Installation manual

Hybrid IT management framework

Document Revision History

Project Name: Hybrid IT management framework – Installation manual

Document Status: Final

Document Version	Date	Prepared / Modified by	Reviewed by	Approved by	Section and Text Revised
1.0	20 May 2021	Sien Van Broekhoven	Dries Verschaeve		Full document

Table of Contents

Introduction

The purpose of this document is to form an installation manual that guides IT administrators through the set-up of the Hybrid IT Management Framework.

This document will describe how all aspects of the Hybrid IT Management Framework were configured in a lab environment, as well as providing information for a set-up in a production environment and integration with an already existing on-premise environment. These configurations are based on the design decisions that were made in the design-guide document.

This document consists of different parts, each describing a specific concept of the infrastructure/configurations.

The source code and documentation can be found on the Azure DevOps site for HPE employees; or in this GitHub repository, and my portfolio for external people.

1. Base of environment

In my lab environment the base of the environment will be configured with Terraform configuration files. These configuration files are written in HashiCorp Language (HCL), which is a declarative infrastructure as code language created by HashiCorp. On top of these Terraform files, PowerShell scripts will be deployed to targets (virtual machines), to provide these machines with additional configurations.

The structure of the coding files can be found below:

- dev/
  - providers.tf - declare the providers to be used
  - on-prem-env.tf - create resources to simulate an on-premises environment
  - shared-env.tf - create resources for an Azure shared virtual environment
  - spokes-env.tf - create resources for an Azure spoke environment
  - files/
    - winrm[*].ps1 - initial configurations that will be deployed to machines
    - Configure.ps1 - additional configurations
    - WAC.ps1 - WAC configuration
    - FirstLogonCommands.xml- PowerShell script to request SSL certificates
    - FirstLogonCommands.xml - file that deploys initial configurations to machines

---

Important
An organization usually already has an existing on-premise environment. The on-prem-env.tf file was used in my lab to simulate an on-premise environment in Azure, but can be deleted if an actual on-premise environment already exists.

This working structure is built inside a development environment called dev. To create this working environment, use the following CLI commands.

```
terraform workspace new dev
```

```
terraform workspace select dev
```

It is best practice to work with different working directories, representing different environments for every situation. A production environment will thus not be built in the dev environment, it should be built in a “prod” environment.

Table 1: Naming of resources

	on-prem.tf	shared-env.tf	spokes-env.tf
Resource groups	• rg_On-Prem-Networking • rg_On-Prem-DC • rg_On-Prem-Clients	• rg_Network • rg_DC • rg_Bastion • rg_Security • rg_Management	• rg_Services
vNET	• On-PremVNET	• NetworkingVNET	• SpokesVNET
Subnets	• On-Prem-DC-Subnet • On-Prem-Clients-Subnet • On-Prem-Gateway-Subnet	• DC-Subnet • Management-Subnet • Gateway-Subnet • AzureBastionSubnet	• Services-Subnet
Network Security Groups	• On-Prem-DC-NSG • On-Prem-Clients-NSG	• DC-NSG • Management-NSG • Bastion-NSG	• Services-NSG
Public IPs		• Bastion-PIP	
Network Interface Cards	• On-Prem-DC-NIC • On-Prem-ADC-NIC • On-Prem-Client-NIC	• DC-NIC • Management-NIC • Bastion-NIC	• Service-NIC
Allocations NSG - NIC	• On-Prem-DC-NSG_to_Client-NIC • On-Prem-Clients-NSG_to_Client-NIC	• DC-NSG_to_Client-NIC • Bastion-NSG_to_Bastion-NIC • Management-NSG_to_Management-NIC	• Services-NSG_to_DC-Service
Network Peering	• On-Prem-to-Shared	• Shared-to-On-Prem	• Spokes-to-Shared
Virtual Machines	• On-Prem-DC • On-Prem-ADC • On-Prem-Client	• DC • ManagementVM	
Availability sets		• Management-Availability • Bastion-Availability	
Load balancers for availability sets		• Management-LB • Bastion-LB	

Note
All resources starting with “On-Prem” are to simulate an on-premise environment for my lab. An organization using this guide to migrate to the Azure cloud will not need this On-Prem environment, they will have their own physical resources.

The configuration is further explained with comments in the terraform files.

Terraform commands explained:

`Terraform init` - prepare your working directory for other commands

`Terraform validate` - check whether the configuration is valid

`Terraform plan` - show changes required by the current configuration

`Terraform apply` - create or update infrastructure

`Terraform destroy` - destroy previously-created infrastructure

All passwords used in the Terraform configuration files are saved in an Azure Keyvault and then retrieved to use in the deployment of resources. This is more secure than using plaintext passwords directly in the configuration files. A keyvault containing a password can be created with the following PowerShell script.

```
### Replace XXXX with a random number to ensure uniqueness
New-AzKeyVault -VaultName 'SecretsKeyVaultXXXX' -ResourceGroupName 'rg_KeyVault' -Location 'WestEu
rope'

### Replace Password with a secure password
$Secret = ConvertTo-SecureString -String 'Password' -AsPlainText -Force
PS C:\> Set-AzKeyVaultSecret -VaultName 'SecretsKeyVaultXXXX' -Name 'sien' -SecretValue $Secret
```

2. Active Directory

Active Directory brings identity authentication and management to the environment. Both Microsoft Active Directory and Microsoft Azure Active Directory will be used.

- **Microsoft Active Directory** - This will be used to simulate an already existing on-premise active directory environment. It will contain users, groups, and computers.
- **Microsoft Azure Active Directory** - This is a cloud solution for Active Directory, it can be combined with the on-premise AD to expand the management and service possibilities to the cloud.

Table 2: Active Directory resources

Resource	Description
On-Prem-DC	First DC in new forest and domain (stagesien.com)
DC	Second DC, connected to On-Prem-DC, provides failover support
On-Prem-ADC	Active Directory Sync to Azure AD
Azure AD tenant	Azure AD, clients authenticate here

Note
The On-Prem-DC is configured to simulate a server configured as the primary domain controller in my lab environment.

On-Prem-ADC is a server which will run Active Directory Connect to synchronize the on-premises Active Directory environment with the Azure Active Directory (cloud) environment. This is again a simulated virtual machine for my lab, but would be a Windows server machine in an already existing organization's environment.

Both DC and the Azure AD tenant are cloud resources.

Configuration:

Make sure the DNS settings of NetworkingVNET and On-PremVNET (lab) are set to the IP address of On-Prem-DC, else they cannot join the Active Directory domain.

```
# Create NetworkingVNET within rg_Network
resource "azurerm_virtual_network" "NetworkingVNET" {
```

```
  name                = "NetworkingVNET"
  resource_group_name = azurerm_resource_group.rg_Network.name
  location            = azurerm_resource_group.rg_Network.location
  address_space       = ["10.1.0.0/16"]
  dns_servers         = [ azurerm_network_interface.On-Prem-DC-NIC.private_ip_address ]
}
```

Important
In an environment with an already existing on-premise network the DNS settings of NetworkingVNET need to point to a DC which hosts a DNS server.

The winrm[*].ps1 configuration files in the folder files/ and the file FirstLogonCommands.xml deliver the necessary configurations to the virtual machines to make them join the AD domain.

The following code in the creation of the DC in shared-env.tf delivers winrm.ps1 as custom data to the machine, this custom data is converted to a binary file. FirstLogonCommands converts this binary file back to a PowerShell script and executes it. Every machine that is created in the terraform files will have a code block like this.

```
os_profile {
    computer_name  = "DC"
    # Note: you can't use admin or Administrator in here, Azure won't allow you to do so :-)
    admin_username = "sien"
    admin_password = "XXXX"
    custom_data    = "${file("./files/winrm.ps1")}"
  }

  os_profile_windows_config {
    provision_vm_agent = "true"
    timezone           = "Central European Standard Time"
    winrm {
      protocol = "http"
    }
    # Auto-Login's required to configure WinRM
    additional_unattend_config {
      pass         = "oobeSystem"
      component    = "Microsoft-Windows-Shell-Setup"
      setting_name = "AutoLogon"
      content      = "<AutoLogon><Password><Value>XXXX</Value></Password><Enabled>true</Enabled><L
ogonCount>1</LogonCount><Username>sien</Username></AutoLogon>"
    }
    additional_unattend_config {
      pass         = "oobeSystem"
      component    = "Microsoft-Windows-Shell-Setup"
      setting_name = "FirstLogonCommands"
      content      = "${file("./files/FirstLogonCommands.xml")}"
    }
  }
}
```

A timer in shared-env.tf will wait until On-Prem-DC is created before configuring the other machines. This is necessary in my lab environment because the virtual machines will not be able to join the AD domain if the domain does not exist yet. The creation of this initial forest and domain takes around 15 minutes.

For every machine created, except the On-Prem-DC, a dependency on this timer will be created. An example of this can be found in the second code block below.

```
## Wait some time to make sure forest and first DC are configured before creating other machines
resource "time_sleep" "wait_for_15_minutes" {
  depends_on = [azurerm_resource_group.rg_On-Prem-DC]
  create_duration = "15m"
}

## create the DC VM
resource "azurerm_virtual_machine" "DC" {
  depends_on            = [time_sleep.wait_for_15_minutes]
  [truncated]
}
```

Note
This timer and it's dependencies can be deleted if an on-premise environment with an Active Directory forest already exists. Make sure that the connection to the on-premise environment is configured, else the new machines will not be able to join the AD domain. This connection can either be a VPN tunnel or an Azure Expressroute depending on the needs of your organization.

Azure AD tenant:

If an organization does not have an Azure AD tenant yet, it can be manually created in the Azure portal, there is no API support to do this through a CLI yet. The Azure AD domain that is used in my dev environment is stagesien.onmicrosoft.com. The .onmicrosoft.com can be changed if you have a registered domain name.

A connection between the on-premise AD and the Azure AD will be made through a Windows Server running AD connect. This server will sync the changes made in the on-premise environment to the cloud AD environment.

The On-Prem-ADC virtual machine is added to the AD domain stagesien.com with the commands that can be found in winrmADC.ps1. This script will also install the RSAT-AD-Tools.

The following PowerShell script snippet can be found in Configure.ps1, this will download the necessary files and modules that are needed to install and configure Microsoft Active Directory Connect. This script can be delivered to the On-Prem-ADC VM through remote Powershell, or through an automation tool such as DevOps pipelines.

```
## Download AD Connect .msi and save on C drive
$url = "https://download.microsoft.com/download/B/0/0/B00291D0-5A83-4DE7-86F5-980BC00DE05A/AzureAD
Connect.msi"
$outpath = "C:/AzureADC.msi"
Invoke-WebRequest -Uri $url -OutFile $outpath

## Run .msi
Start-Process -Filepath "C:/AzureADC.msi"

Start-Sleep -s 20

## Import AD sync module
Import-Module "C:\Program Files\Microsoft Azure Active Directory Connect\AdSyncConfig\AdSyncConfig
.psm1"
```

After this, you will need to manually configure the variables in the AD connect installation GUI. This installation program is automatically downloaded and run, but there is no PowerShell support to configure it remotely. After the variables are configured, Azure AD sync will start.

These variables can be configured through a Bastion, which will provide a RDP connection to the VM. Or they can be configured through the Windows Admin Center portal which will be hosted on the ManagementVMs, after this is configured.

Password writeback should be enabled in the configuration settings of AD sync to be able to reset account passwords through self service password reset later on.

To manually sync or check sync settings:

```
Import-Module -Name "C:\Program Files\Microsoft Azure AD Sync\Bin\ADSync"

## Check sync settings, default is sync once every 30 mins
Get-ADSyncScheduler

## Force sync to start
Start-ADSyncSyncCycle -PolicyType Delta
```

In the lab environment a UPN to the .onmicrosoft.com domain needs to be added in the On-Prem-DC. This is configured with the following code, which can be found in Configure.ps1. This can also be done through a bastion RDP connection.

```
Set-ADForest -UPNSuffixes @{add="stagesien.onmicrosoft.com"}
```

Note
In production environments custom domain names are usually already in use. Creating a UPN to the .onmicrosoft.com domain is then not necessary.

Users can be created and managed in the On-Prem AD, which will then replicate to the Azure AD. The users will get an .onmicrosoft.com account to log onto microsoft services and log onto client pcs. If a different domain name is used, the users will receive a user account with this domain suffix instead.

When users are created in Azure AD they will not sync back to the on-premise environment, so they must be managed through the on-prem AD, then they will sync to the Azure AD. If you try to modify a synced user through Azure AD, you will receive an error message.

3. Windows Admin Center

Windows Admin Center will be hosted on an availability cluster that consists of ManagementVM0 and ManagementVM1. If one of the VMs fails, the other one will take over. These servers are also load-balanced, meaning the incoming network traffic will be distributed between the two servers.

Both of these VMs are added to the AD domain through the winrmclient.ps1 custom data which is delivered and executed on the VMs through terraform.

The “failover clustering” role will be installed on both VMs, this code snippet can be found in Configure.ps1.

```
Install-WindowsFeature -Name Failover-Clustering –IncludeManagementTools
Install-windowsFeature RSAT-Clustering -IncludeAllSubFeature -Restart
```

To allow these VMs to be clustered together in an availability set, a shared disk needs to created in Azure and connected to the virtual machines. This can fully be automated through the following CLI commands/scripts.

The following Azure CLI script creates a shared P15 premium SSD, and attaches it to the ManagementVMs.

```
az disk create -g rg_Management -n ManagementDisk --size-gb 256 -l westeurope --sku Premium_LRS --
max-shares 2

az vm disk attach -g rg_Management --vm-name ManagementVM0 --name ManagementDisk
az vm disk attach -g rg_Management --vm-name ManagementVM1 --name ManagementDisk
```

The disk then has to be initialized, partitioned, and formatted on ManagementVM0. This can be done with the following script.

```
Get-Disk | Where-Object IsOffline –Eq $True | Set-Disk –IsOffline $False
Get-Disk | Where-Object PartitionStyle –Eq 'RAW' | Initialize-Disk –PartitionStyle MBR
New-Partition –DiskNumber 2 -DriveLetter F –UseMaximumSize
Format-Volume -DriveLetter F -FileSystem NTFS -Confirm:$false
```

The script to create a failover cluster and install Windows Admin Center on this cluster has to be executed locally on ManagementVM0. The following script will download the necessary files to install Windows Admin Center. It will also create a PowerShell script to configure an availability cluster and place it on the C drive of the machine, it then only needs to executed by a logged on administrator.

```
## Install Windows Admin Center

## Download Install-WindowsAdminCenterHA.zip
$url = "https://aka.ms/WACHAScript"
$outpath = "C:\Install-WindowsAdminCenterHA.zip"
Invoke-WebRequest -Uri $url -OutFile $outpath

## Unzip
Expand-Archive -LiteralPath 'C:\Install-WindowsAdminCenterHA.zip' -DestinationPath C:\Scripts

## Download Windows Admin Center .msi
$url = "https://aka.ms/WACDownload"
$outpath = "C:/WindowsAdminCenter.msi"
Invoke-WebRequest -Uri $url -OutFile $outpath

$text = @"
## Create availability cluster
New-Cluster -Name ManagementCluster -Node ManagementVM0.stagesien.com, ManagementVM1.stagesien.com
Set-ClusterQuorum -NoWitness
```

```
## Add cluster disk
Get-ClusterAvailableDisk | Add-ClusterDisk
Add-ClusterSharedVolume -Name "Cluster Disk 1"

## Install WAC in HA mode
C:\Scripts\Install-WindowsAdminCenterHA.ps1 -clusterStorage "C:\ClusterStorage\Volume1" -clientAcc
essPoint "stagesien-ha-gateway" -msiPath "C:/WindowsAdminCenter.msi" -generateSslCert -Verbose
"@

$text -f 'string' | Out-File C:\script.ps1
```

Windows Admin Center is available at https://ManagementCluster.stagesien.com:443 on google chrome, edge, and firefox. Internet Explorer is not supported.

Note
The domain name will be ManagementVM0.stagesien.com if WAC is not deployed in high availability mode, but instead is installed on a single machine where ManagementVM0 is the fully qualified domain name (FQDN) of the machine.

This site can be accessed through any computer in the environment. The different administrator accounts that can log in, are defined and configured in role based access control. Currently the configuration uses a self-signed certificate, but this is replaced with an Active Directory certificate in the next chapter.

AD certificates

Certificates can be issued to computers and users in an Active Directory environment through the Active Directory Certificate Services role. Certificates confirm the identity of a specific person, device or service over a secure SSL connection. A certificate will be created for the Windows Admin Center console, which will then be distributed through an AD Group Policy Object. Users will then be able to securely connect to this console without getting a certificate error.

Important
SSL certificates are a must in production environments. If an organization uses a different SSL authority service than AD certificates, that SSL authority may be used instead.

Install AD CS role

The first step is installing the Active Directory Certificate Services role on a Windows Server that you want to use as the root authority server. In my environment, this is on DC.stagesien.com. This server has to be configured as a standalone root CA, and the CA web enrollment service also has to be installed on this server. This last service allows clients to connect to the CA through a web browser to perform tasks, such as requesting certificates.

The following PowerShell command installs a new standalone root CA which will use RSA encryption, then the web enrollment service will be installed:

```
Install-WindowsFeature ADCS-Cert-Authority -IncludeManagementTools
Install-AdcsCertificationAuthority -CAType StandaloneRootCa –CACommonName “DC-CA”  -CryptoProvider
Name "RSA#Microsoft Software Key Storage Provider" -KeyLength 2048 -HashAlgorithmName SHA1 -Validi
tyPeriod Years -ValidityPeriodUnits 3 -Force
Import-Module ServerManager
Add-WindowsFeature Adcs-Web-Enrollment
Install-AdcsWebEnrollment -CAConfig "DC\DC-CA-1" -Force
```

Request certificate

We can request an SSL certificate for the WAC console through the Microsoft Management Console (MMC).

In MMC on a management server click on File - Add/remove Snap in, and add “Certificates”. Select “computer account” in the next screen, then local computer and finish. Open the console, expand the certificates, right click on Personal and create a custom request.

In the enrollment wizard select “Proceed without enrollment policy”. Use “(no template) Legacy key” as template and PKCS#10 as request format. In the next screen click on properties to fill in the information for the certificate.

The following table summarizes the values that the certificate must contain.

Table 3: Certificate values

Field	Value
Friendly name	ManagementCertificate
Common name	ManagementCluster.stagesien.com
Organization	Stagesien.com
OU	Managed Devices
Locality	Tessenderlo
State	Limburg
Country	BE
Keytype	Microsoft RSA SChannel Cryptographic Provider (Encryption)
Keysize	2048

The friendly name is a name to identify the certificate.

Note
The Managed Devices OU contains all servers that are configured with LAPS in the lab environment, which is explained later in this document.

The common name has to match the hostname of WAC, so depending on whether WAC is deployed in high availability mode it will be ManagementCluster.stagesien.com or if it’s deployed on one server only, it will be ManagementVM0.stagesien.com, where ManagementVM0 is the hostname of the server running WAC.

Create the certificate request when these fields are entered.

Then browse to http://10.1.1.4/certsrv, where 10.1.1.4 is the IP of the DC. On this website, click on request a certificate, advanced certificate request, and upload the content of the previously created request in this textbox.

Back on the DC, open the Certification Authority tool, go to pending requests, right click the previously created request and click “issue”. The request is now accepted.

On the managementVM again, browse back to http://10.1.1.4/certsrv, view the status of the request and download the certificate file. Run this file and install it under the “Personal” folder in the certificates store. Export the certificate and its key to update the Windows Admin Center configuration with the correct certificate. Save the exported file in the “C:/Scripts” folder and make sure to use a password that you can remember. Then execute the following PowerShell script, which will update the WAC configuration.

```
$certPassword = Read-Host -AsSecureString
.\Install-WindowsAdminCenterHA.ps1 -certPath "cert.pfx" -certPassword $certPassword -Verbose
```

The public key of the Certification Authority server is automatically distributed to domain member machines. This certification authority now holds the certificate for the management servers, making the ManagementVMs trusted by domain member machines.

Register with Azure

To use Azure services in the Windows Admin Center portal, WAC needs to be registered as an application in Azure first. This can be done through settings - Azure in the WAC portal, then just follow the instructions on screen.

Important
Applications can only be registered by accounts with global administrator or application administrator privileges in Azure Active Directory.

This application then has to be granted the following permissions by a Global Administrator, Privileged Role Administrator, or an Application Administrator.

- Read directory data
- Read and write directory data
- Access directory as the signed in user
- Read applications
- Read and write all applications
- Access your organization’s directory
- Enable sign-on and read users’ profiles
- Access Azure Service Management as organization users (preview)

There is also an option to use Azure AD authentication in addition to regular Windows authentication as an extra security measure for the WAC gateway. Enabling this option will automatically register an application in Azure AD regardless or not if you registered manually in the previous step.

Users would have to authenticate with their AD account after having authenticated with their Windows account. User allocation can be managed in Azure AD, the users that need to be able to access the WAC portal, have to be registered in the Azure AD application. If the option “User assignment required” in the properties of the app is set to Yes, then only the registered users are able to access the WAC portal.

Note
After testing this out in my lab environment, it is clear that this is not a good fit for a production environment at the moment of writing yet. There is not a lot of documentation of this feature available online, so troubleshooting any encountered issues is not easy. On top of that, users would also need to be assigned to the local users group on the gateway server to be able to use Azure AD authentication, while this is not required while only using Widows authentication.

Extensions

There are many extensions available to install on a Windows Admin Center server. These extensions bring additional management possibilities to the WAC portal.

Extensions can be installed by going to settings - extensions in the WAC portal.

Active Directory

The Active Directory extension allows you to create users & groups, reset user passwords, configure objects, and more from the WAC interface instead of having to directly do this on an Active directory domain controller.

4. Jumphost & Services

Azure Bastion

The following code in shared-env.tf creates a bastion jumphost. This machine can be used to open connections to other machines in the environment. This eliminates the need for public IP addresses on other machines, and thus improves security.

```
## Create a bastion jumphost
resource "azurerm_bastion_host" "Bastion" {
  name                = "Bastion"
  location            = azurerm_resource_group.rg_Bastion.location
  resource_group_name = azurerm_resource_group.rg_Bastion.name

  ip_configuration {
    name                 = "configuration"
    subnet_id            = azurerm_subnet.AzureBastionSubnet.id
    public_ip_address_id = azurerm_public_ip.Bastion-PIP.id
  }
}
```

5. Privileged Access implementation

The following guide instructs how to configure several Privileged access components. All three security levels (enterprise, specialized, and privileged) will be set up, to then be assigned to roles in the organization.

These configurations are made to improve security for domain joined devices, and harden device configurations.

Azure AD device configuration

In Azure Active Directory create the following users and groups.

Table 4: Users

Name	Username	Directory role	Usage location
Secure Workstation User	secure-ws-user@stagesien.onmicrosoft.com	Intune Administrator	Belgium
Secure Workstation Administrator	secure-ws-admin@stagesien.onmicrosoft.com	Intune Administrator	Belgium

Table 5: Groups

Group name	Group type	Membership type	Group name
Secure Workstation Users	Security	Assigned	Secure Workstation Users
Secure Workstation Admins	Security	Assigned	Secure Workstation Admins
Emergency BreakGlass	Security	Assigned	Emergency BreakGlass
Secure Workstations	Security	Dynamic Device	Secure Workstations

Add the following Dynamic Membership rule to the Secure Workstations group:

```
(device.devicePhysicalIds -any _ -contains "[OrderID]:PAW")
```

User secure-ws-user is added to the Secure Workstation Users group, and user secure-ws-admin is added to the Secure Workstations Admin group.

Specify the group “Secure Workstation Users” as the users who may join devices to Azure AD, under Azure AD - Devices - Device settings.

Important
The action above will prevent users from joining their device towards Azure AD. It is important to check with your organization policies, and existing configurations before making changes in these settings.

In Azure AD - Devices - Device Settings, select none under “additional local admins on Azure AD joined devices” to prevent users from having administrator rights on their machines.

Also enable Multi Factor Authentication (MFA) in Devices - Device Settings.

To configure mobile device management set “MDM user scope” to all in Azure AD - Mobility (MDM and MAM) - Microsoft Intune.

Note
Privileged Admin Workstations are beyond the scope of this guide, but more information can be found in the Microsoft documentation.

Azure AD Conditional Access

Azure AD Conditional Access has the ability to restrict access to certain resources for certain groups/users. Emergency access however should always be possible, so this group has to be excluded from conditional access definitions.

The following guide is to create a conditional access rule to only allow secured workstations to access the Azure portal.

In the Microsoft Endpoint Manager (MEM) admin center under devices, conditional access, create a new policy with the following information:

```
Name: Compliant device access Azure portal
Include directory roles: Global Administrator, Privileged Role Administrator, Privileged Authentic
ation Administrator, Security Administrator, Compliance Administrator, Conditional Access Administ
rator, Application Administrator, Cloud Application Administrator, Intune Service Administrator
Exclude group: Emergency BreakGlass
Cloud apps: All cloud apps
Conditions: Device platforms - Windows
Access controls: Grant access when device is marked as compliant
Enable policy: on
```

Microsoft Intune configuration

Prevent BYOD access

Preventing Bring Your Own Devices (BYOD) access improves security because this setting will prevent potentially insecure devices from joining the domain.

1. In MEM - Devices - Enrollment restrictions, choose the default restriction “All users”
2. In properties - platform settings click “edit”
3. Select “block” for all types except for Windows MDM
4. Select “block” for all personally owned items

Create an autopilot deployment profile

A deployment profile needs to be created to configure the autopilot devices. This will be linked to a previously created device group.

In the MEM admin center under Device enrollment - Windows enrollment - Deploy profiles - create a new profile with the following information.

```
Name: Secure workstation deployment profile
Description: Deployment of secure workstations
Convert all targeted devices to Autopilot: Yes
NEXT
Deployment mode: User-driven
Join to Azure AD as: Hybrid Azure AD joined
Language (Region): Operating system default
User account type: standard
```

```
NEXT
Assignments - Assign to - Selected Groups - Include: Secure Workstations
NEXT
CREATE
```

Note
Convert all targeted devices to Autopilot= all devices in the list get registered with the Autopilot deployment service

Enrollment Status Page

The Enrollment Status Page (ESP) displays the provisioning progress after a new device is enrolled. To ensure that devices are fully configured before use, Intune provides a means to Block device use until all apps and profiles are installed.

In MEM - Devices - Windows - Windows enrollment - Enrollment Status Page - Create profile with the following information

```
Name: Secure Workstations ESP
Description: Enrollment status page for Secure Workstations group.
CREATE
Show app and profile configuration progress: Yes
Block device use until all apps and profiles are installed: Yes
NEXT
Included groups: Secure Workstations
NEXT
NEXT
CREATE
```

Configure Windows Update

An update ring is a service that manages the pace updates are applied to workstations. Keeping Windows up to date is one of the most important things to do to secure a workstation.

An update ring can be created in MEM admin center - Devices - Windows - Software updates - Windows 10 Update Rings - Create profile with the following information.

```
Name: Azure-managed workstation updates
NEXT
Servicing channel: Semi-annual channel
Quality update deferral (days): 3
Feature update deferral period (days): 3
Automatic update behavior: Auto install and reboot without end-user control
Option to pause Windows update: Disable
NEXT
Assignments - Include: Secure Workstations
NEXT
CREATE
```

Microsoft Defender for Endpoint Intune integration

Microsoft Intune can work together with Microsoft Defender for Endpoint to help prevent security breaches, and limit the impact of breaches.

To configure Microsoft Defender go to MEM admin center - Endpoint Security - Microsoft Defender for Endpoint - Configuring Microsoft Defender for Endpoint - Connect Microsoft Defender for Endpoint to Microsoft Intune in the Microsoft Defender Security Center

In Windows Defender security center:

```
Settings - Advanced features
Microsoft Intune connection: On
SAVE
In MEM - refresh and set Connect Windows devices version(20H2) 19042.450 and above to Windows Defe
nder ATP to On, then save.
```

Create the device configuration profile to onboard Windows devices

In MEM admin center - Endpoint Security - Endpoint detection and response - create a new policy with the following information.

```
Platform: Windows 10 and later
Profile type: Endpoint detection and response
CREATE
Name: PAW - Defender for Endpoint
NEXT
Configuration settings - Endpoint Detection and response:
Expedite telemetry reporting frequency - Enable for high risk devices
NEXT x2
Assignments: Secure Workstation group
NEXT
CREATE
```

Finish workstation profile hardening

To finish the hardening of the device download and execute the appropriate configurations depending on the security level of the workstation.

1. Enterprise: https://github.com/Azure/securedworkstation/blob/master/ENT/Readme.md

Clone the github repository to a local folder, then execute the following commands.

```
Install-Module AzureAD
Import-Module AzureAD -force
Set-ExecutionPolicy RemoteSigned

## Unblock the repo files from execution policy
Unblock-File -Path ../Import-ENT-DeviceConfiguration.ps1
Unblock-File -Path ../Import-ENT-DeviceCompliancePolicies.ps1
Unblock-File -Path ../Import-ENT-DeviceConfigurationADMX.ps1
```

Execute MasterScript-ENT.ps1

2. Specialized: https://github.com/Azure/securedworkstation/blob/master/SPE/Readme.md

Clone the github repository to a local folder, then execute the following commands.

```
Install-Module AzureAD
Import-Module AzureAD -force
```

```
Set-ExecutionPolicy RemoteSigned

## Unblock the repo files from execution policy
Unblock-File -Path ../Import-SPE-DeviceConfiguration.ps1
Unblock-File -Path ../Import-SPE-DeviceCompliancePolicies.ps1
Unblock-File -Path ../Import-SPE-DeviceConfigurationADMX.ps1
```

Execute MasterScript-SPE.ps1

3. Privileged: https://github.com/Azure/securedworkstation/blob/master/PAW/Readme.md

Clone the github repository to a local folder, then execute the following commands.

```
Install-Module AzureAD
Import-Module AzureAD -force
Set-ExecutionPolicy RemoteSigned

## Unblock the repo files from execution policy
Unblock-File -Path ../Import-PAW-DeviceConfiguration.ps1
Unblock-File -Path ../Import-PAW-DeviceCompliancePolicies.ps1
Unblock-File -Path ../Import-PAW-DeviceConfigurationADMX.ps1
```

Execute MasterScript-PAW.ps1

Table 6: Security levels per machine

Virtual machine	Security level
On-Prem-DC	Privileged
DC	Privileged
On-Prem-ADC	Privileged
ManagementVMs	Specialized
On-Prem-Client	Enterprise

Note
In a production environment there will be more servers/machines in the environment. It is up to the IT staff to determine a proper security level for these machines. Microsoft's privileged access security levels guide can be used as a reference for this.

6. Security rapid modernization plan

This plan builds on the foundation that was configured in Privileged Access Implementation to configure specific security controls in the environment.

Separate and manage privileged accounts

Emergency access accounts

These accounts ensure that you are not accidentally locked out of Azure AD in an emergency situation. The accounts are rarely used and highly damaging to an organization if compromised.

Requirements:

- Emergency access accounts should be cloud-only accounts and they should not be federated/synchronized from an on-premises environment.
- Emergency access accounts should not be associated with any individual user in the organization
- The authentication mechanism used for an emergency access account should be distinct from that used by your other administrative accounts
- The device or credential must not expire or be in scope of automated cleanup due to lack of use
- The emergency access account should permanently have the Global Administrator role
- At least one account should be excluded from MFA & Conditional Access Policies
- These accounts should be monitored and audited closely

Configuration:

Create 2 new users in the Azure portal - Users

```
Name: EmergencyBreakglass1
User name: Emergency1@stagesien.onmicrosoft.com
Groups: Emergency Breakglass
CREATE
NEW USER
Name: EmergencyBreakGlass2
User name: Emergency2@stagesien.onmicrosoft.com
Groups: Emergency Breakglass
CREATE
```

Important
Make sure to save the password to a secure location.

The Emergency Breakglass group has to be excluded from MFA and conditional access policies. Make sure to check that this is configured correctly. To check the MFA settings of a user go to Microsoft 365 Admin Center - users - configure MFA.

Enable Azure AD Privileged Identity Management

Azure AD Privileged Identity Management (PIM) discovers and secures privileged accounts. This service uses time-based and approval-based role activation, which mitigates the risk of excessive, unnecessary, or misused access permissions..

How does PIM work?

```
1. Start using Privileged Identity Management so that users are eligible for privileged roles.
2. When an eligible user needs to use their privileged role, they activate the role using Privileg
ed Identity Management.
```

```
3. The user can be required in settings to:
   1. Use multi-factor authentication
   2. Request approval for activation
   3. Provide a business reason for activation
4. After the user successfully activates their role, they'll have the role permissions for a set d
uration.
5. Administrators can view a history of all Privileged Identity Management activities in the audit
log. They can also further secure their Azure AD organizations and meet compliance using Privilege
d Identity Management features such as access reviews and alerts.
```

Configuration:

PIM will be activated for the following users for the following roles in my lab environment.

Table 7: Role allocations

User	Role
Phil	Global Administrator
Claire	Intune Administrator
Alex	Service Support Administrator

Note
These role allocations will be different per organization. A clear overview of the users and their role allocations should be documented.

```
In Azure AD - Users, select a user - assigned roles
Add an assignment
Select the role
NEXT
Assignment type: Eligible
Permanently eligible: Yes
ASSIGN
```

Users can request their roles to be activated through Azure AD - Privileged Identity Management - My roles. Here they can activate the roles they are eligible for, by doing this they will first have to prove their identity with MFA and fill out some other information such as the duration of the assignment.

This request will then have to be approved first before they have access to the role.

Note
In a production environment PIM rollout should be documented and configured while making use of Microsoft's deployment plan.

Identify and categorize privileged accounts (Azure AD)

Identify all roles and groups with high business impact that require a privileged security level. This is done to minimize the amount of people that require separate accounts and privileged access protection.

Configuration:

```
In Azure AD Privileged Identity Management (PIM), view the users who are in the following roles:

* Global administrator
* Privileged role administrator
* Exchange administrator
* SharePoint administrator
```

Separate accounts (On-premises AD accounts)

Secure on-prem privileged administrator accounts. This includes:

- Creating separate admin accounts for users who needs to conduct on-premises administrative tasks
- Deploying privileged access workstations for AD admins
- Creating unique local admin passwords for workstations and servers (Microsoft LAPS)

Configuration:

For every user that has administrative privileges, two accounts have to exist. One standard user account and one administrative account.

```
In AD create
* an account with administrative privileges for the user
* a standard user account without administrative user privileges
```

Microsoft Defender for Identity

Microsoft Defender for Identity combines on-prem signals with cloud insights to monitor, protect, and investigate events.

Configuration:

```
    Sign in to the defender for identity portal with a global administrator account or a user with
the Defender for Identity role: https://portal.atp.azure.com/
    Create a new instance
    Enter the AD information
    Download and install the MDI sensor package on the domain controllers using the generated acce
ss key
    Add the EmergencyBreakGlass users to the sensitive accounts list in settings - entity tags
```

Note
The configuration of Microsoft Defender can be fully customized to an organization's needs, in the lab environment only a baseline is configured.

Improve credential management experience

Implement and document self-service password reset and combined security information registration

Enable and configure SSPR and enable the combined security information registration experience.

Self-Service Password Reset enables users to reset their own passwords through pre-configured authentication information. The combined security information registration experience enable users to register and manage their security info for Multi-Factor Authentication and self-service password reset in a single experience

Configuration:

Self-Service Password Reset (SSPR)

```
To enable combined registration:
Azure Active Directory - User settings - Manage user feature preview settings
Set "Users can use the combined security information registration experience" to All
```

Protect admin accounts - Enable and require MFA / Passwordless for Azure AD privileged users

Enable and require MFA for admin accounts.

Configuration:

```
Enable MFA for all users in the following roles, this can be done through Microsoft 365 admin cent
er - manage MFA - filter on role:

* Global administrator
* Privileged Role administrator
* Exchange administrator
* SharePoint administrator

Require administrators to use passwordless sign-in methods such as FIDO2 security keys or
Windows Hello for Business in conjunction with unique, long, complex passwords
```

Block legacy authentication protocols for privileged user accounts

Legacy authentication protocols should be blocked because MFA cannot be enforced against them, which could create an entry point for attackers.

A list of legacy authentication protocols can be found here

Configuration:

```
Identify legacy authentication use in Azure AD - Sign-ins - filter on Client app: legacy auth prot
ocols

Block legacy authentication by creating a conditional access policy, in MEM - Endpoint security -
Conditional Access - Create new policy with the following info

Name: Block Legacy Authentication
Assign - Include: All users
Assign - Exclude: Emergency BreakGlass group
Conditions - Client apps: Set to yes and check all Legacy authentication clients
rant: Block access
Set policy to On
CREATE
```

Application consent process

Disable end-user consent to Azure AD applications. This prevents users from creating an organizational risk by giving consent to an app that can maliciously access organizational data.

Configuration:

```
Disable future user consent operations to any application in Azure AD - Users - User settings - Ma
nage how end users launch and view their application: Set "Users can allow apps to access their da
ta" to No

Set "Users can request admin consent to apps they are unable to consent to" to Yes and add the rol
es of Global administrator and Application administrator.

SAVE
```

With this configuration users cannot consent to any application, but administrators can on the user's behalf.

Clean up account and sign-in risks

Enable Azure AD Identity Protection, create a user risk and sign in risk policy and cleanup the risks that it discovers.

Configuration:

```
In Azure AD - Security - Identity Protection - User Risk Policy
Assignments - Users: All, but exclude group Emergency BreakGlass
Conditions - User risk: High
Controls - Access: Allow access and require password change
Enforce policy: On
SAVE

Sign-in risk policy
Assignments - Users: All, but exclude group Emergency BreakGlass
Conditions - Sign-in risk: Medium and above
Controls - Access: Allow access and require MFA
Enforce policy: On
SAVE
```

Admin workstations initial deployment

Privileged accounts, like global administrators, have dedicated workstations to perform administrative tasks from.

Configuration:

Configuration and hardening of workstations is described in Privileged access implementation

7. Role Based Access Control

Certain functionalities of Role Based Access Control (RBAC) have already been implemented in the previous chapters. In this chapter a proper overview of the users, roles, and groups in my lab environment including explanation and configurations will be listed.

Important
In a production environment many of these roles will already exist. Make sure every role has an appropriate security level and that Emergency BreakGlass accounts are configured.

Roles and assignments

Local Admin accounts are secured with LAPS to safeguard and rotate their passwords. These accounts should not be used to log in to machines unless in the case of an emergency. Logins to machines should always happen with domain accounts.

Table 8: Role assignments

Username*	Roles	Member of groups	Security level
sien	Global administrator, Intune administrator	High level admins	Privileged
EmergencyBreakGlass1	Global administrator	Emergency BreakGlass	Privileged
EmergencyBreakGlass2	Global administrator	Emergency BreakGlass	Privileged
Enterprise-Admin	Enterprise Admin	High level admins	Privileged
Domain-Admin	Domain Admin	High level admins	Privileged
ADC-Admin	Server Operator		Privileged
WAC-Admin	Server Operator		Privileged
IT-Manager	Account Operator	IT, Management	Specialized
Finance-Manager	Account Operator	Finance, Management	Specialized
Marketing-Manager	Account Operator	Marketing, Management	Specialized
Phil**	Domain user	IT	Enterprise
Claire	Domain user	IT	Enterprise
Haley	Domain user	IT	Enterprise
Alex**	Domain user	Marketing	Enterprise
Luke	Domain user	Marketing	Enterprise
Cameron**	Domain user	Finance	Enterprise
Mitchell	Domain user	Finance	Enterprise

*All usernames are in the @stagesien.onmicrosoft.com domain

**These users have access to the management accounts of their respective departments.

Accounts with administrative privileges should only be used to perform administrative actions. Regular user actions should be performed on general user accounts.

Policies

Role based access control can be used to configure the specific access per user per machine in the Windows Admin Center portal. The following roles are supported and can be applied per user per machine. Role Based Access Control is available for Servers and HyperV clusters but not for client computers.

1. **Administrators**: Allows users to use most of the features in Windows Admin Center without granting them access to Remote Desktop or PowerShell. This role is good for “jump server” scenarios where you want to limit the management entry points on a machine.
2. **Readers**: Allows users to view information and settings on the server, but not make changes.
3. **Hyper-V Administrators**: Allows users to make changes to Hyper-V virtual machines and switches, but limits other features to read-only access. (This role will not be used in my lab environment)

Configuration:

```
Open WAC with a global administrator account and connect to the target machine.
In Settings - Role-based access control click on Apply

This starts a process that configures the remote machine to support RBAC, can take up to 10 minute
s and may restart the session.

After this process is completed do the following:
In Local Users and Groups - Groups add users to the appropriate groups.
```

The following table provides an overview of the RBAC assignment in my lab environment per machine.

Table 9: RBAC per machine

Machine name	Administrators	Readers
On-Prem-ADC	Enterprise Admins, Domain Admins, ADC-Admin	WAC-Admin
ManagementVMs	Enterprise Admins, Domain Admins, WAC-Admin	ADC-Admin

Important

Cannot configure RBAC on Domain Controllers with custom roles because the local users and groups tab is disabled for these machines. Roles like “Domain administrator” and “Enterprise administrator” are however recognized, and thus only the users with these roles will have rights on the DCs, everyone else is denied access.

Enterprise and domain administrators have full access to all machines in the domain.

To automatically configure RBAC on multiple systems, instead of system per system like the method above, the following configurations can be made.

```
## Download the configuration package, change the following variable with the address of the WAC g
ateway
$WindowsAdminCenterGateway = 'https://managementvmcluster.stagesien.com'
Invoke-RestMethod -Uri "$WindowsAdminCenterGateway/api/nodes/all/features/jea/endpoint/export" -Me
thod POST -UseDefaultCredentials -OutFile "~\Desktop\WindowsAdminCenter_RBAC.zip"

Update InstallJeaFeature.ps1 file to match your desired configuration for the RBAC endpoint. Then
execute the following script.
```

```
## Copy the needed files to the target servers, and run the configuration script
$ComputersToConfigure = 'ManagementVM0', 'ManagementVM1', 'On-Prem-ADC', 'On-Prem-DC', 'DC'

$ComputersToConfigure | ForEach-Object {
    $session = New-PSSession -ComputerName $_ -ErrorAction Stop
    Copy-Item -Path "~\Desktop\WindowsAdminCenter_RBAC\JustEnoughAdministration\" -Destination "$e
nv:ProgramFiles\WindowsPowerShell\Modules\" -ToSession $session -Recurse -Force
    Copy-Item -Path "~\Desktop\WindowsAdminCenter_RBAC" -Destination "$env:TEMP\WindowsAdminCenter
_RBAC" -ToSession $session -Recurse -Force
    Invoke-Command -Session $session -ScriptBlock { Import-Module JustEnoughAdministration; & "$en
v:TEMP\WindowsAdminCenter_RBAC\InstallJeaFeature.ps1" } -AsJob
    Disconnect-PSSession $session
}
```

Table 10: Workstations security levels

Machine name	Roles	Member of groups	Security level
On-Prem-DC	Domain Controller	Domain Controllers	Privileged
DC	Domain Controller	Domain Controllers	Privileged
On-Prem-ADC	Domain Computer	Managed Devices	Privileged
ManagementVMs	Domain Computer	Managed Devices	Specialized
On-Prem-Client	Domain Computer	Managed Devices	Enterprise

By default only domain and enterprise admins can RDP to a workstation in the domain, to allow the other accounts to access the machine over RDP they have to be added the the local Remote Desktop Users group. Because everything in my lab is virtual, I will create different RDP groups to allow RDP access to different machines.

Domain accounts could also be defined as local administrators per server, which would grant them RDP access to the server. However these users would then have full administrator rights on said server, not all users that need to RDP to a server need to have full administrator rights. Working with RDP groups gives the users RDP access and no additional privileges on top of the privileges they already have with their role assignments.

Important
In an environment with physical machines, physical access to these machines will also have to be secured.

Table 11: RDP groups

Group name	Members	Target machines
Allow_RDP	IT, Marketing, Finance	On-Prem-Client
RDP_ADC	ADC-Admin	On-Prem-ADC
RDP_Management	WAC Admin, Management	ManagementVMs

Note
Domain administrators can RDP to servers in the domain by default, so they do not need to be added to an RDP group.

The connection port of RDP should also be secured. This is explained later in the document.

Configuration

The following PowerShell script creates the users, groups, and assigns the users to the correct groups.

```
## Create RDP groups
New-ADGroup -Name "Allow_RDP" -GroupCategory Security -GroupScope Global -DisplayName "Allow_RDP"
-Path "CN=Users,DC=stagesien,DC=Com" -Description "Members of this group can RDP to the Client VM"
New-ADGroup -Name "RDP_ADC" -GroupCategory Security -GroupScope Global -DisplayName "RDP_ADC" -Pat
h "CN=Users,DC=stagesien,DC=Com" -Description "Members of this group can RDP to the ADC"
New-ADGroup -Name "RDP_Management" -GroupCategory Security -GroupScope Global -DisplayName "RDP_Ma
nagement" -Path "CN=Users,DC=stagesien,DC=Com" -Description "Members of this group can RDP to the
ManagementVMs"

## Create user groups
New-ADGroup -Name "High level admins" -GroupCategory Security -GroupScope Global -DisplayName "Hig
h level admins" -Path "CN=Users,DC=stagesien,DC=Com" -Description "Members of this group have have
highly privileged abilities"
New-ADGroup -Name "Server Operators" -GroupCategory Security -GroupScope Global
New-ADGroup -Name "Management" -GroupCategory Security -GroupScope Global -DisplayName "Management
" -Path "CN=Users,DC=stagesien,DC=Com" -Description "Members of this group have management privile
ges"
New-ADGroup -Name "IT" -GroupCategory Security -GroupScope Global -DisplayName "IT" -Path "CN=User
s,DC=stagesien,DC=Com" -Description "Members of this group are part of the IT team"
New-ADGroup -Name "Marketing" -GroupCategory Security -GroupScope Global -DisplayName "Marketing"
-Path "CN=Users,DC=stagesien,DC=Com" -Description "Members of this group are part of the marketing
team"
New-ADGroup -Name "Finance" -GroupCategory Security -GroupScope Global -DisplayName "Finance" -Pat
h "CN=Users,DC=stagesien,DC=Com" -Description "Members of this group are part of the finance team"


## Create new users
New-ADUser -Name Enterprise-Admin -UserPrincipalName "Enterprise-Admin@stagesien.onmicrosoft.com"
-Enabled:$True -Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Domain-Admin  -UserPrincipalName "Domain-Admin@stagesien.onmicrosoft.com" -Enable
d:$True -Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name ADC-Admin -UserPrincipalName "ADC-Admin@stagesien.onmicrosoft.com" -Enabled:$True
-Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name WAC-Admin -UserPrincipalName "WAC-Admin@stagesien.onmicrosoft.com" -Enabled:$True
-Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name IT-Manager -UserPrincipalName "IT-Manager@stagesien.onmicrosoft.com" -Enabled:$Tr
ue -Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Finance-Manager -UserPrincipalName "Finance-Manager@stagesien.onmicrosoft.com" -E
nabled:$True -Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Marketing-Manager -UserPrincipalName "Marketing-Manager@stagesien.onmicrosoft.com
" -Enabled:$True -Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Phil -UserPrincipalName "Phil@stagesien.onmicrosoft.com" -Enabled:$True -Accountp
assword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Claire -UserPrincipalName "Claire@stagesien.onmicrosoft.com" -Enabled:$True -Acco
untpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Hailey -UserPrincipalName "Hailey@stagesien.onmicrosoft.com" -Enabled:$True -Acco
untpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Alex -UserPrincipalName "Alex@stagesien.onmicrosoft.com" -Enabled:$True -Accountp
assword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
```

```
New-ADUser -Name Luke -UserPrincipalName "Luke@stagesien.onmicrosoft.com" -Enabled:$True -Accountp
assword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Cameron -UserPrincipalName "Cameron@stagesien.onmicrosoft.com" -Enabled:$True -Ac
countpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)
New-ADUser -Name Mitchell -UserPrincipalName "Mitchell@stagesien.onmicrosoft.com" -Enabled:$True -
Accountpassword (ConvertTo-SecureString "XXXX" -AsPlainText -Force)

## Assign roles to users
Add-ADGroupMember -Identity "Enterprise Admins" -Members sien, Enterprise-Admin
Add-ADGroupMember -Identity "Domain Admins" -Members Domain-Admin
Add-ADGroupMember -Identity "Server Operators" -Members ADC-Admin, WAC-Admin
Add-ADGroupMember -Identity "Account Operators" -Members IT-Manager, Finance-Manager, Marketing-Ma
nager


## Add users to user groups
Add-ADGroupMember -Identity "High level admins" -Members sien, Enterprise-Admin, Domain-Admin
Add-ADGroupMember -Identity "Management" -Members IT-Manager, Finance-Manager, Marketing-Manager
Add-ADGroupMember -Identity "IT" -Members Phil, Claire, Hailey
Add-ADGroupMember -Identity "Marketing" -Members Alex, Luke
Add-ADGroupMember -Identity "Finance" -Members Cameron, Mitchell

## Add users to RDP groups
Add-ADGroupMember -Identity "Allow_RDP" -Members Phil, Claire, Hailey, Alex, Luke, Cameron, Mitche
ll
Add-ADGroupMember -Identity "RDP_ADC" -Members ADC-Admin, sien, Enterprise-Admin, Domain-Admin
Add-ADGroupMember -Identity "RDP_Management" -Members WAC-Admin, IT-Manager, Marketing-Manager, Fi
nance-Manager
```

On the machines, the groups can be added to the remote desktop users groups with the following script. This script can be executed remotely through Remote PowerShell connections.

```
$Username = "stagesien\sien"
$SecurePassword = ConvertTo-SecureString "XXXX" -AsPlainText -Force
$credentials = New-Object System.Management.Automation.PSCredential($Username, $SecurePassword)


## On On-Prem-Client
$conn = New-PSSession -ComputerName On-Prem-Client.stagesien.com -Credential (Get-Credential $cred
entials)

Invoke-Command -Session $conn -ScriptBlock {
net localgroup "remote desktop users" /add "Allow_RDP"
net localgroup "remote desktop users" /add "RDP_ADC"
net localgroup "remote desktop users" /add "RDP_Management"}


## On On-Prem-ADC
$conn = New-PSSession -ComputerName On-Prem-ADC.stagesien.com -Credential (Get-Credential $credent
ials)
```

```
Invoke-Command -Session $conn -ScriptBlock {
net localgroup "remote desktop users" /add "RDP_ADC"}

## On the ManagementVMs
$conn = New-PSSession -ComputerName ManagementVM0.stagesien.com -Credential (Get-Credential $crede
ntials)

Invoke-Command -Session $conn -ScriptBlock {
net localgroup "remote desktop users" /add "RDP_Management"}
```

8. Security and monitoring

This chapter explains the configurations made to deploy password management, security- and monitoring solutions in the environment.

Local Admin Password Management - LAPS

Microsoft Local Administrator Password Solution (LAPS) is a password manager that utilizes Active Directory to manage and rotate passwords for local administrator accounts in an Active Directory environment. This solution forces all local admin accounts to have unique, complex passwords, which greatly improves security. The management of these passwords is done through Active Directory.

Local administrator accounts should not be used unless absolutely necessary in case of an emergency, in all other situations a domain account should be used.

Configure LAPS manager

First download LAPSx64.msi from https://www.microsoft.com/en-us/download/details.aspx?id=46899 and install it on the On-Prem-DC (or on another DC). Install all the Management tools, but not the AdmPwd GPO extension.

Raise the Domain functional level of the environment through: AD Domains and trusts, right click on domain, raise domain functional level (2012 -> 2016).

Then import the AdmPwd module and update the AD schema with the following commands:

```
Import-Module AdmPwd.PS
Update-AdmPwdADSchema
```

Make sure that all the computers are in the OU Managed Devices.

Check rights of users/groups in Managed Devices OU:

```
Find-AdmPwdExtendedRights –Identity "Managed Devices"
```

Give computers ability to update ms-Mcs-AdmPwd and ms-Mcs-AdmPwdExpirationTime attributes in AD:

```
Set-AdmPwdComputerSelfPermission –Identity "Managed Devices"
```

Set up clients

Create a new GPO (LAPS GPO) in the Group Policy Management console and link it to the Managed Devices OU. Then go to Computer configuration, policies, administrative templates, LAPS. Enable the password management and password settings policies.

In the same GPO (or a new one) create a new policy, through computer configuration, policies, software settings, software installation, new package. Make sure the previously downloaded .msi file is available in a shared network drive and add it to the package, make the deployment method the “assigned” type.

When executing `echo y | gpupdate /force` on the computers, they will restart and download this package. Back on the On-Prem-DC in the LAPS shortcut, the local admin passwords and expiry times can be viewed.

Note
In a production environment all devices should not necessarily be in the “Managed Devices” OU, LAPS can instead be configured on different custom OUs.

Azure AD Self-Service Password Reset

Azure Active Directory Self-Service Password Reset is a service that gives users the ability to reset their password without the need for an administrator or help desk involvement. If Azure AD locks a user's account or they forget their password, they can follow prompts to unblock themselves and get back to work. This ability reduces help desk calls and loss of productivity when a user can't sign in to their device or an application.

SSPR has to be configured in the Azure AD tenant, and can only be applied to cloud users. Users that are created in the on-premise AD and then synced to the Azure AD cloud can also make use of this feature.

The feature can be enabled for all users in in the tenant, or for a selected subset of users in a group. In this environment it was enabled for all users of a group.

Create users and group

Currently SSPR can only be enabled for one selected group of users, however nested groups can still be created in this group. In this environment the group called "SSPR" has the SSPR feature enabled for them. All the members of the groups IT, Marketing, and Finance are in this SSPR group and can thus reset their own passwords.:

```
## Create new group "SSPR"
New-ADGroup -Name "SSPR" -GroupCategory Security -GroupScope Global -DisplayName "SSPR" -Path "CN=
Users,DC=stagesien,DC=Com" -Description "Members of this group have SSPR enabled for them"


## Add users to group
Add-ADGroupMember -Identity "SSPR" -Members Phil, Claire, Hailey, Alex, Luke, Cameron, Mitchell
```

Enable SSPR

Self-Service Password Reset has to be enabled for a group through the Azure portal, only global administrator users can configure this.

From the left pane select Password reset, click "selected" and select the SSPR group.

Authentication methods can be configured through the "authentication methods" page.

Before users can unlock their account or reset a password, they must first register their contact information. This contact information is used for the different authentication methods. Administrators can manually provide contact information for users, or the users can go to a registration portal themselves to enter this information.

By default notifications are enabled. If an SSPR event happens for a user account, the user will receive an email about this.

When a user logs in to Azure services they will be redirected to a Microsoft site where they can choose their authentication method and enter information.

Important
If password write back was not configured in the AD sync configuration then users will not be able to reset their own passwords, but instead will be met with an error.

To enable password writeback, the Azure AD tenant must have at least an Azure AD Premium P1 license enabled.

Azure Monitor

Azure Monitor is a monitoring service on the Microsoft Azure platform. This tool serves as a solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments. This information helps you understand how your resources are performing and proactively identify issues affecting them and the resources they depend on.

This chapter will describe how Azure monitor is implemented in the environment.

A Log Analytics workspace is needed to collect data and analyze it.

Some features of Azure monitor are already enabled by default and do not need to be configured manually. These features are: Azure Active Directory Logs, Activity Logs, and Platform metrics. These logs can also be sent to a Log Analytics workspace to analyze the events.

Once a workspace has been configured, each virtual machine can be enabled by installing the Log Analytics agent and Dependency agent on these VMs.

Create a workspace

A Log Analytics workspace can be deployed through Terraform. The code for this is included in shared-env.tf.

```
## Create a log analytics workspace
resource "azurerm_log_analytics_workspace" "MonitorWorkspace" {
  name                = "MonitorWorkspace"
  location            = azurerm_resource_group.rg_Security.location
  resource_group_name = azurerm_resource_group.rg_Security.name
  sku                 = "PerGB2018"
  retention_in_days   = 30
}
```

Diagnostics settings

Diagnostic settings define where resource logs should be sent for a particular resource.

These settings have to be defined per resource under Monitoring/ Diagnostic Settings. Enable guest-level monitoring on this page as well.

For Azure Active Directory diagnostic settings also have to be defined. In Azure AD - Monitoring - Diagnostics Settings create a new setting that sends all logs to the previously created Log analytics workspace.

Make sure all VMs are connected to the correct Log Analytics workspace. This can be checked under Workspace Data Sources/ Virtual machines at the right side of the log analytics workspace. An activity log can also be connected, through Workspace Data Sources/ Azure Activity Log.

The logging agent can be downloaded on a VM with the following PowerShell script:

```
$url = "https://go.microsoft.com/fwlink/?LinkId=828603"
$outpath = "C:/log.exe"
Invoke-WebRequest -Uri $url -OutFile $outpath
```

Diagnostics data of the Azure AD tenant can also be sent to this Log Analytics workspace. This can be configured through Monitoring/ Diagnostics Settings in the Azure AD tenant page.

Queries

Queries can be executed on the data that a Log Analytics Workspace collects. This can be done through the “logs” tab on the page of the workspace. Many pre-meade queries already exist here, and you only have to click on them, and on execute to run them.

Queries that are used a lot can be added to “favorites”, this will list them at the top of the query page and thus will make them easier to find.

Active Directory event logs

Through Active Directory audit policies can be configured to collect event logs of machines. These event logs provide information about who did what action at what time, and wether or not this action was successful.

Configuration to audit domain controllers:

```
In Active Directory - Group Policy Management select the local domain controllers policy.

Go to Windows Settings - Security Settings - Local Policies - Audit policy

In this window different policies can be enabled. Every policy can be enabled to log entries in ca
se of success, in case of failure, or in both cases.
```

The machines in the managed devices OU can also be audited by doing the same steps as above but instead of working in the Default DC policy, create a new GPO that's linked to the Managed Devices OU.

Event logs can then be viewed in the Security logs of the event viewer, this can be done through the Windows Admin Center interface.

Dummy event logs can be created with the following command to test if the logging events come through properly in the monitoring interface.

```
eventcreate /ID 999 /L Application /SO DummySource /T Warning /D "This is a test message."
```

Important
Auditing policies on a system should be configured based on the security level of the system, with privileged systems auditing most events, and enterprise systems auditing only a baseline of events.

Azure AD Connect Health

Azure Active Directory Connect Health provides monitoring for on-premises identity infrastructure. This is achieved by installing an agent on the domain controllers in an environment. The data can be monitored in the Azure AD Connect Health portal.

This tool will be used to monitor and gain insights into the identity infrastructure in the environment.

In my lab environment I installed the agent on both of my domain controllers by downloading it with the following script.

Important
In a production environment all domain controllers should be configured with this agent.

```
## Download the Azure AD Connect Health agent
$url = "https://go.microsoft.com/fwlink/?LinkID=820540"
$outpath = "C:/AzureADConnectHealth.exe"
Invoke-WebRequest -Uri $url -OutFile $outpath
```

Before executing the previously downloaded file, add the following websites to the trusted sites list of Internet Explorer, else you will not be able to log into Azure AD.

- https://login.microsoftonline.com
- https://secure.aadcdn.microsoftonline-p.com
- https://login.windows.net
- https://aadcdn.msftauth.net

Then execute C:/AzureADConnectHealth.exe and log in with a domain administrator account.

It will take a few minutes before the domain controllers show up in the Azure AD Connect Health portal.

Connectivity to the Microsoft endpoints can be tested with the following command.

```
Test-AzureADConnectHealthConnectivity -Role ADDS
```

The agents collect data about the domain and sites, including authentication methods and login attempts. Different counters can be selected to monitor in Azure AD Connect Health - AD DS Services - clicking on the domain name - Performance Monitors Collection.

AD Domain Services will also generate alerts in case an error happens on a domain controller.

Note
The configuration above requires outbound connectivity from the domain controllers to send the generated alerts, this might not be preferred in a production environment. As an alternative port mirroring or a network TAP can be configured.

Azure Sentinel

Azure Sentinel is Microsoft's cloud-native security information and event manager (SIEM) platform. This solution uses built-in AI to help analyze large volumes of data.

This solution can be used to collect the data we've previously sent to the log analytics workspace and visualize it in workbooks. Alerts can also be configured to notify admins when a certain event takes place, such as suspicious activity or when a threat is detected. Based on these alerts, incidents and automated responses can be configured.

Different connector sources can be configured in Azure sentinel to connect to different data sources, such as Azure AD data, Windows event logs, and much more.

Note
For my lab solution Azure Sentinel is a fitting solution, many existing organizations that wish to migrate to the cloud already use different SIEM/SOAR solutions. These organizations can choose to move forward with their previously used SIEMs/SOARs or switch over to Azure Sentinel based on their own preferences.

To start, an Azure sentinel has to be created for our previously created log analytics workspace.

In the left pane - Data connectors add the following connectors:

- Azure AD
- Azure AD Identity Protection
- Azure Activity
- Security Events

For the Azure AD Identity protection connector enable incident creation for alerts generated by this service in the connection settings.

For the Security events connector make sure the agent is installed on all machines you want to monitor. The events to stream can be configured based on the monitoring needs. "Minimal" will only collect a small set of events that might indicate potential threats, while "common" will collect more events and thus provides more auditing possibilities.

Workbooks

Workbooks are visualizations of queries that are executed on a set of collected data. Many workbook templates are available to choose from, or a custom workbook can be created. The visualizations in workbooks are created with the Kusto Query Language (KQL).

Important
In a production environment an organization will have to decide what data is important for them to be monitored. Based on these design decisions, Kusto queries and workbooks can be configured.

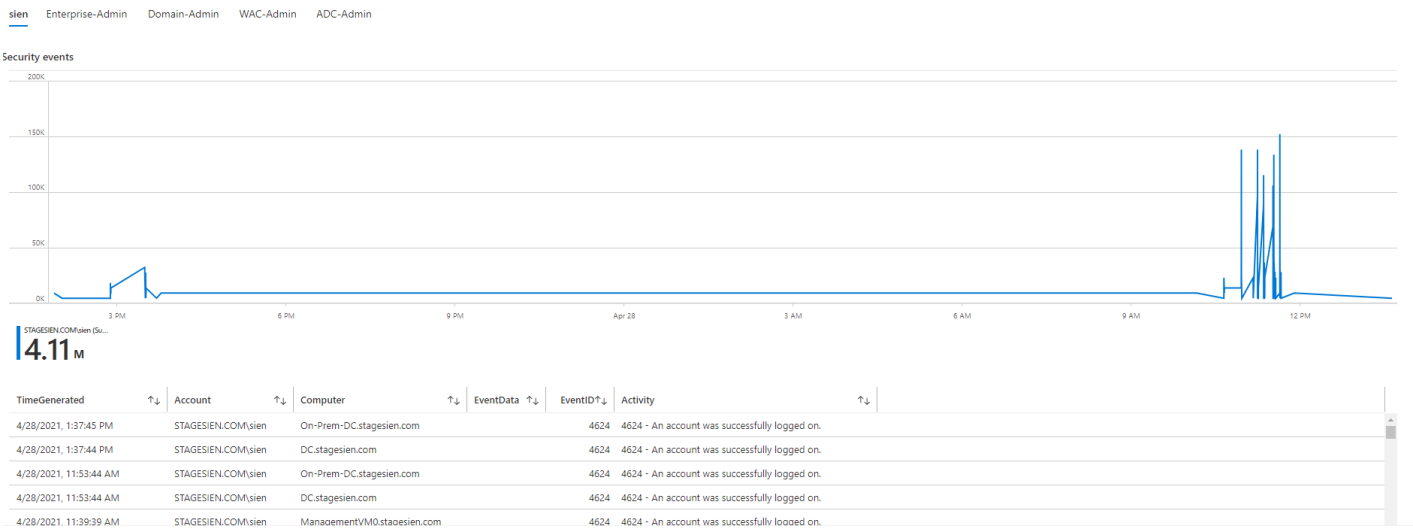
For my lab environment I've created a custom workbook named "Privileged user activity workbook" which displays privileged user activities and suspicious activities such as password reset attempts or failed logins. The workbook provides an easy way to monitor the data to keep a close eye on privileged user accounts.

The queries I used to create this workbook are:

```
// Separate table for EmergencyBreakGlass accounts
SecurityEvent
| where ((Account like "STAGESIEN.COM\\EmergencyBreakGlass1") or (Account like "STAGESIEN.COM\\Eme
rgencyBreakGlass2"))
| order by TimeGenerated
| project TimeGenerated , Account , Computer , EventData , EventID , Activity

// To display security events per user (configured per privileged user, displayed in grid and time
chart)
SecurityEvent
| where (Account like "STAGESIEN.COM\\sien")
| order by TimeGenerated
| project TimeGenerated , Account , Computer , EventData , EventID , Activity

// Password change attempts
SecurityEvent
| where EventID in (4723, 4724)
| order by TimeGenerated
| project TimeGenerated , TargetAccount , Computer , EventID , Activity

// Privileged users failed logons
SecurityEvent
| where EventID == 4625
| project TimeGenerated , Account , Computer  , EventID , Activity
```

All of the above queries are linked to a dropdown menu to select the time range. The security event logs are displayed in tabs, one tab per privileged user, as can be seen in the image below.

Figure 1: Screenshot of security events tabs



For the EmergencyBreakGlass users there is a different table at the top, because this group should not be used unless when there is an emergency situation, so their activity should be monitored even closer.

I also made a server auditing workbook that keeps track of the security events of servers in the environment. The following query was used to create this workbook. This query is used for every server in the machine. They are also displayed in tabs like in the privileged user activity workbook.

```
// Display security events per machine, in grid and in time chart
SecurityEvent
| where (Computer like "On-Prem-DC")
| order by TimeGenerated
| project TimeGenerated , Account , Computer , EventData , EventID , Activity
```

Incidents

Using Kusto queries, different incident creation rules can be configured. For example an incident can be created every time a privileged user logs in, or when malware is detected. Incidents can provide insights in when a certain action/event has taken place. This is useful to keep an eye on privileged accounts, and make sure no suspicious behaviour is happening in the environment.

Queries are fully customisable so there are many possibilities in which events you want to connect to an incident. The incidents can be used later on to create automated responses.

Important
Automated responses can greatly improve security by for example locking suspicious accounts, shutting down infected machines, etc. These automated responses are quick and thus reduce the time an issue has to spread further in the environment.

In a production environment incidents will have to be configured following pre-made design decisions. These design decisions should clarify which incidents to create in case of which specific events, and how to respond to these incidents.

For my lab environment I've configured incident creation rules for each of the following cases in which a specific event takes place:

- High volume of traffic to uncommon domains
- EmergencyBreakGlass user login
- Privileged user login
- Downtime of critical systems (DCs)
- Errors on critical systems (DCs)

Every incident can be configured with a security level to set a priority system in place.

These incidents are made with the following Kusto queries. These can be configured in Azure Sentinel - Analytics - Create scheduled query rule, to create custom query rules. Or Azure Sentinel - Analytics - Create Microsoft incident creation rule, to create incidents from alerts generated in other security solutions such as Microsoft Endpoint Protection.

```
// High volume traffic to uncommon domains
// This query identifies domains receiving an uncommon amount of data volume. This could be an ind
ication of adversary attempts to steal and exfiltrate data.
let isInternal = (url_hostname:string){url_hostname endswith ".local" or url_hostname endswith ".l
an" or url_hostname endswith ".home"};
    // used to exclude internal traffic
let top1M =  (externaldata (Position:int, Domain:string) [@"http://s3-us-west-1.amazonaws.com/umbr
ella-static/top-1m.csv.zip"]  with (format="csv", zipPattern="*.csv"));
    // fetch the alexa top 1M domains
let top2ndLevelDomain=top1M
    | extend Domain = tolower(extract("([^.]*).{0,7}$", 1, Domain))
    | distinct Domain;
let rareDomainTraffic = NetworkSessions
    | where isnotempty(UrlHostname) and not(isInternal(UrlHostname))
    | extend SndLevelDomain=tolower(extract("([^.]*).{0,7}$", 1, UrlHostname))
    | where SndLevelDomain !in (top2ndLevelDomain)
    | summarize BytesSent=sum(SrcBytes) by SndLevelDomain, UrlHostname;
rareDomainTraffic | summarize TotalBytes=sum(BytesSent) by SndLevelDomain
| join kind=innerunique
    rareDomainTraffic
        on SndLevelDomain
| sort by TotalBytes desc
```

The above rule will detect when a there is a suspiciously large volume of traffic to uncommon domains. This could be an indication of an attempt to steal data. When the result of the this query is 1 result or more, an incident will be generated. This query will run every hour and check for results in the last 65 minutes.

The UrlHostName parameter is mapped to the Hostname entity. By mapping this parameter to an entity, it can be used as input later on in an automated response.

```
// EmergencyBreakGlass user login
SecurityEvent
| where EventID == 4624 and ((Account like "STAGESIEN.COM\\EmergencyBreakGlass1") or (Account like
"STAGESIEN.COM\\EmergencyBreakGlass2"))
| project TimeGenerated , Account , Computer , EventData , EventID , Activity, TargetUserName
```

The query above detects when an Emergency BreakGlass user logs in, when these results are 1 or higher a critical incident will be created. The Emergency accounts should only be used in case of an emergency. This incident creation rule is configured to execute the query every 5 minutes and check for results for the last 8 minutes, if this result is higher than 0 it will create an incident.

Note

The query will check for results in the last 8 minutes over a 5 minute interval because there often is a time delay for logs to enter the workspace. It is better to receive an alert double than to not receive an alert at all.

The above incident creation rule uses the TargetUserName variable as an entity to map the user for who the login was detected to an Azure AD user. This entity is used later to automate the response for the incident.

```
// Privileged user login
SecurityEvent
| where EventID == 4624 and ((Account like 'STAGESIEN.COM\\Enterprise-Admin') or (Account like 'ST
AGESIEN.COM\\Domain-Admin'))
| project TimeGenerated , Account , Computer , EventData , EventID , Activity, TargetUserName
```

The above rule will create a medium incident when one of the listed accounts logs in. These accounts are priviliged user accounts and will normally not be used regularly, they are only used when a high level of privilege is needed (like when a new domain in an AD forest has to be created). This query will execute every 10 minutes and check for results in the last 13 minutes. Incidents created by this rule will be grouped together to maintain a proper overview.

The above incident creation rule uses the TargetUserName variable as an entity to map to an Azure AD user. This entity is used later to automate the response for the incident.

Important
For automation purposes, don't query global administrator accounts in this incident creation rule. Query these users in a different rule, to which a different automated response will be connected in later steps.

```
Heartbeat
| summarize LastHeartbeat=max(TimeGenerated) by Computer
| where LastHeartbeat < ago(5m) and ((Computer == 'On-Prem-DC.stagesien.com') or (Computer == 'DC.
stagesien.com'))
| extend Machine = iff(Computer == 'On-Prem-DC.stagesien.com', 'On-Prem-DC', 'DC')
| project Machine, LastHeartbeat
```

The query above will check when a machine sent its last heartbeat event, if this was more than five minutes ago then it will assume that the machine is down. This is checked for critical machines such as the domain controllers, which should always remain in an online state. If one of these machines is down, a critical alert will be generated. Incidents created by this rule will be grouped together to maintain a proper overview.

This query is executed every 15 minutes and will look for results of the last 18 minutes.

The above incident creation rule uses the Machine variable as an entity to map the hostname of the machine that went offline. This entity is used later to automate the response for the incident.

```
union Event, Syslog
| where ((Computer == '_On-Prem-DC') or (Computer == '_DC'))
| extend Machine = iff(Computer == '_On-Prem-DC', 'On-Prem-DC', 'DC')
| where EventLevelName == "Error"
| project TimeGenerated, Machine, Source, EventLog, EventLevel, EventLevelName, EventID, RenderedD
escription
```

The above incident creation rule will check if any errors are detected on critical systems. These errors are retrieved from the security events from the machines. This query will be executed every 15 minutes and look for results of the last 18 minutes, like the Critical System downtime incident creation rule.

The above incident creation rule uses the Machine variable as an entity to map the hostname of the machine that had the error. On top of this, a few custom variables are passed along to provide more information about the error that was

detected. These variables are: EventID, EventLog, RenderedDescription, and Source. These parameters are used later to automate the response for the incident.

Automation

Automation rules can be configured to automate incident configuration (like assigning an incident to the right IT staff), run playbooks when an incident takes place, trigger playbooks for Microsoft providers, and apply incident suppression.

Playbooks are collections of procedures that can be run automatically from Azure Sentinel in response to an incident or an alert. These playbooks can help automate your response to incidents/alerts.

When an Azure Sentinel automation rule runs a playbook, it uses a special Azure Sentinel service account specifically authorized for this action. The use of this account (as opposed to your user account) increases the security level of the service.

In order for an automation rule to run a playbook, this account must be granted explicit permissions to the resource group where the playbook resides. At that point, any automation rule will be able to run any playbook in that resource group.

These permissions can be configured in Azure Sentinel - Settings - Settings - Playbook permissions.

Important
The user who assigns these permissions must have owner permissions on the targeted resource group.

For my lab environment I have created a few different playbooks to automatically respond to incidents. The first playbook will be executed when an Emergency BreakGlass account login is detected, and will automatically send an email to an administrator.

A playbook is basically an Azure Logic app, and can thus be deployed through Terraform with the following code, which creates an empty Logic App.

```
resource "azurerm_logic_app_workflow" "BreakGlassUser" {
  name                = "BreakGlassUser"
  location            = azurerm_resource_group.rg_Security.location
  resource_group_name = azurerm_resource_group.rg_Security.name
}
```

When this logic app is created it can be configured with the specific tasks that have to happen after an incident is triggered.

I’ve configured my playbook with the following json code:

```
{
    "definition": {
        "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-
01/workflowdefinition.json#",
        "actions": {
            "Alert_-_Get_accounts": {
                "inputs": {
                    "body": "@triggerBody()?['object']?['properties']?['relatedEntities']",
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "method": "post",
                    "path": "/entities/account"
```

```
                },
                "runAfter": {
                    "For_each_2": [
                        "Succeeded"
                    ]
                },
                "type": "ApiConnection"
            },
            "For_each": {
                "actions": {
                    "Get_user_2": {
                        "inputs": {
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['azuread_1']['connectionI
d']"
                                }
                            },
                            "method": "get",
                            "path": "/v1.0/users/@{encodeURIComponent(concat(items('For_each')?['N
ame'], '@', 'stagesien.onmicrosoft.com'))}"
                        },
                        "runAfter": {},
                        "type": "ApiConnection"
                    },
                    "Send_an_email_(V2)": {
                        "inputs": {
                            "body": {
                                "Body": "<p>The following account has been detected in a suspicous
login: @{body('Get_user_2')?['userPrincipalName']}<br>\n<br>\nDisable the user account here: https
://portal.azure.com/#blade/Microsoft_AAD_IAM/UserDetailsMenuBlade/Profile/userId/@{body('Get_user_
2')?['id']}</p>",
                                "Importance": "High",
                                "Subject": "Suspicious login detected - @{items('For_each')?['Name
']}",
                                "To": "sien.van-broekhoven@hpe.com"
                            },
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['office365']['connectionI
d']"
                                }
                            },
                            "method": "post",
                            "path": "/v2/Mail"
                        },
                        "runAfter": {
                            "Get_user_2": [
                                "Succeeded"
                            ]
                        },
                        "type": "ApiConnection"
                    }
                },
                "foreach": "@body('Alert_-_Get_accounts')?['Accounts']",
                "runAfter": {
```

```
                    "Alert_-_Get_accounts": [
                        "Succeeded"
                    ]
                },
                "type": "Foreach"
            },
            "For_each_2": {
                "actions": {
                    "Alert_-_Get_incident": {
                        "inputs": {
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['azuresentinel']['connect
ionId']"
                                }
                            },
                            "method": "get",
                            "path": "/Cases/@{encodeURIComponent(items('For_each_2')?['properties'
]?['systemAlertId'])}/@{encodeURIComponent(triggerBody()?['workspaceInfo']?['SubscriptionId'])}/@{
encodeURIComponent(triggerBody()?['workspaceId'])}/@{encodeURIComponent(triggerBody()?['workspaceI
nfo']?['ResourceGroupName'])}"
                        },
                        "runAfter": {},
                        "type": "ApiConnection"
                    }
                },
                "foreach": "@triggerBody()?['object']?['properties']?['Alerts']",
                "runAfter": {},
                "type": "Foreach"
            }
        },
        "contentVersion": "1.0.0.0",
        "outputs": {},
        "parameters": {
            "$connections": {
                "defaultValue": {},
                "type": "Object"
            }
        },
        "triggers": {
            "When_Azure_Sentinel_incident_creation_rule_was_triggered": {
                "inputs": {
                    "body": {
                        "callback_url": "@{listCallbackUrl()}"
                    },
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "path": "/incident-creation"
                },
                "type": "ApiConnectionWebhook"
            }
        }
    },
```

```
    "parameters": {
        "$connections": {
            "value": {
                "azuread_1": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/azuread-2",
                    "connectionName": "azuread-2",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/azuread"
                },
                "azuresentinel": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/azuresentinel-Block-AADUser",
                    "connectionName": "azuresentinel-Block-AADUser",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/azuresentinel"
                },
                "office365": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/office365",
                    "connectionName": "office365",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/office365"
                }
            }
        }
    }
}
```

This code above will create the following steps in a playbook:

1. Trigger: Incident creation
2. Step 1: For each alert get the incident
3. Step 2: Get entities in alert (accounts)
4. Step 3: Send an email to an administrator with a link to disable the account

---

Important
Make sure the entity settings in the incident creation rule are configured with the Account parameter, else the playbook will not be able to retreive the user account

---

This playbook is added as an automation step in the “Detect Emergency BreakGlass logins” incident creation rule.

So after a login with an Emergency BreakGlass account is detected, an incident will be created, based on this incident the playbook will be executed, which will send an email to an administrator with an option to disable the user account.

Because Emergency BreakGlass users have the Global Administrator role assigned to them, it is not possible to disable the accounts directly from the logic app. This is only possible on regular non-admin user accounts. The administrator will have to do this manually.

A second playbook will be used to automatically respond to the “Privileged user login” incident trigger. The users that are detected in this playbook will be automatically disabled through an option given in an email that gets sent out. Because

these users will not have the Global Administrator role assigned to them, the Logic app is able to disable the accounts without the need for manual intervention. A second option will be given in the email to ignore and close the incident.

The playbook (logic app) is created with the following Terraform code.

```
resource "azurerm_logic_app_workflow" "PrivilegedUser" {
  name                = "PrivilegedUser"
  location            = azurerm_resource_group.rg_Security.location
  resource_group_name = azurerm_resource_group.rg_Security.name
}
```

The playbook is configured with the following json code.

```
{
    "definition": {
        "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-
01/workflowdefinition.json#",
        "actions": {
            "Alert_-_Get_accounts": {
                "inputs": {
                    "body": "@triggerBody()?['object']?['properties']?['relatedEntities']",
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "method": "post",
                    "path": "/entities/account"
                },
                "runAfter": {
                    "For_each_2": [
                        "Succeeded"
                    ]
                },
                "type": "ApiConnection"
            },
            "For_each": {
                "actions": {
                    "Condition": {
                        "actions": {
                            "For_each_3": {
                                "actions": {
                                    "Get_user": {
                                        "inputs": {
                                            "host": {
                                                "connection": {
                                                    "name": "@parameters('$connections')['azuread_
1']['connectionId']"
                                                }
                                            },
                                            "method": "get",
                                            "path": "/v1.0/users/@{encodeURIComponent(concat(items
('For_each_3')?['Name'], '@', 'stagesien.onmicrosoft.com'))}"
                                        },
                                        "runAfter": {},
                                        "type": "ApiConnection"
```

```
                                    },
                                    "Update_user_2": {
                                        "inputs": {
                                            "body": {
                                                "accountEnabled": false
                                            },
                                            "host": {
                                                "connection": {
                                                    "name": "@parameters('$connections')['azuread_
1']['connectionId']"
                                                }
                                            },
                                            "method": "patch",
                                            "path": "/v1.0/users/@{encodeURIComponent(body('Get_us
er')?['id'])}"
                                        },
                                        "runAfter": {
                                            "Get_user": [
                                                "Succeeded"
                                            ]
                                        },
                                        "type": "ApiConnection"
                                    }
                                },
                                "foreach": "@body('Alert_-_Get_accounts')?['Accounts']",
                                "runAfter": {},
                                "type": "Foreach"
                            }
                        },
                        "else": {
                            "actions": {
                                "For_each_4": {
                                    "actions": {
                                        "Alert_-_Get_incident_2": {
                                            "inputs": {
                                                "host": {
                                                    "connection": {
                                                        "name": "@parameters('$connections')['azur
esentinel']['connectionId']"
                                                    }
                                                },
                                                "method": "get",
                                                "path": "/Incidents/subscriptions/@{encodeURICompo
nent(triggerBody()?['workspaceInfo']?['SubscriptionId'])}/resourceGroups/@{encodeURIComponent(trig
gerBody()?['workspaceInfo']?['ResourceGroupName'])}/workspaces/@{encodeURIComponent(triggerBody()?
['workspaceId'])}/alerts/@{encodeURIComponent(items('For_each_4')?['properties']?['systemAlertId']
)}"
                                            },
                                            "runAfter": {},
                                            "type": "ApiConnection"
                                        },
                                        "Change_incident_status_(to_be_deprecated)_2": {
                                            "inputs": {
                                                "host": {
                                                    "connection": {
                                                        "name": "@parameters('$connections')['azur
```

```
esentinel']['connectionId']"
                                                    }
                                                },
                                                "method": "put",
                                                "path": "/Case/@{encodeURIComponent(triggerBody()?
['workspaceInfo']?['SubscriptionId'])}/@{encodeURIComponent(triggerBody()?['workspaceId'])}/@{enco
deURIComponent(triggerBody()?['workspaceInfo']?['ResourceGroupName'])}/@{encodeURIComponent('Incid
ent')}/@{encodeURIComponent(body('Alert_-_Get_incident_2')?['properties']?['incidentNumber'])}/Sta
tus/@{encodeURIComponent('Closed')}"
                                            },
                                            "runAfter": {
                                                "Alert_-_Get_incident_2": [
                                                    "Succeeded"
                                                ]
                                            },
                                            "type": "ApiConnection"
                                        }
                                    },
                                    "foreach": "@triggerBody()?['object']?['properties']?['Alerts'
]",
                                    "runAfter": {},
                                    "type": "Foreach"
                                }
                            }
                        },
                        "expression": {
                            "and": [
                                {
                                    "equals": [
                                        "@body('Send_approval_email')?['SelectedOption']",
                                        "Approve"
                                    ]
                                }
                            ]
                        },
                        "runAfter": {
                            "Send_approval_email": [
                                "Succeeded"
                            ]
                        },
                        "type": "If"
                    },
                    "Get_user_2": {
                        "inputs": {
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['azuread_1']['connectionI
d']"
                                }
                            },
                            "method": "get",
                            "path": "/v1.0/users/@{encodeURIComponent(concat(items('For_each')?['N
ame'], '@', 'stagesien.onmicrosoft.com'))}"
                        },
                        "runAfter": {},
                        "type": "ApiConnection"
```

```
                    },
                    "Send_approval_email": {
                        "inputs": {
                            "body": {
                                "Message": {
                                    "Body": "A suspicious login with the account @{items('For_each
')?['Name']} has been detected. Click \"Approve\" to disable this account if this was a valid inci
dent, click \"Reject\" to close the incident if this was an invalid incident. ",
                                    "HideHTMLMessage": false,
                                    "Importance": "High",
                                    "Options": "Approve, Reject",
                                    "ShowHTMLConfirmationDialog": false,
                                    "Subject": "Suspicious privileged user login detected",
                                    "To": "sien.van-broekhoven@hpe.com"
                                },
                                "NotificationUrl": "@{listCallbackUrl()}"
                            },
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['office365']['connectionI
d']"
                                }
                            },
                            "path": "/approvalmail/$subscriptions"
                        },
                        "runAfter": {
                            "Get_user_2": [
                                "Succeeded"
                            ]
                        },
                        "type": "ApiConnectionWebhook"
                    }
                },
                "foreach": "@body('Alert_-_Get_accounts')?['Accounts']",
                "runAfter": {
                    "Alert_-_Get_accounts": [
                        "Succeeded"
                    ]
                },
                "type": "Foreach"
            },
            "For_each_2": {
                "actions": {
                    "Alert_-_Get_incident": {
                        "inputs": {
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['azuresentinel']['connect
ionId']"
                                }
                            },
                            "method": "get",
                            "path": "/Cases/@{encodeURIComponent(items('For_each_2')?['properties'
]?['systemAlertId'])}/@{encodeURIComponent(triggerBody()?['workspaceInfo']?['SubscriptionId'])}/@{
encodeURIComponent(triggerBody()?['workspaceId'])}/@{encodeURIComponent(triggerBody()?['workspaceI
nfo']?['ResourceGroupName'])}"
```

```
                        },
                        "runAfter": {},
                        "type": "ApiConnection"
                    }
                },
                "foreach": "@triggerBody()?['object']?['properties']?['Alerts']",
                "runAfter": {},
                "type": "Foreach"
            }
        },
        "contentVersion": "1.0.0.0",
        "outputs": {},
        "parameters": {
            "$connections": {
                "defaultValue": {},
                "type": "Object"
            }
        },
        "triggers": {
            "When_Azure_Sentinel_incident_creation_rule_was_triggered": {
                "inputs": {
                    "body": {
                        "callback_url": "@{listCallbackUrl()}"
                    },
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "path": "/incident-creation"
                },
                "type": "ApiConnectionWebhook"
            }
        }
    },
    "parameters": {
        "$connections": {
            "value": {
                "azuread_1": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/azuread-2",
                    "connectionName": "azuread-2",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/azuread"
                },
                "azuresentinel": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/azuresentinel-Block-AADUser",
                    "connectionName": "azuresentinel-Block-AADUser",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/azuresentinel"
                },
                "office365": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/office365",
                    "connectionName": "office365",
```

```
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/office365"
                }
            }
        }
    }
}
```

This code above will create the following steps in a playbook:

1. Trigger: Incident creation
2. Step 1: For each alert get the incident
3. Step 2: Get entities in alert (accounts)
4. Step 3: Send an email to an administrator with 2 options: Approve or Reject
5. Step 4: Condition: if answer = Approve then the user will be disabled, if answer = Reject then the incident will be closed

Important
Make sure the entity settings in the incident creation rule are configured with the Account parameter, else the playbook will not be able to retreive the user account.

This logic app needs to be assigned permissions in Azure AD to be able to update users. To assign these permissions a managed identity is used, this can be enabled in the Identity tab of the logic app, here the status of “System assigned identity” needs to be switched on. Switching this setting on will register an application in Azure AD.

Afterwards the specific permissions have to be applied to the application. This can be done with the following PowerShell script.

```
# Replace with your managed identity object ID, can be found in the Identity tab of the logic app
$miObjectID = "XXXX"
$permissionsToAdd = "User.ReadWrite.All", "User.ManageIdentities.All", "Directory.ReadWrite.All"
# Microsoft Graph; the ID is the same in all tenants
$appId = "00000003-0000-0000-c000-000000000000"

Connect-AzureAD

$app = Get-AzureADServicePrincipal -Filter "AppId eq '$appId'"

foreach ($permission in $permissionsToAdd)
{
   $role = $app.AppRoles | where Value -Like $permission | Select-Object -First 1
   New-AzureADServiceAppRoleAssignment -Id $role.Id -ObjectId $miObjectID -PrincipalId $miObjectID
-ResourceId $app.ObjectId
}
```

Important
These permissions have to be assigned through a user account with Global administrator privileges or Azure AD administrator permissions.

The permissions of the application can be checked with the following script.

```
# Replace with your managed identity object ID
$miObjectID = "XXXX"
# Microsoft Graph; the ID is the same in all tenants
$appId = "00000003-0000-0000-c000-000000000000"

Connect-AzureAD

$app = Get-AzureADServicePrincipal -Filter "AppId eq '$appId'"

$appRoles = Get-AzureADServiceAppRoleAssignment -ObjectId $app.ObjectId | where PrincipalId -eq $m
iObjectID

foreach ($appRole in $appRoles) {
    $role = $app.AppRoles | where Id -eq $appRole.Id | Select-Object -First 1
    write-host $role.Value
}
```

When these permissions are set, the playbook can be connected to the incident creation rule, in the Automated response step of the configuration.

A third playbook was made to respond to the “Critical system downtime” incident. This playbook will send out an email to an administrator with two options: Approve to bring the virtual machine back online, and Reject to ignore and close the incident.

The playbook (logic app) is created with the following Terraform code.

```
resource "azurerm_logic_app_workflow" "CriticalSystemDown" {
  name                = "CriticalSystemDown"
  location            = azurerm_resource_group.rg_Security.location
  resource_group_name = azurerm_resource_group.rg_Security.name
}
```

The json code for this playbook is the following.

```
{
    "definition": {
        "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-
01/workflowdefinition.json#",
        "actions": {
            "Entities_-_Get_Hosts": {
                "inputs": {
                    "body": "@triggerBody()?['object']?['properties']?['relatedEntities']",
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "method": "post",
                    "path": "/entities/host"
                },
                "runAfter": {
                    "For_each_2": [
                        "Succeeded"
                    ]
```

```
                },
                "type": "ApiConnection"
            },
            "For_each": {
                "actions": {
                    "Condition": {
                        "actions": {
                            "Start_virtual_machine": {
                                "inputs": {
                                    "host": {
                                        "connection": {
                                            "name": "@parameters('$connections')['azurevm_1']['con
nectionId']"
                                        }
                                    },
                                    "method": "post",
                                    "path": "/subscriptions/@{encodeURIComponent(triggerBody()?['w
orkspaceInfo']?['SubscriptionId'])}/resourcegroups/@{encodeURIComponent(concat('rg_', items('For_e
ach')?['NetBiosName']))}/providers/Microsoft.Compute/virtualMachines/@{encodeURIComponent(items('F
or_each')?['NetBiosName'])}/start",
                                    "queries": {
                                        "api-version": "2019-12-01"
                                    }
                                },
                                "runAfter": {},
                                "type": "ApiConnection"
                            }
                        },
                        "else": {
                            "actions": {
                                "Change_incident_status_(to_be_deprecated)": {
                                    "inputs": {
                                        "host": {
                                            "connection": {
                                                "name": "@parameters('$connections')['azuresentine
l']['connectionId']"
                                            }
                                        },
                                        "method": "put",
                                        "path": "/Case/@{encodeURIComponent(triggerBody()?['worksp
aceInfo']?['SubscriptionId'])}/@{encodeURIComponent(triggerBody()?['workspaceId'])}/@{encodeURICom
ponent(triggerBody()?['workspaceInfo']?['ResourceGroupName'])}/@{encodeURIComponent('Incident')}/@
{encodeURIComponent(triggerBody()?['object']?['properties']?['incidentNumber'])}/Status/@{encodeUR
IComponent('Closed')}"
                                    },
                                    "runAfter": {},
                                    "type": "ApiConnection"
                                }
                            }
                        },
                        "expression": {
                            "and": [
                                {
                                    "equals": [
                                        "@body('Send_approval_email')?['SelectedOption']",
                                        "Approve"
```

```
                                    ]
                                }
                            ]
                        },
                        "runAfter": {
                            "Send_approval_email": [
                                "Succeeded"
                            ]
                        },
                        "type": "If"
                    },
                    "Send_approval_email": {
                        "inputs": {
                            "body": {
                                "Message": {
                                    "Body": "System @{items('For_each')?['NetBiosName']} has been
detected to be offline. Click \"Approve\" to bring this system back online if this was a valid inc
ident, click \"Reject\" to close the incident if this was an invalid incident. ",
                                    "HideHTMLMessage": false,
                                    "Importance": "High",
                                    "Options": "Approve, Reject",
                                    "ShowHTMLConfirmationDialog": false,
                                    "Subject": "Critical system downtime",
                                    "To": "sien.van-broekhoven@hpe.com"
                                },
                                "NotificationUrl": "@{listCallbackUrl()}"
                            },
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['office365']['connectionI
d']"
                                }
                            },
                            "path": "/approvalmail/$subscriptions"
                        },
                        "runAfter": {},
                        "type": "ApiConnectionWebhook"
                    }
                },
                "foreach": "@body('Entities_-_Get_Hosts')?['Hosts']",
                "runAfter": {
                    "Entities_-_Get_Hosts": [
                        "Succeeded"
                    ]
                },
                "type": "Foreach"
            },
            "For_each_2": {
                "actions": {
                    "Alert_-_Get_incident": {
                        "inputs": {
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['azuresentinel']['connect
ionId']"
                                }
```

```
                            },
                            "method": "get",
                            "path": "/Cases/@{encodeURIComponent(items('For_each_2')?['properties'
]?['systemAlertId'])}/@{encodeURIComponent(triggerBody()?['workspaceInfo']?['SubscriptionId'])}/@{
encodeURIComponent(triggerBody()?['workspaceId'])}/@{encodeURIComponent(triggerBody()?['workspaceI
nfo']?['ResourceGroupName'])}"
                        },
                        "runAfter": {},
                        "type": "ApiConnection"
                    }
                },
                "foreach": "@triggerBody()?['object']?['properties']?['Alerts']",
                "runAfter": {},
                "type": "Foreach"
            }
        },
        "contentVersion": "1.0.0.0",
        "outputs": {},
        "parameters": {
            "$connections": {
                "defaultValue": {},
                "type": "Object"
            }
        },
        "triggers": {
            "When_Azure_Sentinel_incident_creation_rule_was_triggered": {
                "inputs": {
                    "body": {
                        "callback_url": "@{listCallbackUrl()}"
                    },
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "path": "/incident-creation"
                },
                "type": "ApiConnectionWebhook"
            }
        }
    },
    "parameters": {
        "$connections": {
            "value": {
                "azuresentinel": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/azuresentinel-Block-AADUser",
                    "connectionName": "azuresentinel-Block-AADUser",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/azuresentinel"
                },
                "azurevm_1": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/azurevm-1",
                    "connectionName": "azurevm-1",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
```

```
dApis/azurevm"
                },
                "office365": {
                    "connectionId": "/subscriptions/XXXX/resourceGroups/rg_Security/providers/Micr
osoft.Web/connections/office365",
                    "connectionName": "office365",
                    "id": "/subscriptions/XXXX/providers/Microsoft.Web/locations/westeurope/manage
dApis/office365"
                }
            }
        }
    }
}
```

This code above will create the following steps in a playbook:

1. Trigger: Incident creation
2. Step 1: For each alert get the incident
3. Step 2: Get entities in alert (hosts)
4. Step 3: Send an email to an administrator with 2 options: Approve or Reject
5. Step 4: Condition: if answer = Approve then the machine will be started, if answer = Reject then the incident will be closed

This playbook makes use of a managed identity to acquire the proper permissions to start up the machine. This managed identity has to be enabled in the Identity tab of the logic app, here the status of “System assigned identity” needs to be switched on. Then the role “Virtual Machine Contributor” has to be assigned to the identity in the Azure role assignments tab.

Note
This playbook will only work for virtual machines because it makes use of an Azure function to start the machines up again.

This playbook will only detect when a machine is in a down state, however it will not detect when a machine is not properly working because of an error. To also cover that issue, a last playbook is created that will report all errors that are detected on critical systems to an administrator through email.

The playbook (logic app) is created with the following Terraform code.

```
resource "azurerm_logic_app_workflow" "CriticalSystemError" {
  name                = "CriticalSystemError"
  location            = azurerm_resource_group.rg_Security.location
  resource_group_name = azurerm_resource_group.rg_Security.name
}
```

The json code for this playbook is the following.

```
{
    "definition": {
        "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-
01/workflowdefinition.json#",
        "actions": {
            "Entities_-_Get_Hosts": {
                "inputs": {
                    "body": "@triggerBody()?['object']?['properties']?['relatedEntities']",
```

```
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "method": "post",
                    "path": "/entities/host"
                },
                "runAfter": {
                    "For_each": [
                        "Succeeded"
                    ]
                },
                "type": "ApiConnection"
            },
            "For_each": {
                "actions": {
                    "Alert_-_Get_incident": {
                        "inputs": {
                            "host": {
                                "connection": {
                                    "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                                }
                            },
                            "method": "get",
                            "path": "/Incidents/subscriptions/@{encodeURIComponent(triggerBody()?['workspaceInfo']?['SubscriptionId'])}/resourceGroups/@{encodeURIComponent(triggerBody()?['workspaceInfo']?['ResourceGroupName'])}/workspaces/@{encodeURIComponent(triggerBody()?['workspaceId'])}/alerts/@{encodeURIComponent(items('For_each')?['properties']?['systemAlertId'])}"
                        },
                        "runAfter": {},
                        "type": "ApiConnection"
                    }
                },
                "foreach": "@triggerBody()?['object']?['properties']?['Alerts']",
                "runAfter": {},
                "type": "Foreach"
            },
            "For_each_2": {
                "actions": {
                    "For_each_3": {
                        "actions": {
                            "Parse_JSON_2": {
                                "inputs": {
                                    "content": "@items('For_each_3')?['properties']?['additionalData']?['Custom Details']",
                                    "schema": {
                                        "properties": {
                                            "Description": {
                                                "items": {
                                                    "type": "string"
                                                },
                                                "type": "array"
                                            },
                                            "EventID": {
```

```
                                            "items": {
                                                "type": "string"
                                            },
                                            "type": "array"
                                        },
                                        "EventLog": {
                                            "items": {
                                                "type": "string"
                                            },
                                            "type": "array"
                                        },
                                        "Source": {
                                            "items": {
                                                "type": "string"
                                            },
                                            "type": "array"
                                        }
                                    },
                                    "type": "object"
                                }
                            },
                            "runAfter": {},
                            "type": "ParseJson"
                        },
                        "Send_an_email_(V2)_2": {
                            "inputs": {
                                "body": {
                                    "Body": "<p>The following error was detected on system @{i
tems('For_each_2')?['NetBiosName']}:<br>\nEventID: @{body('Parse_JSON_2')?['EventID']}<br>\nEventL
og: @{body('Parse_JSON_2')?['EventLog']}<br>\nDescription: @{body('Parse_JSON_2')?['Description']}
<br>\nSource: @{body('Parse_JSON_2')?['Source']}<br>\n</p>",
                                    "Importance": "High",
                                    "Subject": "Error detected on critical system - @{items('F
or_each_2')?['NetBiosName']}",
                                    "To": "sien.van-broekhoven@hpe.com"
                                },
                                "host": {
                                    "connection": {
                                        "name": "@parameters('$connections')['office365']['con
nectionId']"
                                    }
                                },
                                "method": "post",
                                "path": "/v2/Mail"
                            },
                            "runAfter": {
                                "Parse_JSON_2": [
                                    "Succeeded"
                                ]
                            },
                            "type": "ApiConnection"
                        }
                    },
                    "foreach": "@triggerBody()?['object']?['properties']?['Alerts']",
                    "runAfter": {},
                    "type": "Foreach"
```

```
                    }
                },
                "foreach": "@body('Entities_-_Get_Hosts')?['Hosts']",
                "runAfter": {
                    "Entities_-_Get_Hosts": [
                        "Succeeded"
                    ]
                },
                "type": "Foreach"
            }
        },
        "contentVersion": "1.0.0.0",
        "outputs": {},
        "parameters": {
            "$connections": {
                "defaultValue": {},
                "type": "Object"
            }
        },
        "triggers": {
            "When_Azure_Sentinel_incident_creation_rule_was_triggered": {
                "inputs": {
                    "body": {
                        "callback_url": "@{listCallbackUrl()}"
                    },
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azuresentinel']['connectionId']"
                        }
                    },
                    "path": "/incident-creation"
                },
                "type": "ApiConnectionWebhook"
            }
        }
    },
    "parameters": {
        "$connections": {
            "value": {
                "azuresentinel": {
                    "connectionId": "/subscriptions/7471f1ab-7ddc-47ab-a294-d91899e5028c/resourceG
roups/rg_Security/providers/Microsoft.Web/connections/azuresentinel-Block-AADUser",
                    "connectionName": "azuresentinel-Block-AADUser",
                    "id": "/subscriptions/7471f1ab-7ddc-47ab-a294-d91899e5028c/providers/Microsoft
.Web/locations/westeurope/managedApis/azuresentinel"
                },
                "office365": {
                    "connectionId": "/subscriptions/7471f1ab-7ddc-47ab-a294-d91899e5028c/resourceG
roups/rg_Security/providers/Microsoft.Web/connections/office365",
                    "connectionName": "office365",
                    "id": "/subscriptions/7471f1ab-7ddc-47ab-a294-d91899e5028c/providers/Microsoft
.Web/locations/westeurope/managedApis/office365"
                }
            }
        }
    }
```

```
}
```

This code above will create the following steps in a playbook:

1. Trigger: Incident creation
2. Step 1: For each alert get the incident
3. Step 2: Get entities in alert (hosts)
4. Step 3: Send an email to an administrator with the info about the error

This playbook is added as an automation step in the “Detect errors” incident creation rule.

Because of this automated response, administrators are quickly notified about errors on critical systems.

Securing WinRM

WinRM is used to execute remote configurations on the machines in the environment. Opening WinRM ports to be accessed by everyone is not very secure, we should limit the access to WinRM to only allow incoming WinRM connections from our ManagementVMs. This adds an extra layer of security, because users will have to authenticate with the ManagementVM first before they can access other servers.

On top of limiting the incoming connections to the ManagementVMs, we should also secure the data that is transmitted over this connection. A way to do that is by using SSL certificates.

AD CS authority

For my lab I will make use of the AD certificate authority that was configured on the DC before.

Note
Any certificate authority can be used to make these configurations, so if an organization already has a CA they can use that CA.

Certificates have to be made for every server in the domain. They have to be requested per server on the AD CS portal, following the same routine as in AD Certificates.

The following PowerShell script creates a server authentication certificate request for a server. It can be remotely executed to the correct machine over RemotePS. The CertName variable and the invoke-command target have to be replaced with the FQDN of the server.

```
Write-Host "Creating CertificateRequest(CSR) for $CertName `r "

Invoke-Command -ComputerName DC.stagesien.com -ScriptBlock {
$myFQDN=(Get-WmiObject win32_computersystem).DNSHostName+"."+(Get-WmiObject win32_computersystem).
Domain

$CertName = $myFQDN
$CSRPath = "c:\$($CertName).csr"
$INFPath = "c:\$($CertName).inf"
$Signature = '$Windows NT$'


$INF =
@"
[Version]
Signature= "$Signature"

[NewRequest]
```

```
Subject = "CN=$CertName"
KeyLength = 2048
Exportable = FALSE
MachineKeySet = TRUE
SMIME = False
PrivateKeyArchive = FALSE
UserProtected = FALSE
UseExistingKeySet = FALSE
ProviderName = "Microsoft RSA SChannel Cryptographic Provider"
ProviderType = 12
RequestType = PKCS10
KeyUsage = 0xa0

[EnhancedKeyUsageExtension]

OID=1.3.6.1.5.5.7.3.1
"@

write-Host "Certificate Request is being generated `r "
$INF | out-file -filepath $INFPath -force
certreq -new $INFPath $CSRPath

}

write-output "Certificate Request has been generated"
```

Then submit the request with the following command:

```
certreq -submit c:\%computername%.%userdnsdomain%.csr
```

On the CA (dc.stagesien.com in my lab) open certsrv.msc and issue the request in the certification manager.

Then back on the server download and install the approved certificate with the following commands:

```
certreq -retrieve 1 c:\%computername%.%userdnsdomain%.cer
certreq -accept c:\%computername%.%userdnsdomain%.cer -machine
```

Note
Replace “1” with the ID of the certificate request.

“%computername%.%userdnsdomain%.cer” will save the certificate request on the C drive as a .cer file with as name the FQDN of the server.

The steps above can also be executed manually with the configuration steps below.

```
Browse to http://DC.stagesien.com/certsrv/certrqxt.asp on the remote machine
Request a certificate - Advanced request - Submit a request...
Paste the content of the previously generated certificate request in the text box and submit the r
equest. The certificate request can be found on the C drive.

On the CA (dc.stagesien.com in my lab) open certsrv.msc and issue the request

Back on the original server browse to http://DC.stagesien.com/certsrv/certckpn.asp - Download the
```

```
issued certificate
Install the certificate to Local Machine in the Personal store
```

Lastly execute the following PS script to configure the WinRM HTTPS Listener to use the previously generated certificate, and deleted the insecure HTTP listener. It will also create a firewall exception for port 5986, on which WinRM runs.

```
## Create the HTTPS listener
winrm quickconfig -transport:https

## Delete the HTTP listener
winrm delete winrm/config/Listener?Address=*+Transport=HTTP

## Create a firewall rule to only allow WinRM connections from ManagementVM0
$FirewallParam = @{
     DisplayName = 'Windows Remote Management (HTTPS-In)'
     Direction = 'Inbound'
     LocalPort = 5986
     Protocol = 'TCP'
     Action = 'Allow'
     Program = 'System'
     RemoteAddress = '10.1.2.4'
 }
 New-NetFirewallRule @FirewallParam
```

The “RemoteAddress” parameter specifies the IP address from which connections to 5986 are allowed. Setting this parameter to the IP address of the managementVM limits connections to that machine.

Important
Set the RemoteAddress parameter to the machine from which you will run WAC. Include the IP addresses of both machines when using an availability cluster.

If the IP addresses of the ManagementVMs ever change, a new rule can be created, to afterwards remove the previous firewall rule. This can be configured with the following PowerShell script

```
## Replace NEW-IP with the new IP address(es)
$FirewallParam = @{
     DisplayName = 'Windows Remote Management – new IP (HTTPS-In)'
     Direction = 'Inbound'
     LocalPort = 5986
     Protocol = 'TCP'
     Action = 'Allow'
     Program = 'System'
     RemoteAddress = 'NEW-IP'
 }
 New-NetFirewallRule @FirewallParam

Remove-NetFirewallRule -DisplayName "Windows Remote Management (HTTPS-In)"
```

Important
Do not remove the first rule before creating a new rule, you might lock yourself out. Create a new rule first.

Testing the Powershell connection can be done with the following command:

```
Enter-PSSession -ComputerName DC.stagesien.com -UseSSL -Credential (Get-Credential)
```

These previous steps (creating a certificate request, installing the issued request, adding the HTTPS listener, creating the firewall rule) have to be configured on every machine in the domain.

Make sure to force the windows Admin Center gateway to use WinRM over HTTPS, else it will not be able to connect to the servers anymore. This can be configured with the following commands on the gateway server.

```
Set-ItemProperty -path "hklm:\SOFTWARE\Microsoft\ServerManagementGateway" -name WinRMHTTPS -value
1
Restart-Service -Name ServerManagementGateway
```

Securing RDP

Just like WinRM, RDP connections should not be allowed from any other machine than the ManagementVMs. Users will have to authenticate with the ManagementVMs first before being able to Remote Desktop Protocol into the other machines.

This can be configured with the following PowerShell script, which can be executed remotely from the ManagementVM.

```
$Username = "stagesien\sien"
$SecurePassword = ConvertTo-SecureString "XXXX" -AsPlainText -Force
$credentials = New-Object System.Management.Automation.PSCredential($Username, $SecurePassword)

## Configure On-Prem-DC
$conn = New-PSSession -ComputerName On-Prem-DC.stagesien.com -Credential (Get-Credential $credenti
als) -UseSSL
Invoke-Command -Session $conn -ScriptBlock{
$FirewallParam = @{
     DisplayGroup = 'Remote Desktop'
     RemoteAddress = '10.1.2.4'
 }
Set-NetFirewallRule @FirewallParam
}

## Configure On-Prem-ADC
$conn = New-PSSession -ComputerName On-Prem-ADC.stagesien.com -Credential (Get-Credential $credent
ials) -UseSSL
Invoke-Command -Session $conn -ScriptBlock{
$FirewallParam = @{
     DisplayGroup = 'Remote Desktop'
     RemoteAddress = '10.1.2.4'
 }
Set-NetFirewallRule @FirewallParam
}

## Configure DC
$conn = New-PSSession -ComputerName DC.stagesien.com -Credential (Get-Credential $credentials) -Us
eSSL
Invoke-Command -Session $conn -ScriptBlock{
$FirewallParam = @{
     DisplayGroup = 'Remote Desktop'
     RemoteAddress = '10.1.2.4'
 }
Set-NetFirewallRule @FirewallParam
```

```
}

## Configure ManagementVM0
$conn = New-PSSession -ComputerName ManagementVM0.stagesien.com -Credential (Get-Credential $crede
ntials) -UseSSL
Invoke-Command -Session $conn -ScriptBlock{
$FirewallParam = @{
     DisplayGroup = 'Remote Desktop'
     RemoteAddress = '10.1.2.4'
 }
Set-NetFirewallRule @FirewallParam
}
```

Important
Replace the IP address in the RemoteAddress parameter with the IP address(es) of the VM(s) running WAC. Make sure not to lock yourself out.

I did not limit the RDP port of the client in my lab environment because I need to be able to RDP to this machine to log on. For clients in a production environment it is recommended to limit the allowed IP addresses to the IPs of the machines that need to be able to log on remotely through RDP (such as employees working from home, management servers).

9. Backups

Backups are critical components in any IT environment. If a system were to fail, backups make it possible to restore that system to a saved state of the past.

Azure backup is a solution that can back up files, folders, and system state of on-premise machines; and entire windows/linux virtual machines. It saves the backed-up data in vaults.

Note
If an organization uses a different solution to backup on-premise systems, this can be used in addition to Azure backup to backup cloud systems/data. Or the organization can fully switch to Azure backup to use for both their on-premise and cloud environment.

There are three different types of backups:

1. **Full backup**: This backup backs up the entire data source. Azure backup uses this backup type to create the initial backup.
2. **Differential**: A differential backup backs up the changes since the initial full backup. Azure backup does not use this backup type.
3. **Incremental**: An incremental backup stores only the data that has changed since the previous backup.

Backup on-premise systems

To backup on-premise Windows machines to Azure, the MARS (Microsoft Azure Recovery Services) agent can be used.

Note
Linux machines are not supported for this agent. Machines running the Linux OS will first have to be backed up to System Center Data Protection Manager (DPM) or Microsoft Azure Backup Server (MABS), then this backup server can be backed up to a Recovery Services vault in Azure.

To backup on-premise systems, first a recovery services vault has to be created. This can be done by creating a new Recovery Services Vault instance through the Azure portal. Afterwards the MARS agent has to be downloaded, this can be done by going to the dashboard of the previously created vault, selecting backup, and selecting “On-Premises” in this new window.

Then click on “Prepare infrastructure” to download the Windows Server or Windows Client agent. Like the name suggests, Windows Servers use the Windows Server agent. If you want to backup client machines as well, this can be achieved with the Windows Client agent. Make sure to also download the Vault credentials file, this file will be needed to connect to the MARS agents with the Recovery Services vault. When these files are downloaded, the agent can be installed on the required servers.

Note
To easily deploy this agent on multiple machines, download and place the required files on a shared drive on one machine and use a GPO to distribute and install it on the other machines in the environment.

Backups can be scheduled through the agent. In the “Schedule backup” pane.

Backup Azure virtual machines

Azure VMs can be backed up directly. Azure backup will install an extension to the Azure VM agent that's running on the VM. This extension backs up the entire VM.

A backup policy can be configured to specify the backup frequency, and retention. This policy can be created through Terraform.

In the script below is the code to create a vault to save backups to, a policy to backup machines every day at 11pm, and the code to attach this policy to the previously created VMs.

```
## Create a vault to save backups to
resource "azurerm_recovery_services_vault" "BackupVault" {
  name                = "BackupVault"
  location            = azurerm_resource_group.rg_Security.location
  resource_group_name = azurerm_resource_group.rg_Security.name
  sku                 = "Standard"
}

## Create a policy to backup systems daily at 11pm
resource "azurerm_backup_policy_vm" "BackupPolicy" {
  name                = "BackupPolicy"
  resource_group_name = azurerm_resource_group.rg_Security.name
  recovery_vault_name = azurerm_recovery_services_vault.r_Security.name

  timezone = "UTC"

  backup {
    frequency = "Daily"
    time      = "23:00"
  }
}

## The On-Prem machines are simulated for my lab, in reality on-premise systems cannot be configur
ed like this, but will instead use the MARS agent.
resource "azurerm_backup_protected_vm" "On-Prem-DC" {
  resource_group_name = azurerm_resource_group.rg_Security.name
  recovery_vault_name = azurerm_recovery_services_vault.BackupVault.name
  source_vm_id        = azurerm_virtual_machine.On-Prem-DC.id
  backup_policy_id    = azurerm_backup_policy_vm.BackupPolicy.id
}

resource "azurerm_backup_protected_vm" "On-Prem-ADC" {
  resource_group_name = azurerm_resource_group.rg_Security.name
  recovery_vault_name = azurerm_recovery_services_vault.BackupVault.name
  source_vm_id        = azurerm_virtual_machine.On-Prem-ADC.id
  backup_policy_id    = azurerm_backup_policy_vm.BackupPolicy.id
}

resource "azurerm_backup_protected_vm" "DC" {
  resource_group_name = azurerm_resource_group.rg_Security.name
  recovery_vault_name = azurerm_recovery_services_vault.BackupVault.name
  source_vm_id        = azurerm_virtual_machine.DC.id
  backup_policy_id    = azurerm_backup_policy_vm.BackupPolicy.id
}

resource "azurerm_backup_protected_vm" "ManagementVM0" {
```

```
  resource_group_name = azurerm_resource_group.rg_Security.name
  recovery_vault_name = azurerm_recovery_services_vault.BackupVault.name
  source_vm_id        = azurerm_virtual_machine.ManagementVM0.id
  backup_policy_id    = azurerm_backup_policy_vm.BackupPolicy.id
}
```

Note
In a production environment, certain machines will need to be backed up more frequently than others. Different backup policies can be created to achieve this.

10. Azure DevOps

An Azure DevOps release pipeline can be used to automate the base configurations in this environment. This reduces administrative overhead, and delivers a clean overview of the different configurations that need to happen to set up the virtual evironment.

To deploy Terraform configurations through Azure DevOps, the state file needs to be hosted in a storage Account on Azure. This can be configured through the following Azure Powershell script.

```
#Create Resource Group
New-AzureRmResourceGroup -Name "rg_tfstate" -Location "westeurope"

#Create Storage Account
New-AzureRmStorageAccount -ResourceGroupName "rg_tfstate" -AccountName "stagesientfstate" -Locatio
n westeurope -SkuName Standard_LRS

#Create Storage Container
New-AzureRmStorageContainer -ResourceGroupName "rg_tfstate" -AccountName "stagesientfstate" -Conta
inerName "tfstatedevops"
```

Note
Hosting the state file in an Azure storage account is recommended whether you use Azure DevOps or not. This will form a centralized storage place for the state of the configurations, making it possible for multiple people to configure the environment with Terraform. The file automatically gets locked when it is in use by a person, and becomes available again when the configurations are made.

A service principal will have to be configured to connect the Azure DevOps pipeline to the Azure environment. This can be created with the following Azure CLI command:

```
az ad sp create-for-rbac --name StageSienDevOps
```

This Service Principal (SP) needs to have at least contributor access on the Azure environment.

This SP then has to be added in the Service Connections in the settings of the Azure DevOps.

The DevOps release pipeline will use code files of the Azure github and consist of multiple stages to build the solution. The github includes all files found in the /dev folder (terraform, powershell, xml).

The stages are:

1. **Destroy environment** - make sure the working environment is clean
2. **Create Azure AD tenant** - create a new Azure AD tenant in the Azure portal, or make sure you have one ready
3. **Deploy Terraform base environment** - deploy the terraform configuration files
4. **Create shared disk** - create a shared disk in Azure

Every stage has several tasks.

1. Destroy environment

In this stage the entire working environment will be destroyed to make it ready to deploy new resources. This is mainly done through Terraform, with a manual check at the end to confirm the working environment is clean.

This stage can be disabled if you don’t want the environment to be destroyed.

To use Terraform, the Terraform extension has to be installed in the Azure DevOps environment. This can be installed from the marketplace: https://marketplace.visualstudio.com/items?itemName=ms-devlabs.custom-terraform-tasks

This stage consists of 3 agent jobs and 1 agentless job. The configuration of the tasks can be found in .yaml format per stage and task.

Agent job

These jobs are run on an agent.

1. Install Terraform

This task installs Terraform on the agent.

```
steps:
- task: ms-devlabs.custom-terraform-tasks.custom-terraform-installer-task.TerraformInstaller@0
  displayName: 'Install Terraform 0.14.9'
  inputs:
    terraformVersion: 0.14.9
  enabled: true
```

2. Terraform: init

This task initializes the Terraform working environment. It uses the statefile which is hosted in Azure, and the Azure git repository to do this.

```
steps:
- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV1@0
  displayName: 'Terraform : init'
  inputs:
    workingDirectory: '$(System.DefaultWorkingDirectory)/_Stage-Sien/Code/dev'
    backendServiceArm: StageSienDevOps
    backendAzureRmResourceGroupName: 'rg_tfstate'
    backendAzureRmStorageAccountName: stagesientfstate
    backendAzureRmContainerName: tfstatedevops
    backendAzureRmKey: tf/terraform.tfstate
  enabled: true
```

3. Terraform: destroy

This task executes `Terraform destroy` on the agent, which destroys all resources configured through Terraform.

```
steps:
- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV1@0
  displayName: 'Terraform : destroy'
  inputs:
    command: destroy
    workingDirectory: '$(System.DefaultWorkingDirectory)/_Stage-Sien/Code/dev'
    environmentServiceNameAzureRM: StageSienDevOps
  enabled: true
```

Agentless job

Agentless jobs are not executed on an agent.

1. Validate destroy

This manual intervention job will pause the pipeline to confirm that all resources are properly deleted. The pipeline will resume with its tasks after the button is clicked.

```
steps:
- task: ManualIntervention@8
  displayName: 'Validate destroy'
  inputs:
    instructions: 'Make sure all resources are destroyed and environment is clean.'
  enabled: true
```

2. Create Azure AD tenant

This stage consists of one manual task. The purpose of this task is to pause the pipeline, so that the Azure AD tenant can be created in the Azure portal.

Agentless job

Agentless jobs are not executed on an agent.

1. Create Azure AD

```
steps:
- task: ManualIntervention@8
  displayName: 'Create Azure AD'
  inputs:
    instructions: |
     Create a new Azure AD tenant: stagesien.onmicrosoft.com
     Create a global admin user: sien@stagesien.onmicrosoft.com
     Reset password for global admin user
  enabled: true
```

3. Deploy Terraform base environment

In this stage the Terraform configuration files will be executed.

Agent job

These jobs are run on an agent.

1. Install Terraform

Identical to Install Terraform in Destroy Environment.

2. Terraform: init

Identical to Terraform: init in Destroy Environment.

3. Terraform: plan

In this task the Terraform configuration files will be tested before being applied, the output of this task will show what will be configured.

```
steps:
- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV1@0
  displayName: 'Terraform : plan'
  inputs:
    command: plan
    workingDirectory: '$(System.DefaultWorkingDirectory)/_Stage-Sien/Code/dev'
    environmentServiceNameAzureRM: StageSienDevOps
  enabled: true
```

4. Terraform: apply

The final task in this stage applies the Terraform configuration.

```
steps:
- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV1@0
  displayName: 'Terraform: apply'
  inputs:
    command: apply
    workingDirectory: '$(System.DefaultWorkingDirectory)/_Stage-Sien/Code/dev'
    environmentServiceNameAzureRM: StageSienDevOps
  enabled: true
```

4. Create shared disk

In this stage the shared disk is created on Azure and linked to the management VMs.

Agent job

These jobs are run on an agent.

1. Create shared disk

This task creates the shared disk in Azure and connects it to the management VMs.

```
steps:
- task: AzureCLI@2
  displayName: 'Create shared disk'
  inputs:
    azureSubscription: StageSienDevOps
    scriptType: bash
    scriptLocation: inlineScript
    inlineScript: |
     az disk create -g rg_Management -n ManagementDisk --size-gb 256 -l westeurope --sku Premium_L
RS --max-shares 2

     az vm disk attach -g rg_Management --vm-name ManagementVM0 --name ManagementDisk
     az vm disk attach -g rg_Management --vm-name ManagementVM1 --name ManagementDisk
  enabled: true
```

5. Additional configurations

This stage will list the additional configurations that have to be configured in the environment. These configurations cannot be automatically executed from the pipeline because of the lack of public IP addresses in the environment.

Agentless job

Agentless jobs are not executed on an agent.

1. Configurations

This task lists the configuration steps.

```
steps:
- task: ManualIntervention@8
  displayName: Configurations
  inputs:
    instructions: |
     1. Download and install Windows Admin Center on the HA cluster
     2. Configure Azure AD sync on On-Prem-ADC
     3. Implement RBAC, conditional access policies, privileged access
     4. Implement monitoring tools: Azure monitor, log analytics workspace, azure AD connect healt
h
     5. Implement security tools: LAPS, SSPR, Sentinel, secure WinRM & RDP
```

```
6. Configure backups
```