

$ErrorActionPreference = 'Stop'

Write-Host "Starting Golden Image QA Tests for VM: $VMName" -Foregrou
ndColor Green

# Test Results Array
$TestResults = @()

# Test 1: Verify Applications
Write-Host "Testing installed applications..." -ForegroundColor Yellow
$RequiredApps = @("Notepad++", "7-Zip", "Google Chrome", "Mozilla Firef
ox")
$InstalledApps = Get-WmiObject -Class Win32_Product | Select-Object -Ex
pandProperty Name

foreach ($App in $RequiredApps) {
    $Found = $InstalledApps | Where-Object { $_ -like "*$App*" }
    $TestResults += [PSCustomObject]@{
        Test = "Application: $App"
        Status = if ($Found) { "PASS" } else { "FAIL" }
        Details = if ($Found) { "Installed" } else { "Not Found" }
    }
}
```

```
# Test 2: System Configuration
Write-Host "Testing system configuration..." -ForegroundColor Yellow

# Timezone Test
$CurrentTZ = Get-TimeZone
$TestResults += [PSCustomObject]@{
    Test = "Timezone Configuration"
    Status = if ($CurrentTZ.Id -eq "Eastern Standard Time") { "PASS" } else {
"FAIL" }
    Details = $CurrentTZ.DisplayName
}

# Power Plan Test
$CurrentPowerPlan = (powercfg -getactivescheme).Split()[3]
$HighPerfGuid = (powercfg -list | Where-Object { $_ -match "High perform
ance" }).Split()[3]
$TestResults += [PSCustomObject]@{
    Test = "Power Plan Configuration"
    Status = if ($CurrentPowerPlan -eq $HighPerfGuid) { "PASS" } else { "FA
IL" }
    Details = "Current: $CurrentPowerPlan"
}

# Test 3: Directory Structure
Write-Host "Testing directory structure..." -ForegroundColor Yellow
$RequiredDirs = @("C:\Scripts", "C:\Logs", "C:\Apps")
foreach ($Dir in $RequiredDirs) {
    $TestResults += [PSCustomObject]@{
        Test = "Directory: $Dir"
        Status = if (Test-Path $Dir) { "PASS" } else { "FAIL" }
        Details = if (Test-Path $Dir) { "Exists" } else { "Missing" }
    }
}

# Test 4: Windows Updates
Write-Host "Checking Windows Updates..." -ForegroundColor Yellow
try {
    $UpdateHistory = Get-HotFix | Measure-Object
```

```
    $TestResults += [PSCustomObject]@{
        Test = "Windows Updates"
        Status = if ($UpdateHistory.Count -gt 10) { "PASS" } else { "WARN" }
        Details = "$($UpdateHistory.Count) updates installed"
    }
} catch {
    $TestResults += [PSCustomObject]@{
        Test = "Windows Updates"
        Status = "FAIL"
        Details = "Could not check update status"
    }
}

# Display Results
Write-Host "`nQA Test Results:" -ForegroundColor Green
Write-Host "==================" -ForegroundColor Green
$TestResults | Format-Table -AutoSize

# Summary
$PassCount = ($TestResults | Where-Object { $_.Status -eq "PASS" }).Cou
nt
$FailCount = ($TestResults | Where-Object { $_.Status -eq "FAIL" }).Count
$WarnCount = ($TestResults | Where-Object { $_.Status -eq "WARN" }).Co
unt
$TotalTests = $TestResults.Count

Write-Host "`nTest Summary:" -ForegroundColor Green
Write-Host "=============" -ForegroundColor Green
Write-Host "Passed: $PassCount/$TotalTests" -ForegroundColor Green
Write-Host "Failed: $FailCount/$TotalTests" -ForegroundColor Red
Write-Host "Warnings: $WarnCount/$TotalTests" -ForegroundColor Yellow

# Exit with appropriate code
if ($FailCount -gt 0) {
    Write-Host "QA Tests FAILED - Image requires attention" -ForegroundCol
or Red
    exit 1
} elseif ($WarnCount -gt 0) {
```

```
    Write-Host "QA Tests PASSED with warnings" -ForegroundColor Yellow
    exit 0
} else {
    Write-Host "All QA Tests PASSED - Image ready for deployment" -Foregr
oundColor Green
    exit 0
}
```

📖 Additional Resources

Documentation Links

- HashiCorp Packer Documentation
- Azure ARM Builder
- PowerShell DSC with Packer
- Azure CLI Reference

Community Resources

- Packer Community Forum
- Azure DevOps Extensions
- GitHub Packer Templates

Enterprise Considerations

1. **Compliance Requirements**
    - NIST cybersecurity framework alignment
    - SOC 2 Type II compliance
    - GDPR data protection requirements
    - Industry-specific regulations (HIPAA, PCI-DSS)
2. **Integration Points**
    - Active Directory domain join
    - Certificate authority integration
    - WSUS server configuration

- Group Policy application

3. **Monitoring Integration**
    - Azure Monitor integration
    - Log Analytics workspace
    - Security Center compliance
    - Custom dashboard creation

Document Version: 1.0

Last Updated: June 2025

Author: Infrastructure Automation Team

Review Date: Quarterly

This documentation is maintained as a living document and should be updated regularly to reflect changes in requirements, procedures, and best practices.